Foundation Gatsby Projects

Create Four Real Production Websites with Gatsby

—

Nabendu Biswas

Apress®

Foundation Gatsby Projects

Create Four Real Production Websites with Gatsby

Nabendu Biswas

Apress®

Foundation Gatsby Projects: Create Four Real Production Websites with Gatsby

Nabendu Biswas
Bangalore, India

ISBN-13 (pbk): 978-1-4842-6557-4 ISBN-13 (electronic): 978-1-4842-6558-1
https://doi.org/10.1007/978-1-4842-6558-1

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Louise Corrigan
Development Editor: James Markham
Coordinating Editor: Nancy Chen

Cover designed by eStudioCalamar

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 1 New York Plaza, New York, NY 10004. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book’s product page, located at www.apress.com/9781484265574. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

To my wife and kid. This book is affectionately dedicated.

Table of Contents

About the Author

Nabendu Biswas is a full-stack JavaScript developer who has been working in the IT industry for the past 15 years. He has worked for some of the world's top development firms and investment banks. He currently works as an Associate Architect at Innominds. He is also a passionate tech blogger who publishes on `thewebdev.tech` and is an all-round nerd, passionate about everything JavaScript, React, and Gatsby. You can find him on Twitter `@nabendu82`.

About the Technical Reviewer

Alexander Chinedu Nnakwue has a background in Mechanical Engineering from the University of Ibadan, Nigeria and has been a frontend developer for over three years. He has worked on both web and mobile technologies. He also has experience as a technical author, writer, and reviewer. He enjoys programming for the web, and occasionally, you can find him playing soccer. He was born in Benin City and is currently based in Lagos, Nigeria.

Introduction

I have done quite a bit of freelance work in WordPress development since 2011. The three things that I didn't like about WordPress at that time were that little coding knowledge was required, the sites were slow, and they were easily hacked.

The awesome static site generator GatsbyJS solves all of these problems. It is built with React, so you can utilize all your React knowledge. Plus, it uses the in-demand GraphQL, so you will work with that also. The sites are blazing fast and completely secure. You need a bit of ReactJS knowledge to work with Gatsby, but it adds so much to the React ecosystem. It has a large plugin system like WordPress, which adds functionality. It can be used with a wide range of backend systems, like CMS, Firebase, and many more. In this book, we will first create a simple site using only Gatsby. After that, we will use Stackbit to quickly build a Gatsby site. Then we will build a complex site with all features using the Contentful CMS. The last chapter shows you how to build a video chat site, similar to Skype but using the Twilio service.

CHAPTER 1

Creating an Agency Site

In this chapter, we will build a simple demo agency site (known as a *service company* in India). Although I could use one of the many starter kits available at the Gatsby site that offer complete CSS, I've decided to use a starter kit with minimal CSS so you can learn how to write your own. We are going to do the setup first, and then move on to the basic styles. After that, we will create the sections and pages on the site. Finally, we will deploy the site using Netlify.

The Setup

In this project you can use any IDE like VS code. Everything else we will install through `npm`. Let's start with Gatsby.

First install Gatsby globally by running the following command in your terminal.

```
npm install -g gatsby-cli
```

To create a new Gatsby site, run the following command in the terminal. This is the most basic Gatsby starter kit, with minimal Gatsby plugins installed (more on that later).

```
gatsby new agencyDemo https://github.com/gatsbyjs/gatsby-starter-hello-world
```

Now, go to the directory and run `gatsby develop`. The commands are shown in Listing 1-1.

Listing 1-1. The gatsby develop Command

```
cd agencyDemo
gatsby develop
```

The basic site is now up and running,[1] as shown in Figure 1-1.

[1]`http://localhost:8000/`

© Nabendu Biswas 2021
N. Biswas, *Foundation Gatsby Projects*, https://doi.org/10.1007/978-1-4842-6558-1_1

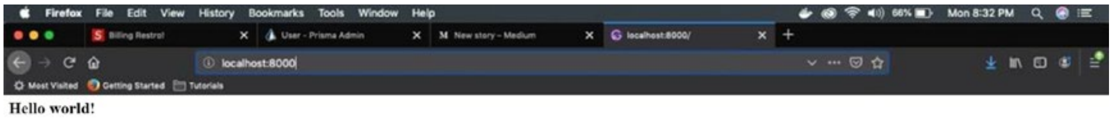


Figure 1-1. *Gatsby is up and running*

We will start by creating the home page.

Creating the Home Page

In Gatsby, everything is React-based, so we will create the home page component. We will first create a set of global styles.

So, first create a folder called `styles` inside `src`. Then create a file called `global.css` inside it. We will put the generic global CSS shown in Listing 1-2 inside this file.

Listing 1-2. The global.css File

```
html {
   box-sizing: border-box;
   font-size: 10px;
   font-weight: 400;
   letter-spacing: 0.075em;
   margin:0;
 }
 *, *:before, *:after {
   box-sizing: inherit;
 }
 body {
   font-family: "Open Sans", Helvetica, sans-serif;
   padding: 0;
   margin: 0;
 }
 a {
   text-decoration: none;
 }
```

Next, create a file called `gatsby-browser.js` in the root directory and include this `global.css` file by adding the following:

```
import "./src/styles/global.css"
```

We will use the extremely popular CSS-in-JS library *styled component* to style the rest of the project.

We need to install some dependencies for `styled-components`. So, open the terminal and type the following command. We have to stop the `gatsby develop` running on the terminal by pressing Ctrl+C.

```
npm install--save gatsby-plugin-styled-components styled-components babel-
plugin-styled-components
```

Next, include the following code in the `gatsby-config.js` file. This file should already be in the root folder.

```
module.exports = {
 plugins: [`gatsby-plugin-styled-components`],
}
```

Then restart `gatsby develop` in the terminal. Next, we will create our index or home page.

Creating the Index Page

We will now start with the index page. We will have a full-page image and some centered text on top of it. First change your `index.js file`, as shown in Listing 1-3.

Listing 1-3. The index.js File

```
import React, { Component } from "react";
import { Link } from "gatsby";
import { Banner, TextWrapper, MoreText } from "../styles/IndexStyles";

export default () => (
   <div style={{position: 'relative'}}>
       <Banner></Banner>
       <TextWrapper>
           <div>
```

```
                <h2>GeekyHacker</h2>
                <p>One Stop for<br/>
                All your development<br />
                And design needs</p>
                <Link to="/works">Our Works</Link>
            </div>
        </TextWrapper>
        <MoreText>Learn More</MoreText>
    </div>
)
```

Now, we will start to write the styled components. Create a file called `IndexStyles.js` inside the `styles` directory.

First we will write styles for the banner. We will show the `banner.jpg` file as a background image, so we will use the `:after` pseudo element.

Also, upload the `banner.jpg` image to the static folder. From the static folder in a Gatsby project, we can directly use an image (see Listing 1-4).

Listing 1-4. The IndexStyles.js File

```
import styled from "styled-components"
const Banner = styled.div`
 &:after {
   content: "";
   display: block;
   height: 100vh;
   width: 100%;
   background-image: url('banner.jpg');
   background-size: cover;
   background-repeat: no-repeat;
   background-position: center;
   filter: grayscale(100%) blur(2px);
 }
```

Next, we will center the text. We need to use the positioning system, as we are showing the text over an image. We already made the parent a `position: relative` and we are making this `div position: absolute`. Then we are using left, top, and transform (see Listing 1-5).

Listing 1-5. The position: absolute Setting

```
const TextWrapper = styled.div`
 position: absolute;
 z-index: 1;
 left: 50%;
 top: 50%;
 transform: translate(-50%, -50%);
 color: white;
 div {
   display: flex;
   justify-content: center;
   align-items: center;
   flex-direction: column;
 }
`;
```

Make sure you have the following at the bottom of the `IndexStyles.js` file or you will get an error.

```
export { Banner, TextWrapper, MoreText }
```

The result is shown in Figure 1-2.

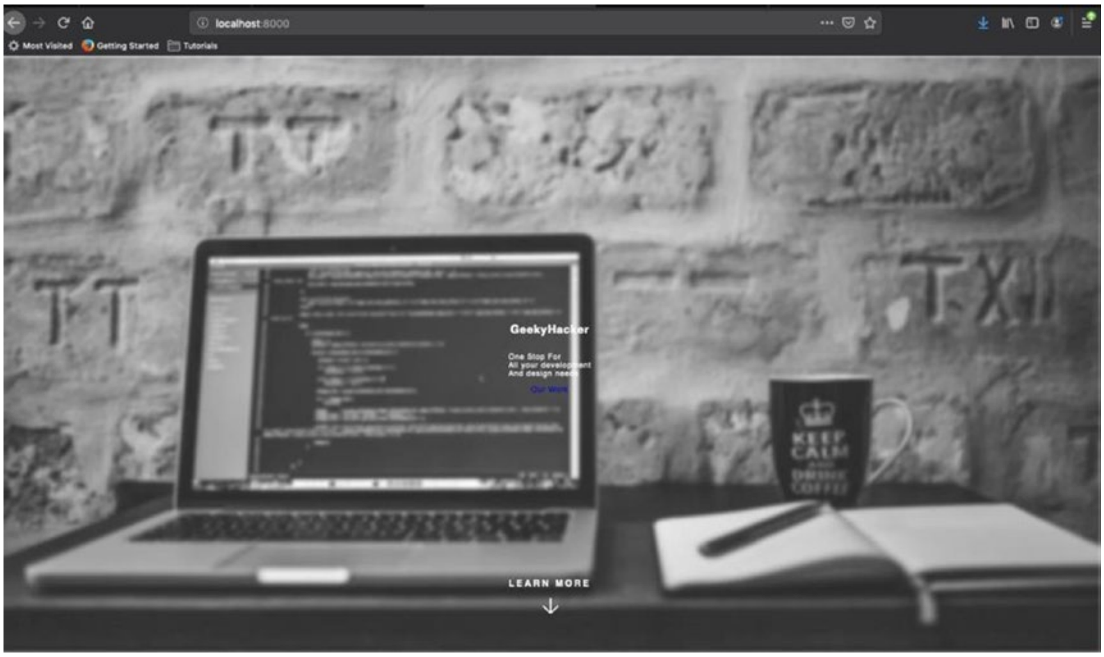


Figure 1-2. *Centered text with the image in the background*

Let's now add some styles to the h2, p, and a tags (a is a link tag that converts to an anchor tag). See Listing 1-6.

Listing 1-6. Link to the Anchor Tag

```
h2 {
   font-size: 5rem;
   opacity: 1;
   padding: 0.35em 1em;
   border-top: 2px solid white;
   border-bottom: 2px solid white;
   text-transform: uppercase;
   margin: 0;
 }
 p {
   text-transform: uppercase;
   text-align: center;
   letter-spacing: 0.225em;
```

```
    font-size: 2.5rem;
  }
  a {
    background-color: #ed4933;
    box-shadow: none;
    color: #ffffff;
    border-radius: 3px;
    border: 0;
    cursor: pointer;
    font-size: 1.5rem;
    font-weight: 600;
    letter-spacing: 0.225em;
    padding: 1.8rem 0.8rem;
    text-align: center;
    text-decoration: none;
    text-transform: uppercase;
  }
```

Let's add the style for the `Learn More text`. For this, we also use the `position: absolute` logic, as shown in Listing 1-7.

Listing 1-7. The `Learn More Text`

```
const MoreText = styled.div`
  position: absolute;
  color: #ffffff;
  text-align: center;
  text-transform: uppercase;
  letter-spacing: 0.225em;
  font-weight: 600;
  font-size: 1.2rem;
  z-index: 1;
  left: 50%;
  bottom: 10%;
  transform: translate(-50%, -50%);
```

```
 &:after {
   content: "";
   display: block;
   height: 2rem;
   width: 2rem;
   left: 50%;
   position: absolute;
   margin: 1em 0 0 -0.75em;
   background-image: url("arrow.svg");
   background-size: cover;
   background-repeat: no-repeat;
   background-position: center;
 }
`;

export { Banner, TextWrapper, MoreText };
```

This code will result in the beautiful page shown in Figure 1-3.

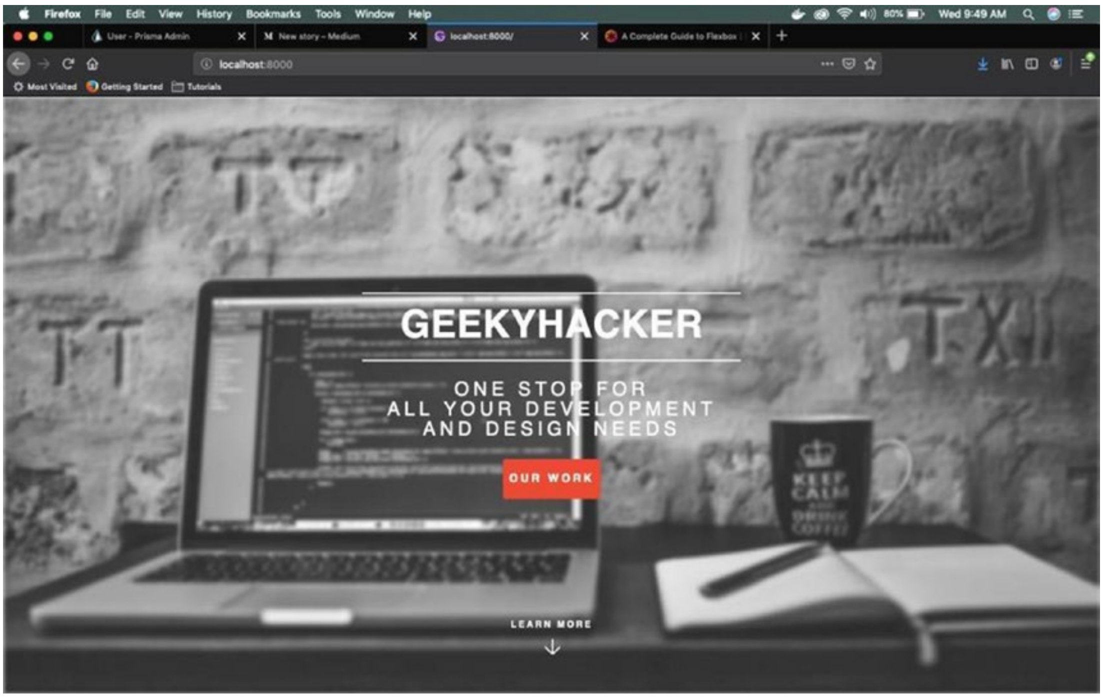


***Figure 1-3.** The beautiful page*

Creating the Sections

Let's start by creating section two.

Creating Section Two

This section will contain a header, a quotation, and three icons. Update the `index.js` file to include Listing 1-8. The new part is shown in bold.

Listing 1-8. Adding a Header to index.js

```
import React, { Component } from "react";
import { Link } from "gatsby";
import { Banner, TextWrapper, MoreText, SectionTwo } from "../styles/
IndexStyles";

export default () => (
   <>
   <section style={{position: 'relative'}}>
       <Banner></Banner>
       <TextWrapper>
           <div>
               <h2>GeekyHacker</h2>
               <p>One Stop for<br/>
               All your development<br />
               And design needs</p>
               <Link to="/works">Our Works</Link>
           </div>
       </TextWrapper>
       <MoreText>Learn More</MoreText>
   </section>
   <SectionTwo>
       <div>
           <h2>Our Passion</h2>
           <p>Most good programmers do programming not because they expect
           to get paid,
           but because it's fun to program.</p>
```

```
            <h5>- Linus Torvalds</h5>
        </div>
    </SectionTwo>

)
```

Now, let's add some styled components to SectionTwo, as shown in Listing 1-9.

Listing 1-9. Adding Styled Components to index.js

```
const SectionTwo = styled.section`
 background-color: #21b2a6;
 text-align: center;
 padding: 10rem 0;
 div {
   width: 66%;
   margin: 0 auto;
 }
 h2 {
   font-size: 3rem;
   padding: 1.35em 0;
   color: #ffffff;
   border-bottom: 2px solid #1d9c91;
   text-transform: uppercase;
   letter-spacing: 0.6rem;
   margin: 0;
 }
 p {
   text-transform: uppercase;
   color: #c8ece9;
   text-align: center;
   letter-spacing: 0.225em;
   font-size: 1.5rem;
 }
 h5 {
   font-size: 1.4rem;
   line-height: 2rem;
```

```
    color: #ffffff;
    border-bottom: 2px solid #1d9c91;
    font-weight: 800;
    letter-spacing: 0.225em;
    text-transform: uppercase;
    padding-bottom: 0.5rem;
    margin-bottom: 5rem;
  }
`
```

We will now use some `font-awesome` icons to give this section a nice finish. Open the terminal and install these dependencies.

```
npm install @fortawesome/react-fontawesome @fortawesome/fontawesome-svg-core
@fortawesome/free-solid-svg-icons
```

Then import the libraries in `index.js`, as shown in Listing 1-10.

Listing 1-10. Adding Icons to index.js

```
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { library } from '@fortawesome/fontawesome-svg-core';
import { faHeart, faCode, faGem, fas } from '@fortawesome/free-solid-svg-
icons';

library.add(faHeart, faCode, faGem, fab, fas);
```

Add the code in Listing 1-11, to the `div` and after the `h5`, and you will get the result shown in Figure 1-4.

Listing 1-11. Completing Section Two for index.js

```
                    <span>
                        <FontAwesomeIcon icon="gem" color="#04F5C6" size="6x"
                        style={{marginRight: '3rem'}} fixedWidth border />
                        <FontAwesomeIcon icon="heart" color="#00F0FF" size="6x"
                        style={{marginRight: '3rem'}} fixedWidth border />
                        <FontAwesomeIcon icon="code" color="#73DBFD" size="6x"
                        fixedWidth border />
                    </span>
```

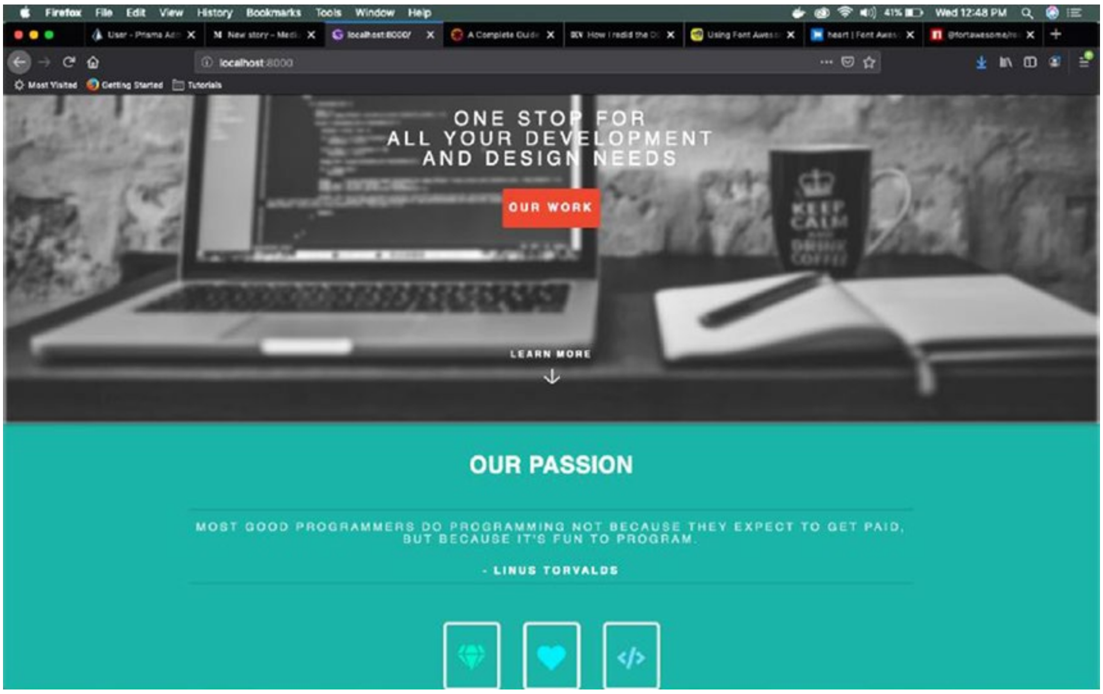

ONE STOP FOR
ALL YOUR DEVELOPMENT
AND DESIGN NEEDS
OUR WORK
LEARN MORE
OUR PASSION
MOST GOOD PROGRAMMERS DO PROGRAMMING NOT BECAUSE THEY EXPECT TO GET PAID, BUT BECAUSE IT'S FUN TO PROGRAM.
- LINUS TORVALDS

Figure 1-4. *Our passion*

We also need some padding in this section. Let's add it. It is shown in bold in Listing 1-12.

Listing 1-12. Adding Padding to IndexStyles.js

```
const SectionTwo = styled.section`
  background-color: #21b2a6;
  text-align: center;
  padding: 10rem 0;
  div {
    width: 66%;
    margin: 0 auto;
  }
...
...
`
```

Next, we will create Section Three.

Creating Section Three

This section will include an image and text, so we will use Flexbox. Add the code in `index.js`, as shown in Listing 1-13, after `SectionTwo`.

Listing 1-13. Adding Section Three to index.js

```
<SectionThree>
        <FlexBoxIndex>
            <div className="image">
                <img src="pic01.jpg" alt="pic01" />
            </div>
            <div className="text__section3">
                <h2>Website Development</h2>
                <p>
                    We hand code beautiful websites using HTML5,
                    CSS3, JS because they are fully customizable and
                    efficient. No WordPress websites here.
                </p>
            </div>
        </FlexBoxIndex>
        <FlexBoxIndex inverse>
            <div className="text__section3">
                <h2>Website Design</h2>
                <p>
                    We have talented and experienced Web Designers, who
                    can design beautiful and usable websites.
                </p>
            </div>
            <div className="image">
                <img src="pic02.jpg" alt="pic02" />
            </div>
        </FlexBoxIndex>
```

```
          <FlexBoxIndex>
              <div className="image">
                  <img src="pic03.jpg" alt="pic03" />
              </div>
              <div className="text__section3">
                  <h2>Mobile App Development</h2>
                  <p>
                      We develop Mobile apps in Reactive Native, which can
                      be used in both ios and Android.
                  </p>
              </div>
          </FlexBoxIndex>
      </SectionThree>
```

Now, let's add the code in Listing 1-14 to `IndexStyles.js`, just below `SectionTwo`.

Listing 1-14. Adding Section Three to the IndexStyles.js File

```
const SectionThree = styled.section`
 background-color: #2b343d;
 color: #ffffff;
`

const FlexBoxIndex = styled.div`
 display: flex;
 .image {
   width: ${props => (props.inverse ? "60%" : "40%")};
 }
 img {
   width: 100%;
 }
 .text__section3 {
   width: ${props => (props.inverse ? "40%" : "60%")};
   display: flex;
   justify-content: center;
```

```
    align-items: center;
    flex-direction: column;
  }
h2 {
    font-size: 3rem;
    color: #ffffff;
    text-transform: uppercase;
    letter-spacing: 0.225rem;
    margin: 0;
  }
  p {
    text-transform: uppercase;
    color: #c8ece9;
    text-align: center;
    letter-spacing: 0.075em;
    font-size: 1.5rem;
  }
  `
```

Let's do some housekeeping. In Listing 1-15, remove **p** from `TextWrapper`, `SectionTwo`, and `FlexBoxIndex`**.** Then create a new styled component component, called `props`.

Listing 1-15. Housekeeping in IndexStyles.js

```
const GenereicPara = styled.p`
 text-transform: uppercase;
 text-align: center;
 letter-spacing: ${props => (props.lessSpacing ? "0.075em" : "0.225em")};
 font-size: ${props => (props.lessSize ? "1.5rem" : "2.5rem")};
 line-height: ${props => (props.lessSize ? "2rem" : "3rem")};
 color: ${props => (props.grey ? "#c8ece9" : "#ffffff")};
```

Now replace all the `p` tags with `GenericPara`. The updated code is highlighted in bold in Listing 1-16.

Listing 1-16. The index.js File

```
<section style={{ position: 'relative' }}>
    <Banner></Banner>
    <TextWrapper>
        <div>
            <h2>GeekyHacker</h2>
            <GenereicPara>One Stop For<br />
                All your development<br />
                And design needs</GenereicPara>
            <Link to="/works">Our Work</Link>
        </div>
    </TextWrapper>
    <MoreText>Learn More</MoreText>
</section>
<SectionTwo>
    <div>
        <h2>Our Passion</h2>
        <GenereicPara lessSize grey>Most good programmers
        do programming not because they expect to get
        paid,<br />
            but because it's fun to program.</GenereicPara>
        <h5>- Linus Torvalds</h5>
    </div>
    <span>
        <FontAwesomeIcon icon="gem" color="#04F5C6"
        size="6x" style={{marginRight: '3rem'}} fixedWidth
        border />
        <FontAwesomeIcon icon="heart" color="#00F0FF"
        size="6x" style={{marginRight: '3rem'}} fixedWidth
        border />
        <FontAwesomeIcon icon="code" color="#73DBFD"
        size="6x" fixedWidth border />
    </span>
</SectionTwo>
<SectionThree>
```

```
        <FlexBoxIndex>
            <div className="image">
                <img src="pic01.jpg" alt="pic01"/>
            </div>
            <div className="text__section3">
                <h2>Website Development</h2>
                <GenereicPara lessSize lessSpacing>We hand code
                beautiful websites using HTML5, CSS3, JS because
                they are fully customizable and efficient. No
                WordPress websites here.</GenereicPara>
            </div>
        </FlexBoxIndex>
        <FlexBoxIndex inverse>
            <div className="text__section3">
                <h2>Website Design</h2>
                <GenereicPara lessSize lessSpacing>We have
                talented and experienced Web Designers, who
                can design beautiful and usable websites.
                </GenereicPara>
            </div>
            <div className="image">
                <img src="pic02.jpg" alt="pic02"/>
            </div>
        </FlexBoxIndex>
        <FlexBoxIndex>
            <div className="image">
                <img src="pic03.jpg" alt="pic03"/>
            </div>
            <div className="text__section3">
                <h2>Mobile App Development</h2>
                <GenereicPara lessSize lessSpacing>We develop
                Mobile apps in Reactive Native, which can be
                used in both ios and Android.</GenereicPara>
            </div>
        </FlexBoxIndex>
    </SectionThree>
```

Let's do the same song and dance for the h2 tag. Remove h2 from SectionTwo and FlexBoxIndex. Then add a styled component called GenericH2 to the IndexStyles.js file, as shown in Listing 1-17.

Listing 1-17. Adding a Generic h2 Tag

```
const GenericH2 = styled.h2`
 font-size: 3rem;
 padding: ${props => (props.none ? "0" : "1.35em 0")};
 color: #ffffff;
 border-bottom: ${props => (props.none ? "0" : "2px solid #1d9c91")};
 text-transform: uppercase;
 letter-spacing: 0.6rem;
 margin: 0;
`
```

Now replace all the h2 tags with GenericH2. The updated code is highlighted in bold in Listing 1-18.

Listing 1-18. Adding GenericH2

```
                <SectionTwo>
                    <div>
                        <GenericH2>Our Passion</GenericH2>
                        <GenereicPara lessSize grey>Most good programmers
                        do programming not because they expect to get
                        paid,<br />
                            but because it's fun to program.</GenereicPara>
                        <h5>- Linus Torvalds</h5>
                    </div>
                  ...
                </SectionTwo>
                <SectionThree>
                    <FlexBoxIndex>
                        <div className="image">
                            <img src="pic01.jpg" alt="pic01"/>
                        </div>
```

```
            <div className="text__section3">
                <GenericH2 none>Website Development</GenericH2>
                <GenereicPara lessSize lessSpacing>We hand
                code beautiful websites using HTML5, CSS3,
                JS because they are fully customizable and
                efficient. No WordPress websites here.
                </GenereicPara>
            </div>
        </FlexBoxIndex>
        <FlexBoxIndex inverse>
            <div className="text__section3">
                <GenericH2 none>Website Design</GenericH2>
                <GenereicPara lessSize lessSpacing>We have
                talented and experienced Web Designers, who
                can design beautiful and usable websites.
                </GenereicPara>
            </div>
            <div className="image">
                <img src="pic02.jpg" alt="pic02"/>
            </div>
        </FlexBoxIndex>
        <FlexBoxIndex>
            <div className="image">
                <img src="pic03.jpg" alt="pic03"/>
            </div>
            <div className="text__section3">
                <GenericH2 none>Mobile App Development
                </GenericH2>
                <GenereicPara lessSize lessSpacing>We develop
                Mobile apps in Reactive Native, which can be
                used in both ios and Android.</GenereicPara>
            </div>
        </FlexBoxIndex>
    </SectionThree>
```

The result is shown in Figure 1-5.

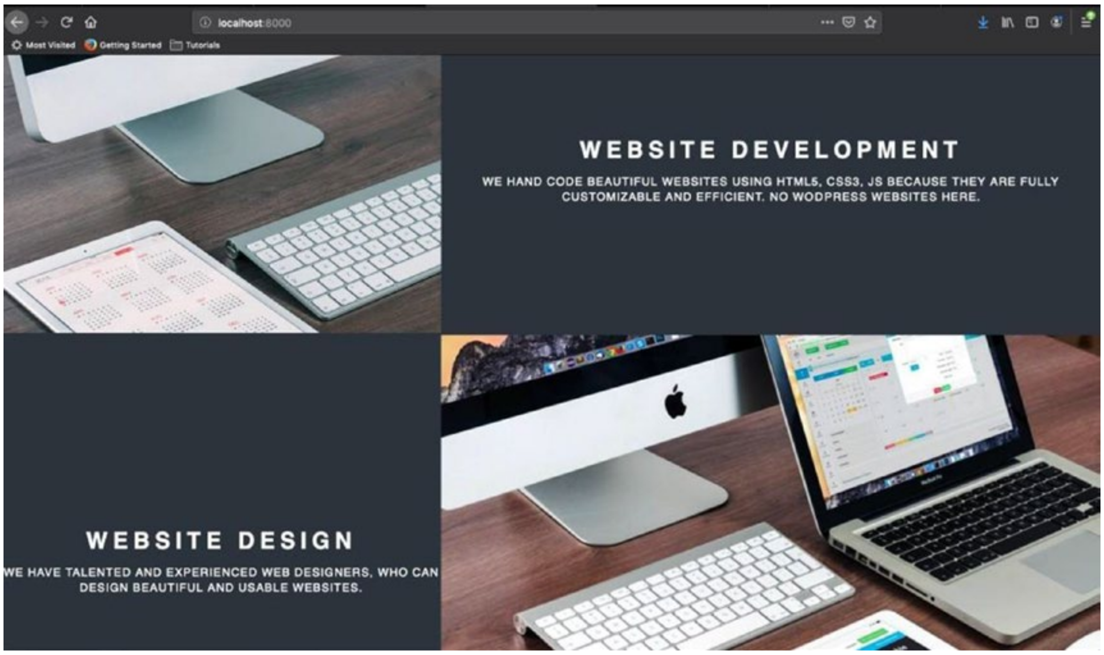

localhost:8000
WEBSITE DEVELOPMENT
WE HAND CODE BEAUTIFUL WEBSITES USING HTML5, CSS3, JS BECAUSE THEY ARE FULLY CUSTOMIZABLE AND EFFICIENT. NO WODPRESS WEBSITES HERE.
WEBSITE DESIGN
WE HAVE TALENTED AND EXPERIENCED WEB DESIGNERS, WHO CAN DESIGN BEAUTIFUL AND USABLE WEBSITES.

Figure 1-5. *Section three*

We have two more sections to go for the Home page; then we will create the Our Works and About Us pages. We will use a Layout component for the header and footer.

Creating Section Four

We will start with Section Four now. We have the usual heading and paragraph, followed by six sections containing some details about the technologies. We are adding `SectionFour` to the `index.js` file using the code shown in Listing 1-19.

Listing 1-19. Adding SectionFour to index.js

```
<SectionFour>
    <div className="header__section4">
        <div className="title__section4">Our Technologies</div>
        <GenereicPara lessSize grey>
        We use modern and latest technologies which helps our clients
        <br />
```

```
                as they are highly scalable and maintainable.
                </GenereicPara>
            </div>
        </SectionFour>
```

Now, let's add CSS for this code to the `IndexStyles.js` file, as shown in Listing 1-20. Also, don't forget to export the newly created `styled` component.

Listing 1-20. Adding CSS to IndexStyles.js

```
const SectionFour = styled.section`
 background-color: #505393;
 color: #ffffff;
 text-align: center;
 .header__section4 {
   width: 66%;
   margin: 0 auto;
 }
 .title__section4 {
   font-size: 3rem;
   padding: 1.35em 0;
   color: #ffffff;
   border-bottom: 2px solid #464981;
   text-transform: uppercase;
   letter-spacing: 0.225em;
   margin: 0;
 }
```

Now we will create the grid, which will contain the six sections.

Creating the Grid

The six sections will be exactly the same, only the background will be different. Put the code in Listing 1-21 under `header__section4`.

Listing 1-21. Changing the Background in index.js

```
<div className="grid__section4">
 <div className="item1" style={{ backgroundColor: "#4D508E" }}>
     <div className="flex__section4">
         <FontAwesomeIcon icon={faReact} color="#00FFCC"
         size="3x" fixedWidth />
         <GenericH2 none>React</GenericH2>
     </div>
     <GenereicPara lessSize lessSpacing grey>
         Modern JavaScript framework which will make your web
         application
         extremely fast and, at the same time, handy for every
         user.
     </GenereicPara>
     </div>
     <div className="item2" style={{ backgroundColor: "#4A4D89" }}>
     <div className="flex__section4">
         <FontAwesomeIcon icon="code" color="#00FFCC" size="3x"
         fixedWidth />
         <GenericH2 none>React Native</GenericH2>
     </div>
     <GenereicPara lessSize lessSpacing grey>
         Cross-platform for mobile app development based on
         JavaScript,
         whose resulting code is compiled to Android and iOS.
     </GenereicPara>
     </div>
     <div className="item3" style={{ backgroundColor: "#484A83" }}>
     <div className="flex__section4">
         <FontAwesomeIcon icon={faJs} color="#00FFCC" size="3x"
         fixedWidth />
         <GenericH2 none>JavaScript</GenericH2>
     </div>
     <GenereicPara lessSize lessSpacing grey>
```

```
    JavaScript is the language of the web. It is used for
    Web development, mobile development and app development
    and everything else.
</GenereicPara>
</div>
<div className="item4" style={{ backgroundColor: "#45477E" }}>
<div className="flex__section4">
    <FontAwesomeIcon icon={faHtml5} color="#00FFCC"
    size="3x" fixedWidth />
    <GenericH2 none>HTML5</GenericH2>
</div>
<GenereicPara lessSize lessSpacing grey>
    HTML, a standardized system for tagging text files to
    achieve font, colour, graphic, and hyperlink effects on
    World Wide Web pages.
</GenereicPara>
</div>
<div className="item5" style={{ backgroundColor: "#424479" }}>
<div className="flex__section4">
    <FontAwesomeIcon icon={faCss3} color="#00FFCC" size="3x"
    fixedWidth />
    <GenericH2 none>CSS3</GenericH2>
</div>
<GenereicPara lessSize lessSpacing grey>
    CSS is a style sheet language used for describing the
    presentation of a document written in a markup language
    like HTML.
</GenereicPara>
</div>
<div className="item6" style={{ backgroundColor: "#3F4174" }}>
<div className="flex__section4">
    <FontAwesomeIcon icon={faGalacticSenate} color="#00FFCC"
    size="3x" fixedWidth />
    <GenericH2 none>Gatsby</GenericH2>
</div>
```

```
                <GenereicPara lessSize lessSpacing grey>
                    Gatsby is a free and open source framework based on
                    React that helps developers build blazing fast websites
                    and apps
                </GenereicPara>
                </div>
            </div>
```

Now, we will add the CSS in Listing 1-22 to our `IndexStyles.js` file, inside the `SectionFour` styled component.

Listing 1-22. Adding CSS to IndexStyles.js

```
.grid__section4 {
   display: grid;
   grid-template-columns: 1fr 1fr;
   width: 66%;
   margin: 0 auto;
   padding: 3rem 0;
 }
 .grid__section4 > * {
   padding: 3rem;
 }
 .flex__section4 {
   display: flex;
   justify-content: center;
   align-items: center;
   padding: 1rem;
 }
 .flex__section4 > h2 {
   margin-left: 1rem;
 }
```

We also need to install a new `fontawesome` library, as we are using some brand icons here. So, in the terminal, stop Gatsby and install this via `npm`:

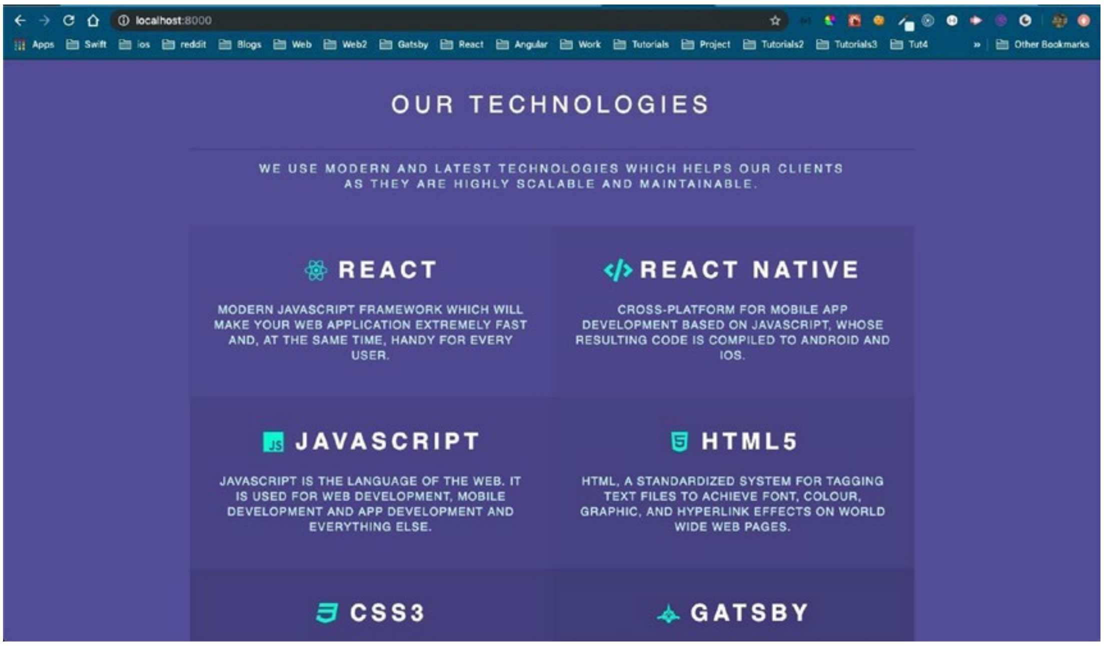
```
npm install @fortawesome/free-brands-svg-icons
```

The changes made to the header of the index.js file are marked in bold in Listing 1-23.

Listing 1-23. Changing the Header in index.js

```
import React from "react"
import { Banner, TextWrapper, MoreText, SectionTwo, SectionThree,
SectionFour, FlexBoxIndex, GenereicPara, GenericH2 } from "../styles/
IndexStyles"
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome"
import { library } from "@fortawesome/fontawesome-svg-core"
import { fab, faHtml5, faJs, faReact, faCss3, faGalacticSenate } from
"@fortawesome/free-brands-svg-icons"
import { faHeart, faCode, faGem, fab, fas } from "@fortawesome/free-solid-
svg-icons"
library.add(faHeart, faCode, faGem, fab, fas)
```

Now, our Section Four looks like Figure 1-6.



OUR TECHNOLOGIES

WE USE MODERN AND LATEST TECHNOLOGIES WHICH HELPS OUR CLIENTS AS THEY ARE HIGHLY SCALABLE AND MAINTAINABLE.

REACT

MODERN JAVASCRIPT FRAMEWORK WHICH WILL MAKE YOUR WEB APPLICATION EXTREMELY FAST AND, AT THE SAME TIME, HANDY FOR EVERY USER.

REACT NATIVE

CROSS-PLATFORM FOR MOBILE APP DEVELOPMENT BASED ON JAVASCRIPT, WHOSE RESULTING CODE IS COMPILED TO ANDROID AND IOS.

JAVASCRIPT

JAVASCRIPT IS THE LANGUAGE OF THE WEB. IT IS USED FOR WEB DEVELOPMENT, MOBILE DEVELOPMENT AND APP DEVELOPMENT AND EVERYTHING ELSE.

HTML5

HTML, A STANDARDIZED SYSTEM FOR TAGGING TEXT FILES TO ACHIEVE FONT, COLOUR, GRAPHIC, AND HYPERLINK EFFECTS ON WORLD WIDE WEB PAGES.

CSS3

GATSBY

Figure 1-6. *Section Four page*

Creating the Contact Us Form

The next section will have a parallax image with a Contact Us form. We will be reusing the `banner` styled component used on the Index page here. Add the code in Listing 1-24 after `SectionFour`.

Listing 1-24. Parallax Image in index.js

```
<section style={{ position: "relative" }}>
    <Banner parallax></Banner>
</section>
```

We need to update the `banner` styled component to include the parallax effect. Update the bold text shown in Listing 1-25.

Listing 1-25. IndexStyles.js

```
const Banner = styled.div`
 &:after {
   content: "";
   display: block;
   height: ${props => (props.parallax ? "80vh" : "100vh")};
   width: 100%;
   background-image: url('banner.jpg');
   background-size: cover;
   background-repeat: no-repeat;
   background-position: center;
   background-attachment: ${props => (props.parallax ? "fixed" : "scroll")};
   filter: grayscale(100%) blur(2px);
 }
`
```

Next, let's add a form to the top of the image. The code is shown in Listing 1-26, which we need to put after `banner`.

Listing 1-26. Form in index.js

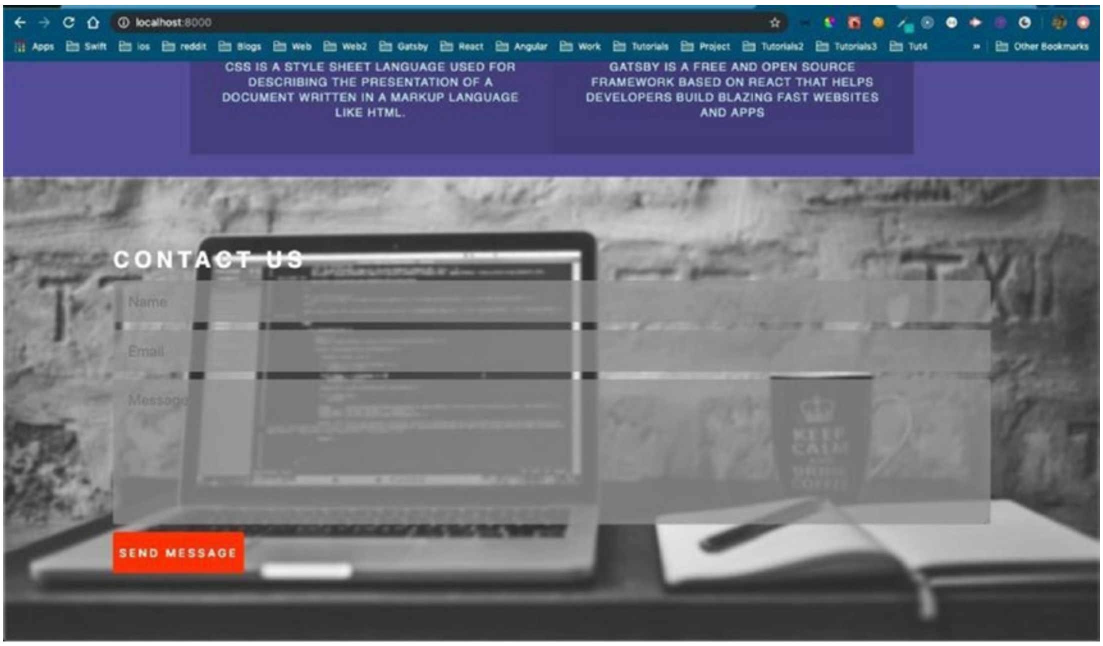
```
<FormFive>
 <form name="contact" method="post" data-netlify="true">
 <div className="fields">
     <GenericH2 none>Contact Us</GenericH2>
     <input type="text" name="name" id="name" placeholder="Name" />
     <input type="email" name="email" id="email" placeholder="Email" />
     <textarea name="message" id="message" placeholder="Message" rows="7">
     </textarea>
     <div className="actions">
         <input type="submit" value="Send Message" className="button__
         primary" />
     </div>
 </div>
 </form>
</FormFive>
```

Next, let's add the styles for `FormFive` to `IndexStyles.js`, as shown in Listing 1-27.

Listing 1-27. Styles for Form Five in IndexStyles.js

```
const FormFive = styled.div`
 position: absolute;
 z-index: 1;
 width: 80%;
 top: 50%;
 left: 50%;
 transform: translate(-50%, -50%);
 color: white;
 form > .fields {
   display: grid;
   grid-template-columns: 1fr;
   grid-gap: 1rem;
 }
 input[type="text"],
 input[type="email"],
textarea {
  appearance: none;
  font-size: 2rem;
  background-color: rgba(144, 144, 144, 0.75);
  border-radius: 3px;
  border: none;
}
input[type="text"],
input[type="email"] {
  height: 2.75em;
  padding: 0.75em 1em;
}
textarea {
  padding: 0.75em 1em;
}
.button__primary {
  background-color: #ed4933;
  box-shadow: none;
  color: #ffffff;
  border-radius: 3px;
  border: 0;
  cursor: pointer;
  font-size: 1.5rem;
  font-weight: 600;
  letter-spacing: 0.225em;
  padding: 1.8rem 0.8rem;
  text-align: center;
  text-decoration: none;
  text-transform: uppercase;
}
`
```

This will result in the section shown in Figure 1-7.


CSS IS A STYLE SHEET LANGUAGE USED FOR DESCRIBING THE PRESENTATION OF A DOCUMENT WRITTEN IN A MARKUP LANGUAGE LIKE HTML.

GATSBY IS A FREE AND OPEN SOURCE FRAMEWORK BASED ON REACT THAT HELPS DEVELOPERS BUILD BLAZING FAST WEBSITES AND APPS

Figure 1-7. *The Contact Us page*

Creating the Footer Section

Now, we will create the footer section. This footer will be common to all our pages, so we will put it in the Layout component.

Create a folder called `components` inside `src` and a file called `layout.js` inside that. The file contains a simple footer with different icons. Here, the `{children}` will show the components, which we will soon use to wrap the Layout component. The code is shown Listing 1-28.

Listing 1-28. Footer Section of layout.js

```
import React from "react"
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome'
import { faTwitter, faFacebook, faInstagram, faLinkedin, faYoutube,
faPinterest } from '@fortawesome/free-brands-svg-icons'
import { Footer } from '../styles/IndexStyles';
export default ({ children }) => (
   <div>
```

```
       {children}
       <Footer>
           <div class="icons">
               <FontAwesomeIcon icon={faTwitter} size="2x" fixedWidth />
               <FontAwesomeIcon icon={faFacebook} size="2x" fixedWidth />
               <FontAwesomeIcon icon={faInstagram} size="2x" fixedWidth />
               <FontAwesomeIcon icon={faLinkedin} size="2x" fixedWidth />
               <FontAwesomeIcon icon={faYoutube} size="2x" fixedWidth />
               <FontAwesomeIcon icon={faPinterest} size="2x" fixedWidth />
           </div>
           <div class="copyright">
               <span>&copy; 2019, GeekyHacker | </span>
               <span>Made with ❤ and ☕ in India</span>
           </div>
       </Footer>
   </div>
)
```

Let's put some styles in `IndexStyles.js` for the footer. These styles can be placed anywhere, as the order doesn't matter. But don't forget to export it at the end. The code is shown in Listing 1-29.

Listing 1-29. Styles for the Footer in IndexStyles.js

```
const Footer = styled.footer`
 padding: 6em 0 4em 0;
 background-color: #1d242a;
 text-align: center;
 .icons > * {
   cursor: pointer;
   margin-right: 1rem;
   color: rgba(255, 255, 255, 0.5);
 }
 .copyright {
   color: rgba(255, 255, 255, 0.5);
   font-size: 1.2rem;
   letter-spacing: 0.225em;
   padding: 0;
   text-transform: uppercase;
   margin-top: 1rem;
 }
`
```

Now, let's wrap our home page with this Layout component. The updated code is marked in bold in Listing 1-30.

Listing 1-30. The Updated index.js File

```
import React, { Component } from "react"
import { Link } from "gatsby"
import { Banner, TextWrapper, MoreText, SectionTwo, SectionThree,
SectionFour, FlexBoxIndex, GenereicPara, GenericH2 } from "../styles/
IndexStyles"
import Layout from "../components/layout"
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome"
import { library } from "@fortawesome/fontawesome-svg-core"
import { fab, faHtml5, faJs, faReact, faCss3, faGalacticSenate } from
"@fortawesome/free-brands-svg-icons"
import { faHeart, faCode, faGem, fas } from "@fortawesome/free-solid-svg-icons"

library.add(faHeart, faCode, faGem, fab, fas)

export default () => (
<Layout>
   <section style={{ position: "relative" }}>...
   <SectionTwo>...
   <SectionThree>...
   <SectionFour>...
   <section style={{ position: "relative" }}>...
</Layout>
)
```

Our home page now has a footer, as shown in Figure 1-8.

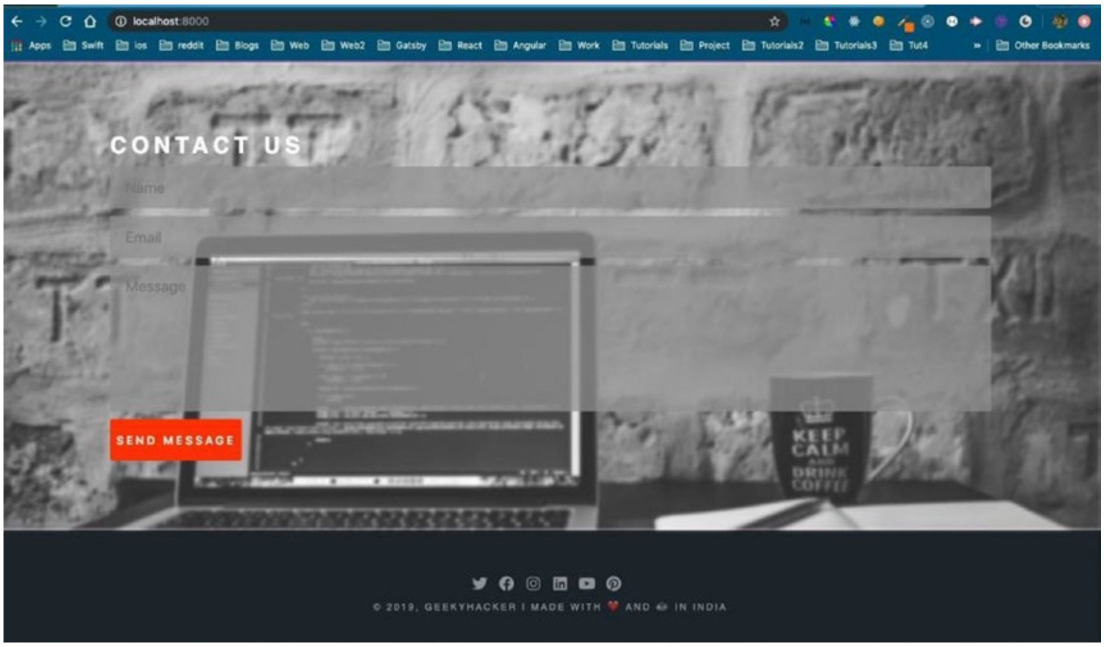


Figure 1-8. *The footer*

We have one more thing left to do to the home page, and that is to add a menu.

Creating the Menu

The menu is also created in the Layout component, as it will be shared with the Our Works and About Us pages. Let's first create a header with navigation by adding the bold text in Listing 1-31 to the `layout.js` file.

Listing 1-31. Header in layout.js

```
import React from "react"
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome'
import { faTwitter, faFacebook, faInstagram, faLinkedin, faYoutube,
faPinterest } from '@fortawesome/free-brands-svg-icons'
import { Header, Footer, GenericH2, GenereicPara } from '../styles/
IndexStyles';
import { Link } from "gatsby";
```

```
export default ({ children }) => (
  <div>
      <Header>
          <GenericH2 none>
              <Link to="/" style={{color: '#fff'}}>GeekyHacker</Link>
          </GenericH2>
          <div className="menu__items">
              <Link to="/works"><GenereicPara lessSize lessSpacing grey>
              Works</GenereicPara></Link>
              <Link to="/about"><GenereicPara lessSize lessSpacing grey>
              About</GenereicPara></Link>
          </div>
      </Header>
      {children}
      <Footer>
          ...
           ...

      </Footer>
  </div>
)
```

Let's add some styles to `styled-components`, as shown in Listing 1-32.

Listing 1-32. Header Styles in IndexStyles.js

```
const Header = styled.header`
 display: flex;
 justify-content: space-between;
 align-items: center;
 width: 100%;
 height: 30px;
 background-color: #1d242a;
 color: #ffffff;
 padding: 3rem 0;
 .menu__items {
   display: flex;
 }
```

```
.menu__items > * {
  margin-right: 1rem;
}
`
```

We now get a header with a menu, as shown in Figure 1-9.

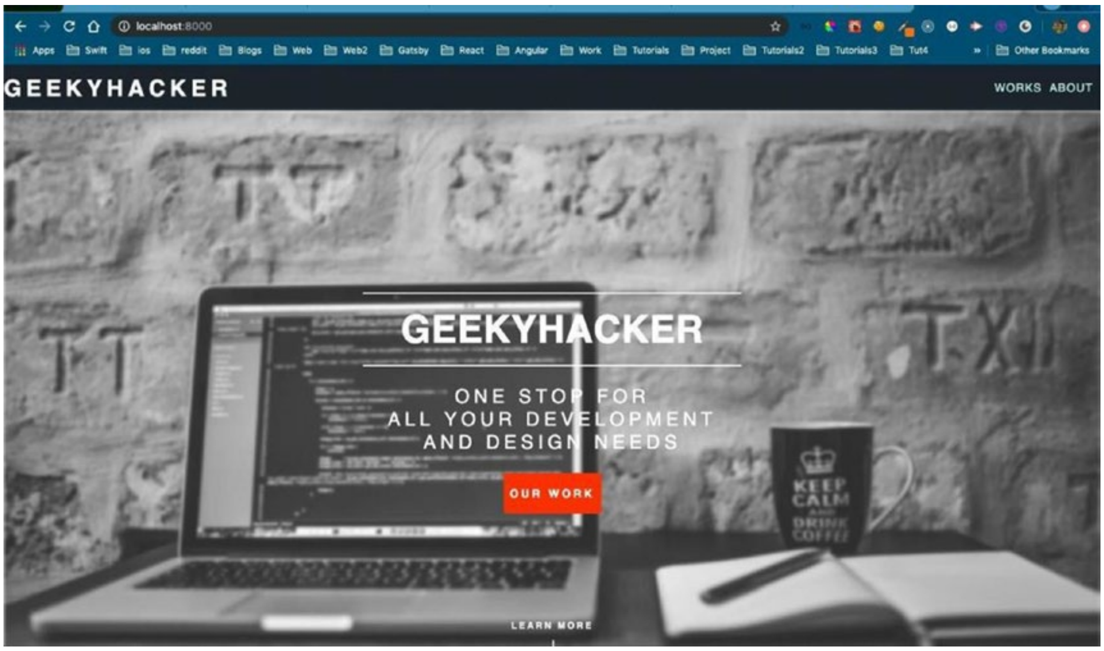


Figure 1-9. *The menu*

Creating the Our Works and About Us Pages

Next, we will create the Our Works page. Create a new file called `works.js` inside the pages folder. The content is shown in Listing 1-33.

Listing 1-33. The works.js File

```
import React from "react";
import { Works, GenericH2 } from "../styles/IndexStyles";
import Layout from "../components/layout";
import Project from "../components/Project";

const projects = []
```

```
export default () => (
   <Layout>
       <Works>
           <GenericH2 none dark style={{textAlign: 'center'}}>Our Works
           </GenericH2>
           <section class="gallery__flex">
             { projects && projects.map(proj => <Project key={proj.title}
             project={proj} />)}
           </section>
       </Works>
   </Layout>
)
```

Let's add some styles to the IndexStyles.js file, as shown in Listing 1-34.

Listing 1-34. Works Styles in IndexStyles.js

```
const Works = styled.div`
 background-color: #ffffff;
 color: #4e4852;
 padding: 2em 0 1em 0 !important;
 .gallery__flex {
   display: grid;
   grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
   grid-gap: 20px;
   justify-items: center;
 }
`
```

We will now create a new component called `project` to which we will pass each project. Let's create an array called `objects` that contains some projects, which we pass to the `project` component by mapping over it. The content is shown in Listing 1-35.

Listing 1-35. Array in works.js

```
const projects = [
   {image: "printbill.png", title: "PrintBill", link: "https://www.
   printbill.in/"},
    {image: "sprung.png", title: "Sprung", link: "https://sprung.us"},
    {image: "orange.png", title: "Orange Health", link: "https://whispering-
    bastion-31600.herokuapp.com/"},
    {image: "billing.png", title: "Billing Restro", link: "https://
    billingrestro-react-prod.herokuapp.com/"},
    {image: "ferrarisports.png", title: "Ferrari Sports", link: "http://
    ferrarisports.com/"},
    {image: "pregnancy.png", title: "Pregnancy info", link: "http://
    pregnancy.info/"},
    {image: "jaagastudy.png", title: "Jaaga Study", link: "https://
    nabendu82.github.io/incognosco/index.html"},
    {image: "responsive1.png", title: "Responsive Site- POC", link:
    "https://shikhacorps.in/corps/"},
    {image: "responsive2.png", title: "Responsive Site2- POC", link:
    "https://shikhacorps.in/cssgridresponsive/"},
    {image: "styleconferences.png", title: "Style Conferences", link:
    "https://nabendu82.github.io/shayhowe/index.html"},
    {image: "itunes.png", title: "iTunes Clone - POC", link: "https://
    shikhacorps.in/mytunes/"},
    {image: "parallax.png", title: "Parallax Site - POC", link: "https://
    shikhacorps.in/parallaxsite/"},
    {image: "photography.png", title: "PhotoGraphy Site-POC", link:
    "https://shikhacorps.in/photographysite/"},
    {image: "yelpcamp.png", title: "YelpCamp", link: "https://hidden-
    coast-48928.herokuapp.com/"},
    {image: "blogsite.png", title: "Blog Site", link: "https://serene-
    wildwood-22136.herokuapp.com/blogs"},
    {image: "portfolio.png", title: "Portfolio Site", link: "https://
    nabendu82.github.io/"},
]
```

Next, let's create the `project` component inside the `components` folder. First create a new `Project.js` file inside the `components` folder. The content is shown in Listing 1-36.

Listing 1-36. The Project.js File

```
import React from 'react';

const Project = ({ project }) => {
   return (
       <div class="card">
           <img class="card__img" src={project.image} alt="" />
           <div style={{marginTop: '1rem'}}>
               <h5 class="card-title">{project.title}</h5>
               <a class="card-link" href={project.link} target="_blank">Link</a>
           </div>
       </div>
       )
}

export default Project;
```

Let's add some styles for this component in `global.css`. Enter the code in Listing 1-37 after the existing styles.

Listing 1-37. Styles in global.css

```
.card {
   background: lightgray;
   padding: 10px;
   width: 300px;
   height: 200px;
   margin: 10px 10px;
   color: white;
   text-align: center;
 }
 .card-link{
   color: blue;
   border-bottom: dotted 1px;
   font-size: 1.5rem;
   line-height: 2rem;
   font-weight: 600;
 }
```

```
.card-title{
  color: black;
  margin:0;
  padding:0;
  font-size: 1.6rem;
  line-height: 2.2rem;
  font-weight: 800;
  letter-spacing: 0.225em;
  text-transform: uppercase;
}
.card__img {
  width: 100%;
  height: 120px;
  object-fit: cover;
}
```

The Our Works page will look as shown in Figure 1-10.

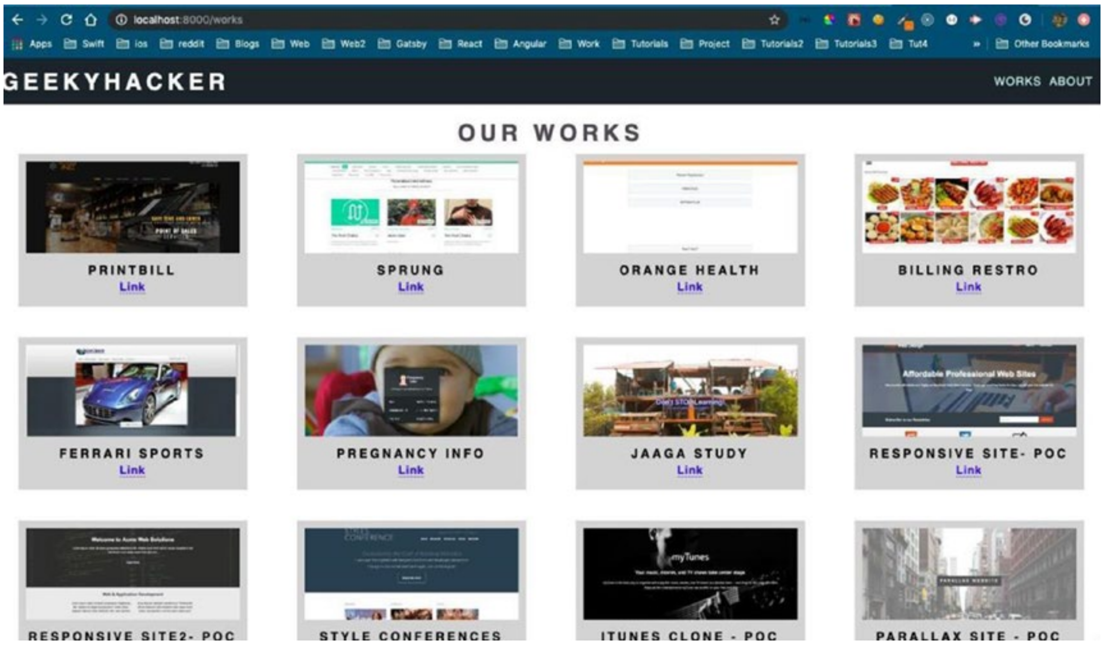


Figure 1-10. *Our Works page*

We missed one thing and that is to make the header *fixed*, which means that if we scroll down the page, the header still remains at the top. To do this, add the code marked in bold in Listing 1-38 to the `IndexStyles.js` file.

Listing 1-38. Fixed Header in IndexStyles.js

```
const Header = styled.header`
 display: flex;
 justify-content: space-between;
 align-items: center;
 width: 100%;
 height: 30px;
 background-color: #1d242a;
 color: #ffffff;
 padding: 3rem 0;
 z-index: 1000;
 position: fixed;
 top:0;
 left:0;
 .menu__items {
   display: flex;
 }
 .menu__items > * {
   margin-right: 1rem;
 }
`
```

The code causes issues with the Our Works text, because of the `margin: 0` set for `GenericH2`. Let's edit it using `props`, so that it can be used in `works.js` without affecting other files. The updated code is marked in bold in Listing 1-39.

Listing 1-39. Header Styles in IndexStyles.js

```
const GenericH2 = styled.h2`
 font-size: 3rem;
 padding: ${props => (props.none ? "0" : "1.35em 0")};
 color: ${props => (props.dark ? "#4E4852" : "#ffffff")};
 border-bottom: ${props => (props.none ? "0" : "2px solid #1d9c91")};
 text-transform: uppercase;
 letter-spacing: 0.6rem;
 margin: ${props => (props.some ? "5rem 0 0 0" : "0")};
`
```

Now, in `works.js`, we just need to add some props, as shown in Listing 1-40.

Listing 1-40. Adding Some Props to works.js

```
export default () => (
   <Layout>
       <Works>
           <GenericH2 none dark some style={{textAlign: 'center'}}>Our
           Works</GenericH2>
           <section class="gallery__flex">
            { projects && projects.map(proj => <Project key={proj.title}
            project={proj} />)}
           </section>
       </Works>
   </Layout>
)
```

We will next create a new page called `about.js` inside the pages folder. The content is shown in Listing 1-41.

Listing 1-41. The about.js File

```
import React from 'react';
import Layout from "../components/layout";
import { Link } from "gatsby";
import { Banner, TextWrapper, GenereicPara, GenericH3 } from "../styles/
IndexStyles";

const about = () => {
   return (
       <Layout>
       <section style={{ position: 'relative' }}>
           <Banner different></Banner>
           <TextWrapper>
```

```
                <div>
                    <GenericH3 none>Who are we</GenericH3>
                    <GenereicPara lessSize grey>GeekyHacker is an experienced and
                    passionate group of designers, developers, and artists. Every
                    client we work with becomes part of the team. Together we
                    face the challenges and celebrate the victories.
                    </GenereicPara>
                    <Link to="/works">Our Work</Link>
                </div>
            </TextWrapper>
        </section>
        </Layout>
    )
}

export default about
```

Here, we are reusing most of the `index.js` file, but using a different banner image. We are doing this again with the help of props of styled components. The changes are marked in bold in Listing 1-42.

Listing 1-42. Banner Image in IndexStyles.js

```
const Banner = styled.div`
 &:after {
   content: "";
   display: block;
   height: ${props => (props.parallax ? "80vh" : "100vh")};
   width: 100%;
   background-image: ${props => (props.different ? "url('developer.jpg')" :
   "url('banner.jpg')")};
   background-size: cover;
   background-repeat: no-repeat;
   background-position: center;
   background-attachment: ${props => (props.parallax ? "fixed" : "scroll")};
   filter: grayscale(100%) blur(2px);
 }
`
```

Let's create SectionTwo now, which we are going to place after </section>. The code is shown in Listing 1-43.

Listing 1-43. Section Two in about.js

```
<SectionTwo white>
<div>
   <GenericH3 dark none>About Us</GenericH3>
   <GenereicParaAbout lessSize grey>Founded in 2016, <b>GeekyHacker</b> is
   a small web design & development company based in Bangalore, India.
   Over the last few years we've made a reputation for building websites,
   mobile apps, and web apps that look great and are easy-to-use.</
   GenereicParaAbout>
   <GenereicParaAbout lessSize grey>We originated from futuristic
   technology and progressing toward success with a great desire. We work
   with self-derived strategies, as we have experienced everything on our
   own. We are equipped with the state-of-the-art work station in the
   website development and testing. So, results delivered on time, every
   time! Your success is our bread and butter!</GenereicParaAbout>
   <GenereicParaAbout lessSize grey>We always happy to say loudly, we smell
   & feel the success every day because of the ShikhaCorps experts in
   execution planning with website design and digital marketing. They
   are always adopting with our valuable clients to satisfy on their
   requirement in each perspective, so our valuable clients make us more
   valuable in our success. While you work with us you feel, we are in
   right place & right time.</GenereicParaAbout>
   <GenereicParaAbout lessSize grey>Our Vision is to make our self
   as India's most valuable corporation through ultimate performance
   and uniqueness in every single project that we do!. Our Mission is
   to enhance the wealth generating capability of the enterprise in a
   globalizing environment by exhibiting our efficiency and adopting the
   innovative "more-than-enough" methodology in our work.
   </GenereicParaAbout>
</div>
</SectionTwo>
```

We have made changes to the `SectionTwo` styled component, to include a white background. These changes are shown in bold in Listing 1-44.

Listing 1-44. White Background in IndexStyles.js

```
const SectionTwo = styled.section`
 background-color: ${props => (props.white ? "#ffffff" : "#21b2a6")};
 text-align: center;
 padding: 10rem 0;
...
...
`
```

We also added two new styled components, as shown in Listing 1-45.

Listing 1-45. New Styled Components in IndexStyles.js

```
const GenericH3 = styled.h3`
 font-size: 3rem;
 padding: ${props => (props.none ? "0" : "1.35em 0")};
 color: ${props => (props.dark ? "#4E4852" : "#ffffff")};
 border-bottom: ${props => (props.none ? "0" : "2px solid #1d9c91")};
 text-transform: uppercase;
 letter-spacing: 0.6rem;
 margin: 0;
`

const GenereicParaAbout = styled.p`
 text-transform: uppercase;
 text-align: center;
 letter-spacing: ${props => (props.lessSpacing ? "0.075em" : "0.225em")};
 font-size: ${props => (props.lessSize ? "1.5rem" : "2.5rem")};
 line-height: ${props => (props.lessSize ? "2rem" : "3rem")};
 color: ${props => (props.grey ? "#4E4852" : "#ffffff")};
`
```

Now the About Us page looks like Figure 1-11.

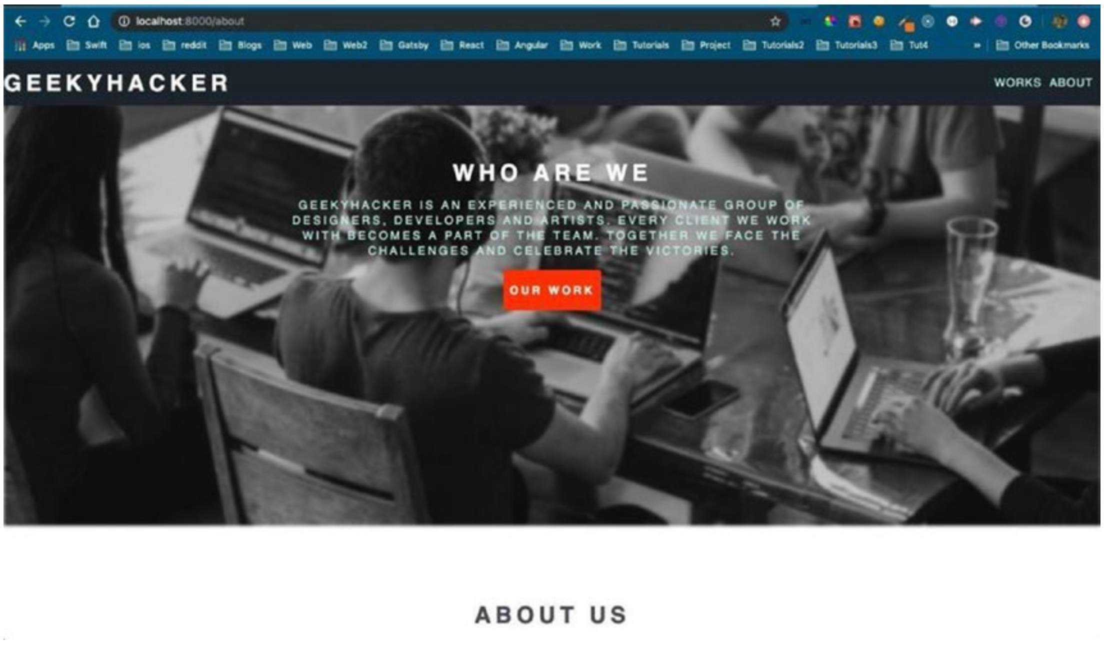

localhost:8000/about
GEEKYHACKER
WORKS ABOUT
WHO ARE WE
GEEKYHACKER IS AN EXPERIENCED AND PASSIONATE GROUP OF DESIGNERS, DEVELOPERS AND ARTISTS. EVERY CLIENT WE WORK WITH BECOMES A PART OF THE TEAM. TOGETHER WE FACE THE CHALLENGES AND CELEBRATE THE VICTORIES.
OUR WORK
ABOUT US

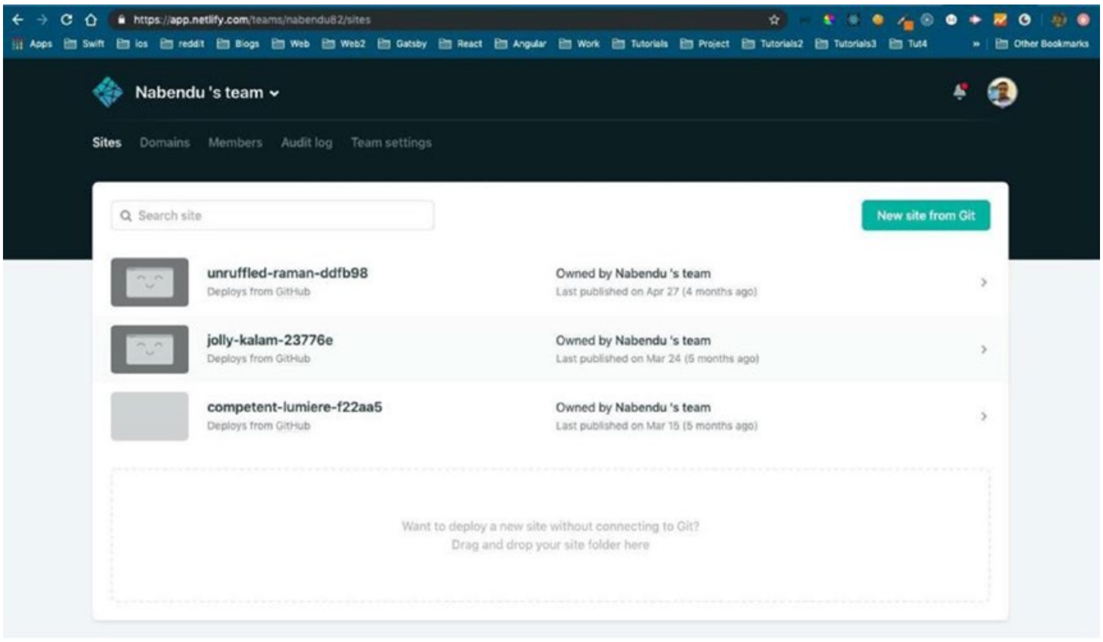
Figure 1-11. *The About Us page*

It's time to deploy the site to Netlify.

Deploying the Site

We already set up a form submission in the `index.js` page with Netlify. If you want to see the form submission setup details, watch this[2] YouTube video by Traversy Media. I have pushed all my code to GitHub, so open your Netlify account. If you are using Netlify for the first time, you need to register through your GitHub account (see Figure 1-12).

[2] https://www.youtube.com/embed/6ElQ689HRcY



unruffled-raman-ddfb98
Deploys from GitHub
Owned by Nabendu 's team
Last published on Apr 27 (4 months ago)

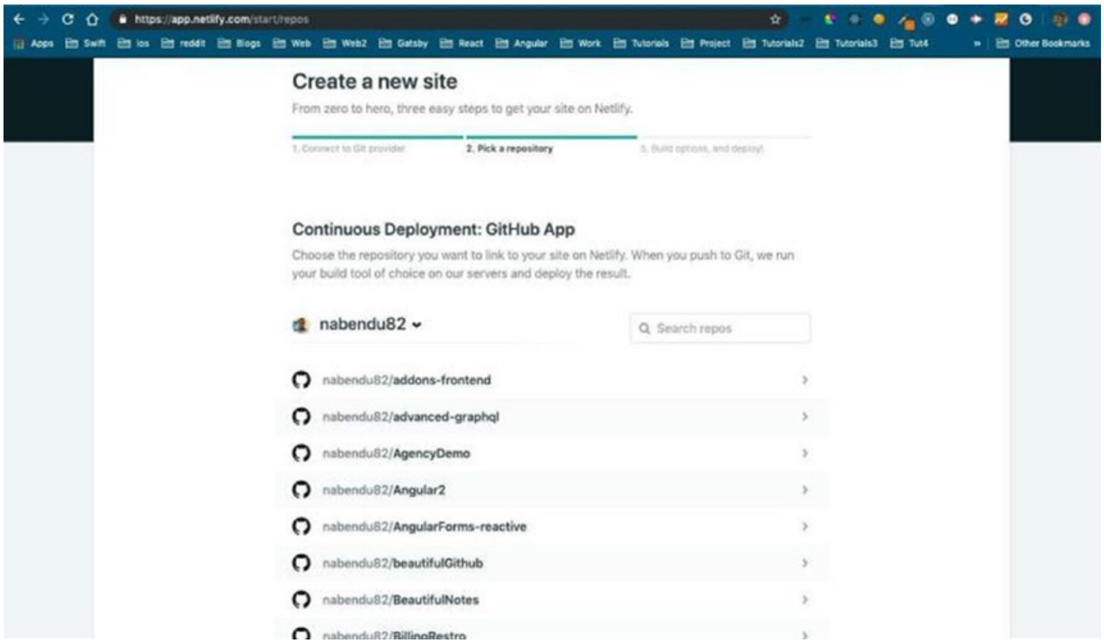
jolly-kalam-23776e
Deploys from GitHub
Owned by Nabendu 's team
Last published on Mar 24 (5 months ago)

competent-lumiere-f22aa5
Deploys from GitHub
Owned by Nabendu 's team
Last published on Mar 15 (5 months ago)

Want to deploy a new site without connecting to Git?
Drag and drop your site folder here

Figure 1-12. *Netlify*

Next, click on New Site from Git and then choose the provider. I chose GitHub, since my code is there. Once you authorize it, it will show you a list of all your GitHub repositories (see Figure 1-13).



https://app.netlify.com/start/repos

Create a new site

From zero to hero, three easy steps to get your site on Netlify.

1. Connect to Git provider
2. Pick a repository
3. Build options, and deploy!

Continuous Deployment: GitHub App

Choose the repository you want to link to your site on Netlify. When you push to Git, we run your build tool of choice on our servers and deploy the result.

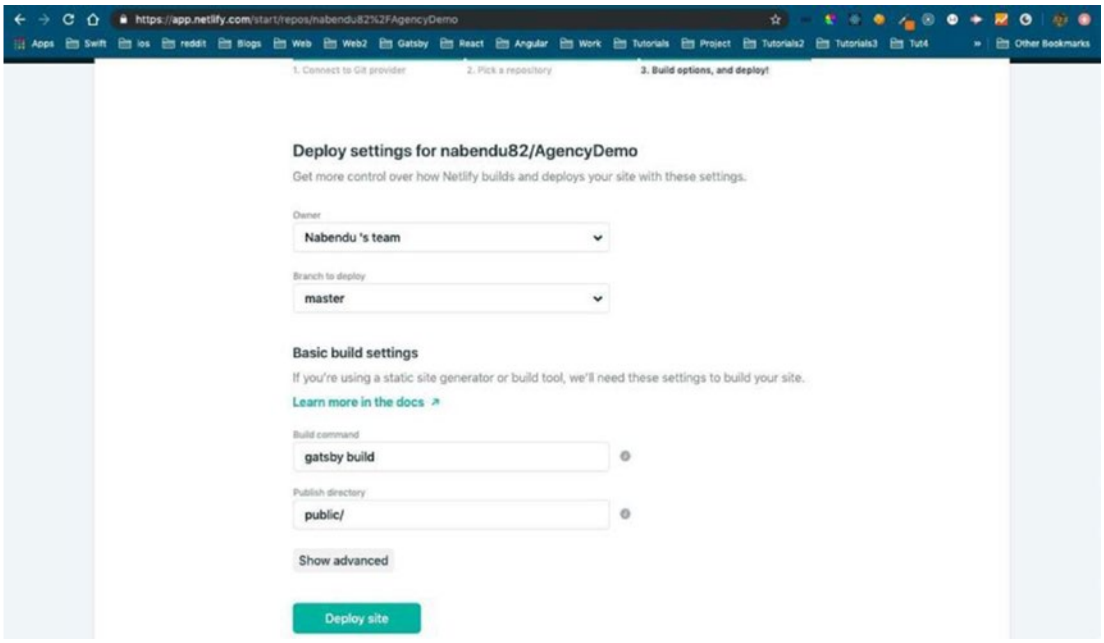
nabendu82

Search repos

nabendu82/addons-frontend
nabendu82/advanced-graphql
nabendu82/AgencyDemo
nabendu82/Angular2
nabendu82/AngularForms-reactive
nabendu82/beautifulGithub
nabendu82/BeautifulNotes
nabendu82/BillingRestro

Figure 1-13. *All the repos are listed*

Click on the repo to deploy it. In my case, the repo is called `AgencyDemo` (see Figure 1-14).



https://app.netlify.com/start/repos/nabendu82%2FAgencyDemo

1. Connect to Git provider
2. Pick a repository
3. Build options, and deploy!

Deploy settings for nabendu82/AgencyDemo

Get more control over how Netlify builds and deploys your site with these settings.

Owner: Nabendu 's team

Branch to deploy: master

Basic build settings

If you're using a static site generator or build tool, we'll need these settings to build your site.

Learn more in the docs

Build command: gatsby build

Publish directory: public/

Show advanced

Deploy site

Figure 1-14. *Defaults*

Keep the defaults and click on Deploy Site. The site will be deployed within minutes (see Figure 1-15).

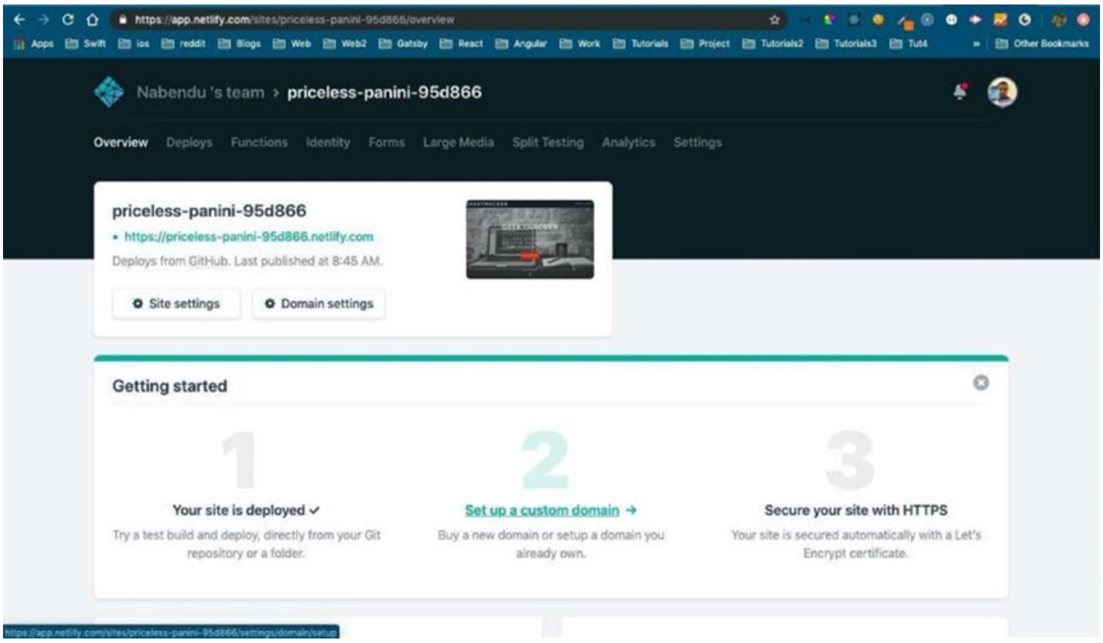

https://app.netlify.com/sites/priceless-panini-95d866/overview
Nabendu 's team › priceless-panini-95d866
Overview
Deploys
Functions
Identity
Forms
Large Media
Split Testing
Analytics
Settings
priceless-panini-95d866
https://priceless-panini-95d866.netlify.com
Deploys from GitHub. Last published at 8:45 AM.
Site settings
Domain settings
Getting started
1
Your site is deployed
Try a test build and deploy, directly from your Git repository or a folder.
2
Set up a custom domain
Buy a new domain or setup a domain you already own.
3
Secure your site with HTTPS
Your site is secured automatically with a Let's Encrypt certificate.

Figure 1-15. *The site is quickly deployed*

Our site has been deployed. It's time to check the form submission feature of Netlify. Go to the form on the home page and add content (see Figure 1-16).

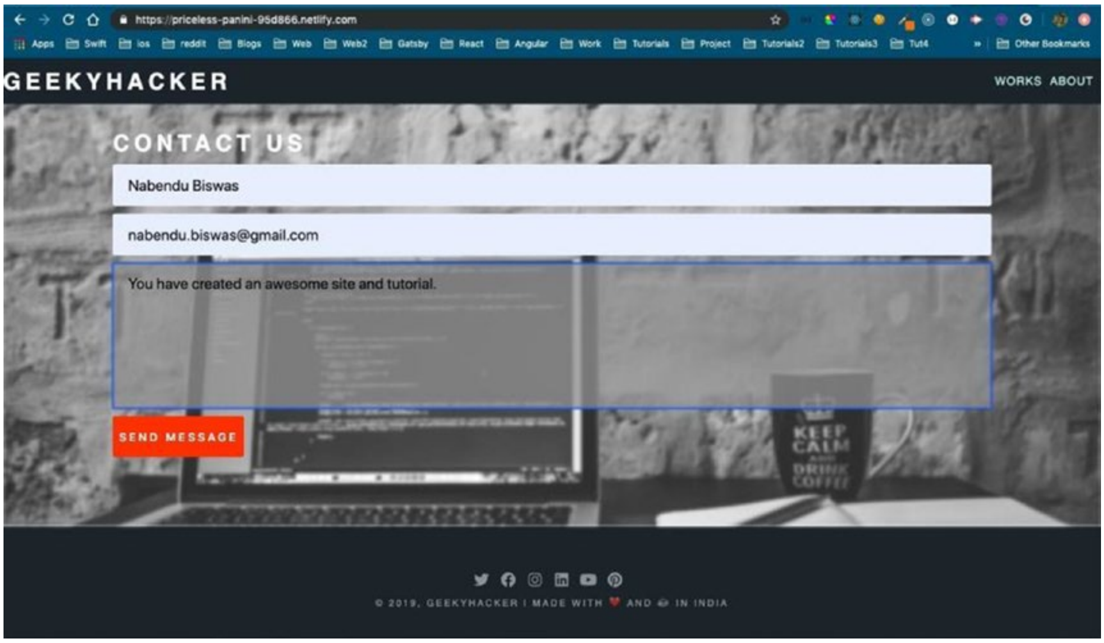

https://priceless-panini-95d866.netlify.com
GEEKYHACKER
WORKS ABOUT
CONTACT US
Nabendu Biswas
nabendu.biswas@gmail.com
You have created an awesome site and tutorial.
SEND MESSAGE
© 2019, GEEKYHACKER | MADE WITH ♥ AND ☕ IN INDIA

Figure 1-16. *The content has been added*

The form submission initially didn't go as expected and gave us a lot of errors. After searching, I found that the form handling in sites generated through Gatsby is a bit different in Netlify. I found these two links to solve the issue:

`https://www.netlify.com/blog/2017/07/20/how-to-integrate-netlifys-form-handling-in-a-react-app/#form-handling-with-static-site-generators`[3]

`https://github.com/sw-yx/gatsby-netlify-form-example-v2`[4]

So, we will change our functional component to a class-based component and make the changes shown in bold in Listing 1-46.

Listing 1-46. Functional to Class-Based Component: The index.js File

```
import React, { Component } from "react";
import { Link } from "gatsby";
import { Banner, TextWrapper, MoreText, SectionTwo, SectionThree, SectionFour,
FormFive, FlexBoxIndex, GenereicPara, GenericH2 } from "../styles/IndexStyles";
```

[3]https://www.netlify.com/blog/2017/07/20/how-to-integrate-netlifys-form-handling-in-a-react-app/#form-handling-with-static-site-generators

[4]https://github.com/sw-yx/gatsby-netlify-form-example-v2

```
import Layout from "../components/layout"
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { library } from '@fortawesome/fontawesome-svg-core';
import { fab, faHtml5, faJs, faReact, faCss3, faGalacticSenate } from
'@fortawesome/free-brands-svg-icons'
import { faHeart, faCode, faGem, fas } from '@fortawesome/free-solid-svg-icons';
import { navigate } from 'gatsby-link';

library.add(faHeart, faCode, faGem, fab, fas);

const encode = (data) => {
   return Object.keys(data)
       .map(key => encodeURIComponent(key) + "=" + encodeURIComponent(data[
       key]))
       .join("&");
}

class IndexPage extends Component {
   constructor(props) {
       super(props)
       this.state = { name: "", email: "", message: "" };
   }

   handleSubmit = e => {
       e.preventDefault();
       const form = e.target
       fetch("/", {
         method: "POST",
         headers: { "Content-Type": "application/x-www-form-urlencoded" },
         body: encode({ 'form-name': form.getAttribute('name'),
         ...this.state })
       })
         .then(() => navigate(form.getAttribute('action')))
         .catch(error => alert(error));
     };

     handleChange = e => this.setState({ [e.target.name]: e.target.value });
```

We also need to make these changes to the `index.js` file, marked in bold in Listing 1-47.

Listing 1-47. More Changes to index.js

```
render() {
 const { name, email, message } = this.state;
   return (
     <Layout>
         <section style={{ position: 'relative' }}>...
         <SectionTwo>...
         <SectionThree>...
         <SectionFour>...
         <section style={{ position: 'relative' }}>
             <Banner parallax></Banner>
             <FormFive>
                 <form name="contact" method="post" action="/thanks/" data-
                 netlify="true"                onSubmit={this.handleSubmit}
                 >
                     <div className="fields">
                         <GenericH2 none>Contact Us</GenericH2>
                         <input type="text" name="name" id="name"
                         placeholder="Name" value={name} onChange={this.
                         handleChange} />
                         <input type="email" name="email" id="email"
                         placeholder="Email" value={email} onChange={this.
                         handleChange} />
                         <textarea name="message" id="message"
                         placeholder="Message" rows="7" value={message}
                         onChange={this.handleChange}></textarea>
                         <div className="actions">
                             <input type="submit" value="Send Message"
                             className="button__primary" />
                         </div>
                     </div>
                 </form>
```

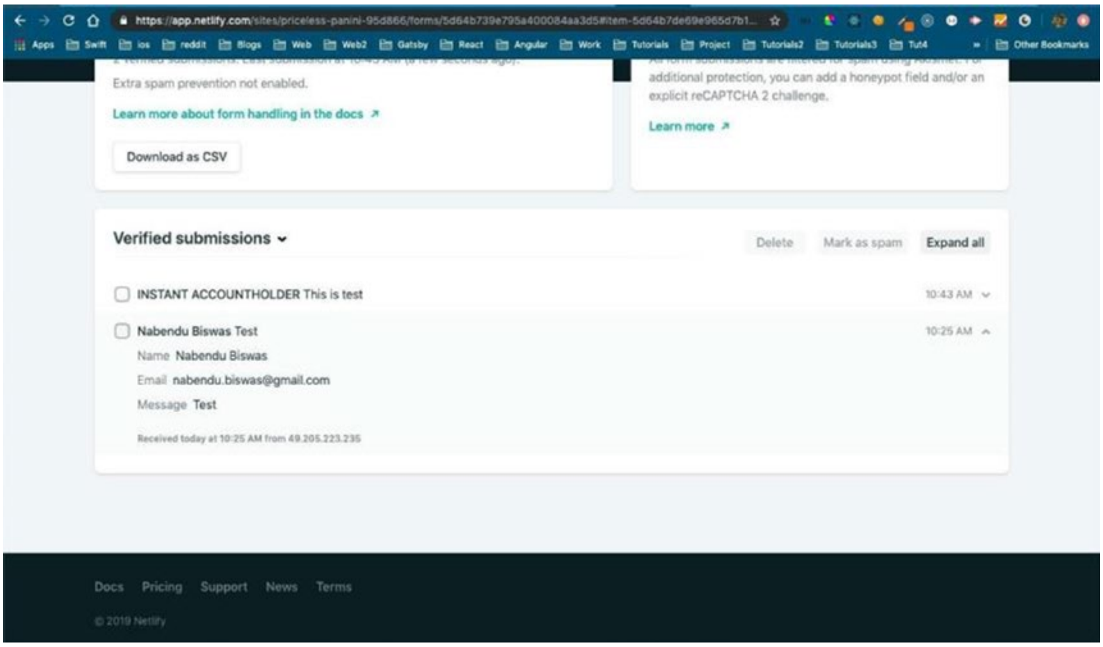
```
            </FormFive>
        </section>
    </Layout>
 )
}
}

export default IndexPage;
```

We also need to add a `thanks.js` file inside the pages folder. The content is shown in Listing 1-48.

Listing 1-48. The thanks.js File

```
import React from 'react'
import Layout from "../components/layout";
import { GenericH2, GenereicParaAbout } from "../styles/IndexStyles";

export default () => (
 <Layout>
   <GenericH2 none dark some style={{textAlign: 'center', padding:
   '3rem'}}>Thank you!</GenericH2>
   <GenereicParaAbout lessSize grey>Form submission has been successful</
   GenereicParaAbout>
 </Layout>
)
```

Now, when we submit the form, we receive the message shown in Figure 1-17 in our Netlify console.


https://app.netlify.com/sites/priceless-panini-95d866/forms/5d64b739e795a400084aa3d5#item-5d64b7de69e965d7b1...

Extra spam prevention not enabled.

Learn more about form handling in the docs ↗

Download as CSV

additional protection, you can add a honeypot field and/or an explicit reCAPTCHA 2 challenge.

Learn more ↗

Verified submissions

Delete | Mark as spam | Expand all

INSTANT ACCOUNTHOLDER This is test — 10:43 AM

Nabendu Biswas Test — 10:25 AM

Name Nabendu Biswas

Email nabendu.biswas@gmail.com

Message Test

Received today at 10:25 AM from 49.205.223.235

Figure 1-17. *The message was received*

You can find the complete code for the project at https://github.com/nabendu82/AgencyDemo.[5]

The Netlify deployment is at https://priceless-panini-95d866.netlify.com/.[6]

Summary

This completes Chapter 1 and our AgencyDemo project. You can use this site whenever you want to start your freelancing business. We covered the following topics in this chapter.

- Creating a Gatsby site with the most basic starter kit
- Using styled components in the project
- Adding a contact form to the project
- Deploying a site with Netlify

In the next chapter, we are going to learn how to convert our existing dev.to blogs to a personal blog site.

[5]https://github.com/nabendu82/AgencyDemo
[6]https://priceless-panini-95d866.netlify.com/

CHAPTER 2

Creating a Blog Site Using Stackbit

I have written 200 blogs at the time of this writing. It had become a habit to write technical articles. I had good traction on Medium, but once `dev.to` was launched, I started republishing my articles there as well. I got a lot of followers and views in `dev.to`.

I decided it is time for me to create my own blog site and I considered using GatsbyJS to do it. There are many ways to create a blog site with GatsbyJS, but I found an article[1] by Ben Halpern[2] (creator of `dev.to`) about its collaboration with Stackbit.

You need to follow the very simple process in this[3] article and your personal blog site will be ready in no time.

The Setup Process

You need to follow four steps:

1. Visit the Stackbit Creation Workflow[4] site and choose a theme.

   I chose Fjord, as shown in Figure 2-1.

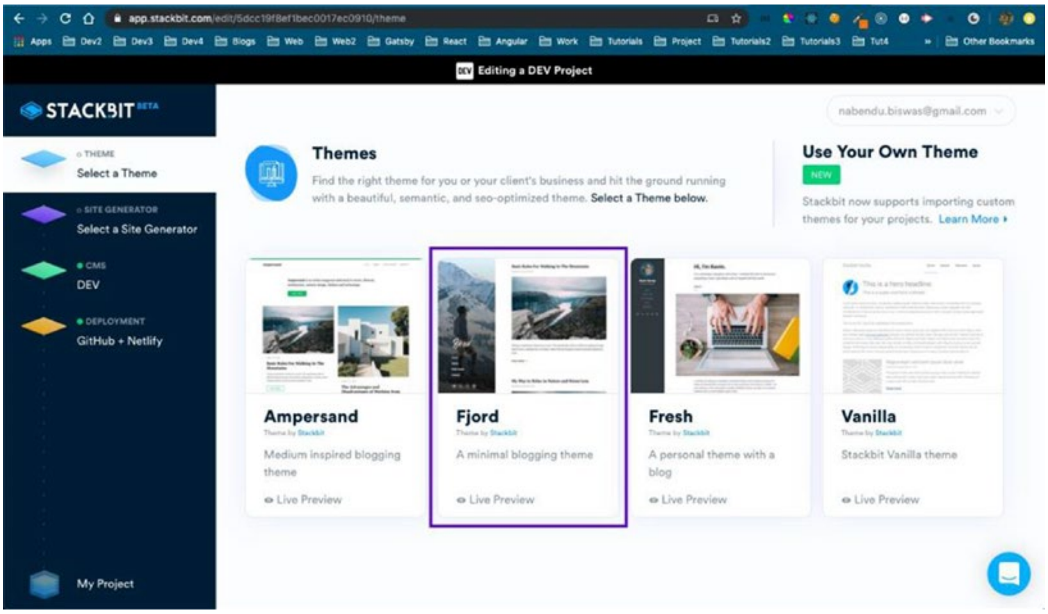
[1] https://dev.to/devteam/you-can-now-generate-self-hostable-static-blogs-right-from-your-dev-content-via-stackbit-7a5

[2] https://twitter.com/bendhalpern

[3] https://dev.to/connecting-with-stackbit

[4] https://app.stackbit.com/edit/5dcc19f8ef1bec0017ec0910/theme

© Nabendu Biswas 2021
N. Biswas, *Foundation Gatsby Projects*, https://doi.org/10.1007/978-1-4842-6558-1_2



Find the right theme for you or your client's business and hit the ground running with a beautiful, semantic, and seo-optimized theme. Select a Theme below.

Stackbit now supports importing custom themes for your projects. Learn More ▸

Ampersand — Theme by Stackbit — Medium inspired blogging theme

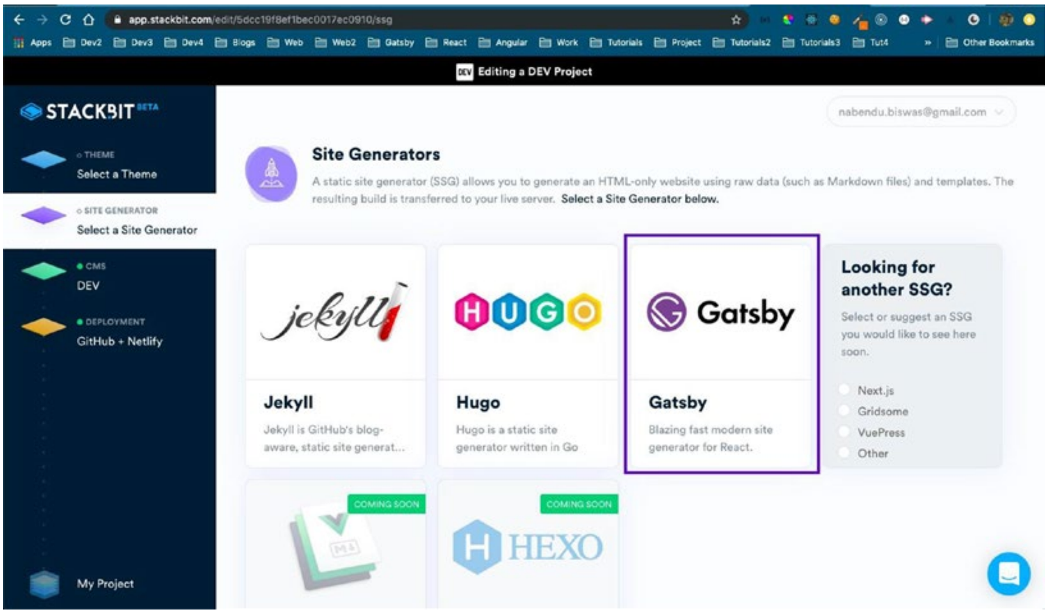
Fjord — Theme by Stackbit — A minimal blogging theme

Fresh — Theme by Stackbit — A personal theme with a blog

Vanilla — Theme by Stackbit — Stackbit Vanilla theme

Figure 2-1. *Choose a theme*

2. Then you need to choose your static site generator. Here I chose Gatsby, as shown in Figure 2-2.



A static site generator (SSG) allows you to generate an HTML-only website using raw data (such as Markdown files) and templates. The resulting build is transferred to your live server. Select a Site Generator below.

Jekyll — Jekyll is GitHub's blog-aware, static site generat...

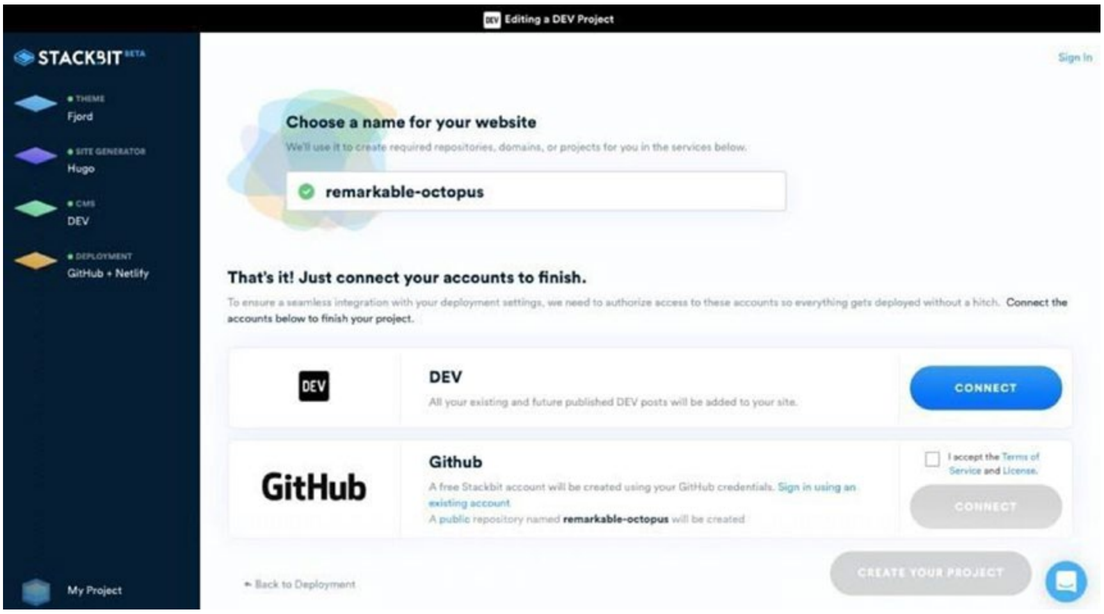
Hugo — Hugo is a static site generator written in Go

Gatsby — Blazing fast modern site generator for React.

Select or suggest an SSG you would like to see here soon.

Figure 2-2. *Choose Gatsby for the site generator*

3. Then you need to connect to your GitHub and DEV accounts, as shown in Figure 2-3.



Choose a name for your website

We'll use it to create required repositories, domains, or projects for you in the services below.

remarkable-octopus

That's it! Just connect your accounts to finish.

To ensure a seamless integration with your deployment settings, we need to authorize access to these accounts so everything gets deployed without a hitch. Connect the accounts below to finish your project.

DEV

All your existing and future published DEV posts will be added to your site.

Github

A free Stackbit account will be created using your GitHub credentials. Sign in using an existing account

A public repository named remarkable-octopus will be created

☐ I accept the Terms of Service and License.

Figure 2-3. *Connect to DEV and GitHub*

4. Next, you need to click Create Your Project. Stackbit will create a new blog site for you.

Then you need to connect to a Netlify account to keep the site live, as shown in Figure 2-4.

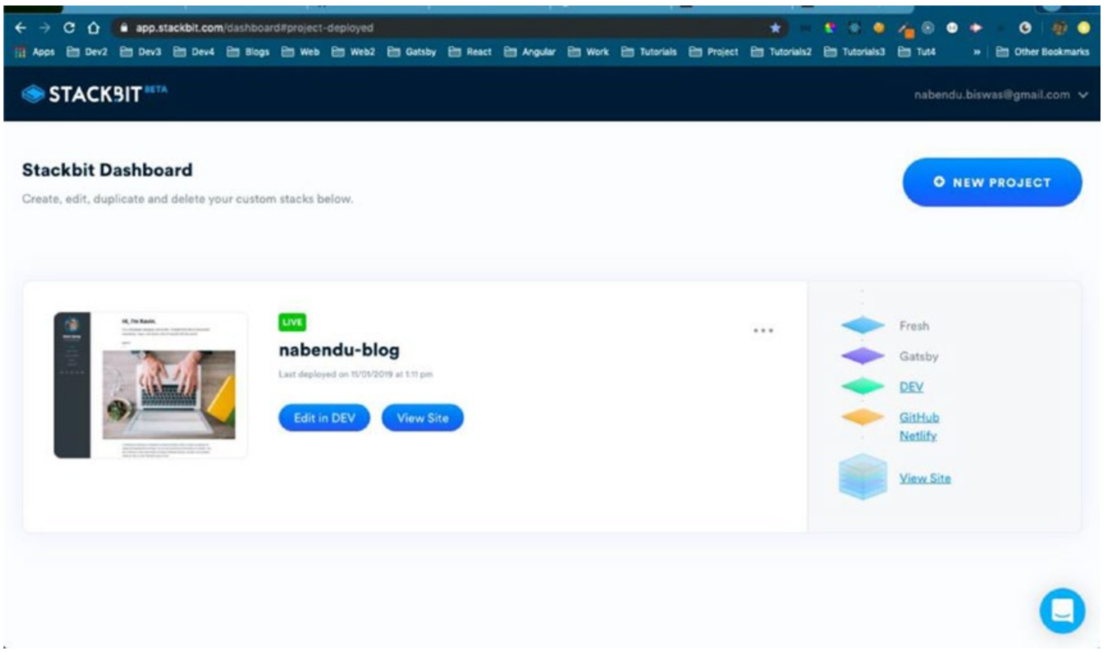
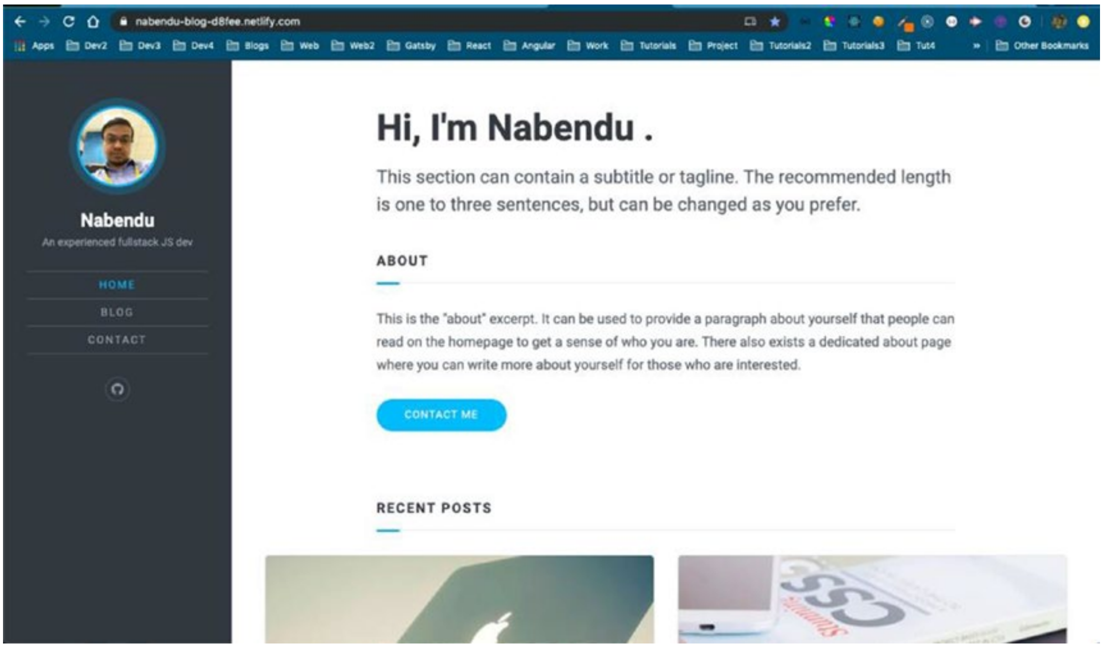

app.stackbit.com/dashboard#project-deployed
STACKBIT BETA
nabendu.biswas@gmail.com
Stackbit Dashboard
Create, edit, duplicate and delete your custom stacks below.
NEW PROJECT
LIVE
nabendu-blog
Last deployed on 11/01/2019 at 1:11 pm
Edit in DEV
View Site
Fresh
Gatsby
DEV
GitHub
Netlify
View Site

Figure 2-4. The site is live

By following these simple steps, I took my blog site live, as shown in Figure 2-5. It's found at `https://nabendu-blog-d8fee.netlify.com/`.[5]

[5]`https://nabendu-blog-d8fee.netlify.com/`



Hi, I'm Nabendu .

This section can contain a subtitle or tagline. The recommended length is one to three sentences, but can be changed as you prefer.

ABOUT

This is the "about" excerpt. It can be used to provide a paragraph about yourself that people can read on the homepage to get a sense of who you are. There also exists a dedicated about page where you can write more about yourself for those who are interested.

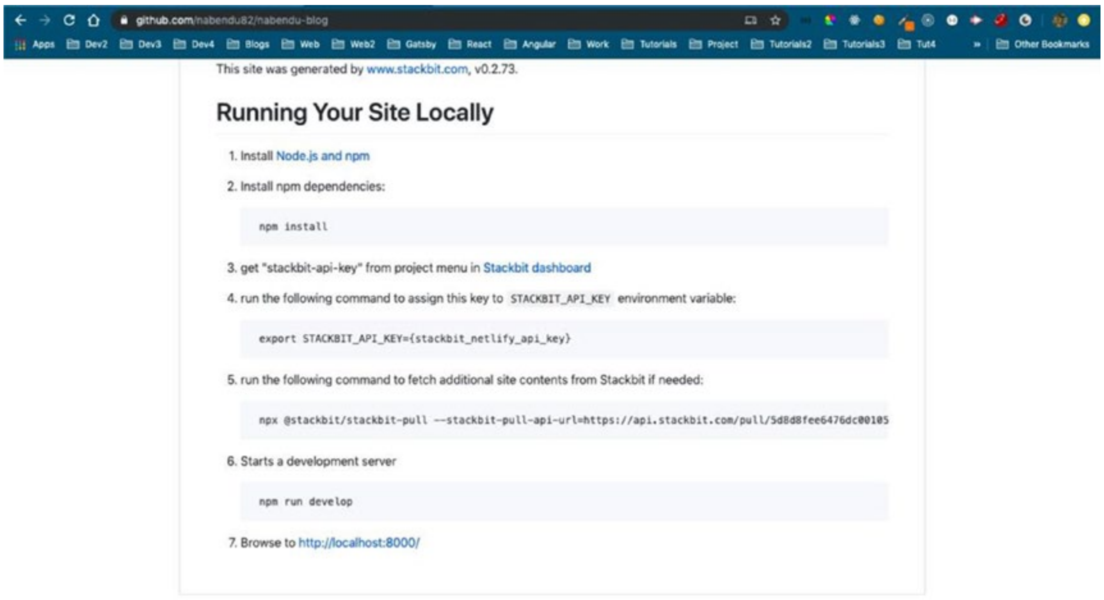
Figure 2-5. *The live site*

Now, whenever I publish a post in DEV (`dev.to`), it will be reflected on this site. This makes DEV a headless CMS for my site.

As you can see in Figure 2-5, there is some work to be done, like editing the home page, buying a good domain name, connecting in Netlify, and adding some Gatsby plugins.

Stackbit created a repository on GitHub[6]. So, I am heading over there and following the instructions shown in Figure 2-6.

[6]`https://github.com/nabendu82/nabendu-blog`



This site was generated by www.stackbit.com, v0.2.73.

Running Your Site Locally

1. Install Node.js and npm
2. Install npm dependencies:

```
npm install
```

3. get "stackbit-api-key" from project menu in Stackbit dashboard
4. run the following command to assign this key to `STACKBIT_API_KEY` environment variable:

```
export STACKBIT_API_KEY={stackbit_netlify_api_key}
```

5. run the following command to fetch additional site contents from Stackbit if needed:

```
npx @stackbit/stackbit-pull --stackbit-pull-api-url=https://api.stackbit.com/pull/5d8d8fee6476dc00105
```

6. Starts a development server

```
npm run develop
```

7. Browse to http://localhost:8000/

Figure 2-6. *The Stackbit instructions*

First I will `git clone` my repository in my desktop with the following command:

```
git clone https://github.com/nabendu82/nabendu-blog.git
```

Next, I will change to the directory and run `npm install`, with this command:

```
cd nabendu-blog
npm install
```

Now, we need to go to the Stackbit dashboard[7] to get our API keys, as shown in Figure 2-7.

[7]https://app.stackbit.com/dashboard

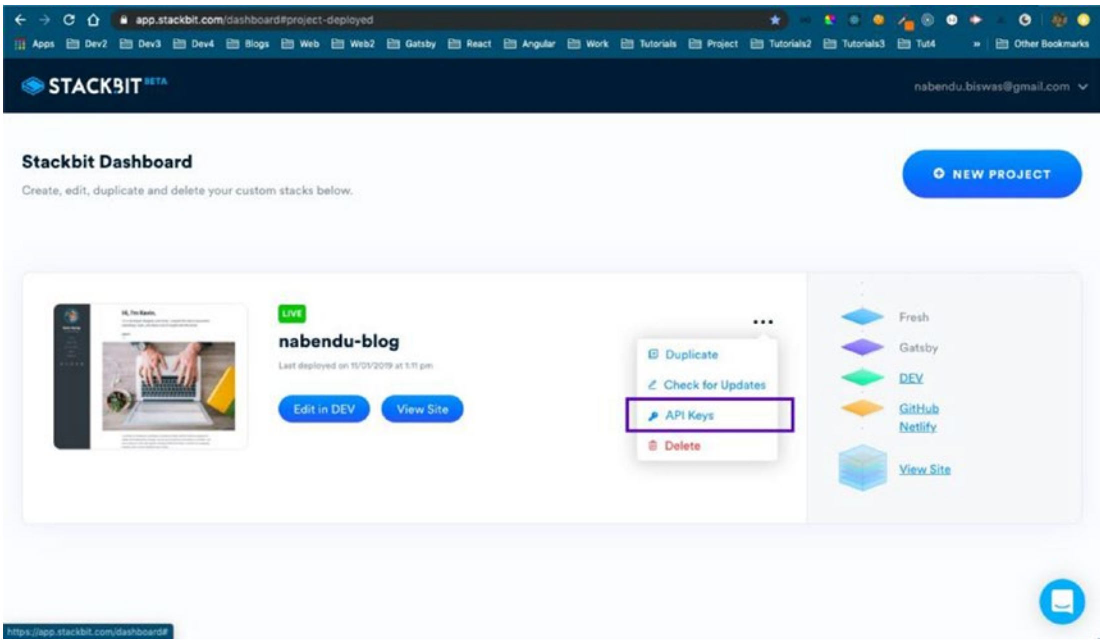


Figure 2-7. *The API keys*

Then we need to assign this key to STACKBIT_API_KEY:

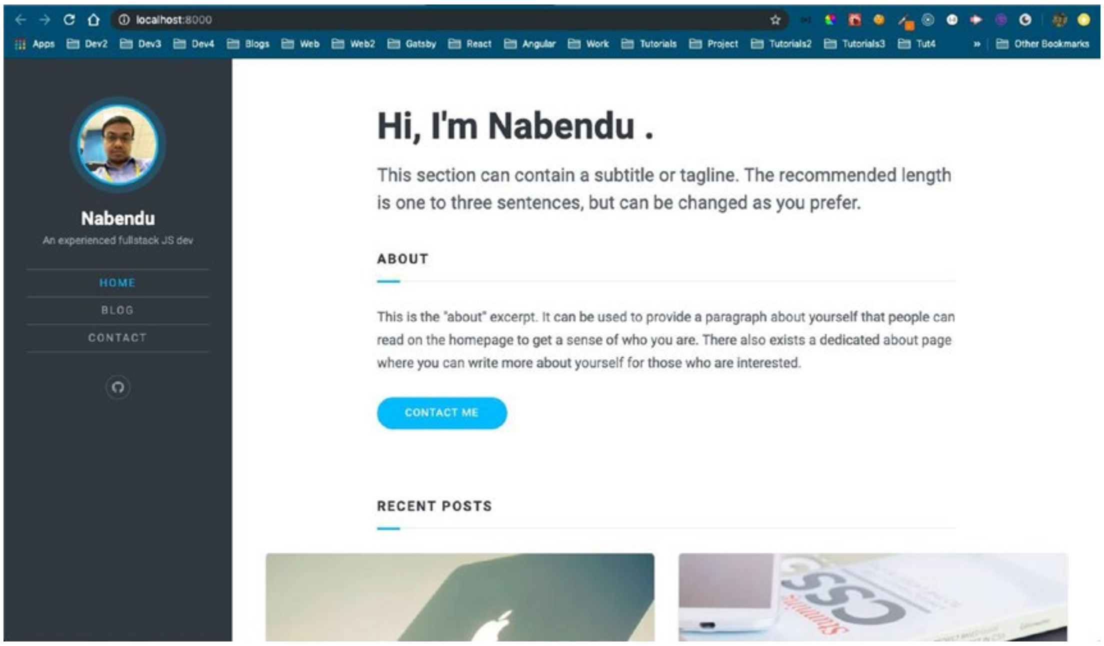
```
export STACKBIT_API_KEY=your_api_key
```

Then, as per the GitHub instruction, run the following command. It seems to fetch all posts.

```
npx @stackbit/stackbit-pull --stackbit-pull-api-url=https://api.
stackbit.com/pull/5d8d8fee6476dc00105e91ac
```

Then, as per the instructions, run `npm run develop`**.** Once it compiles successfully, open `http://localhost:8000/`[8]. Figure 2-8 shows that we successfully created our local development environment.

[8]http://localhost:8000/



Hi, I'm Nabendu .

This section can contain a subtitle or tagline. The recommended length is one to three sentences, but can be changed as you prefer.

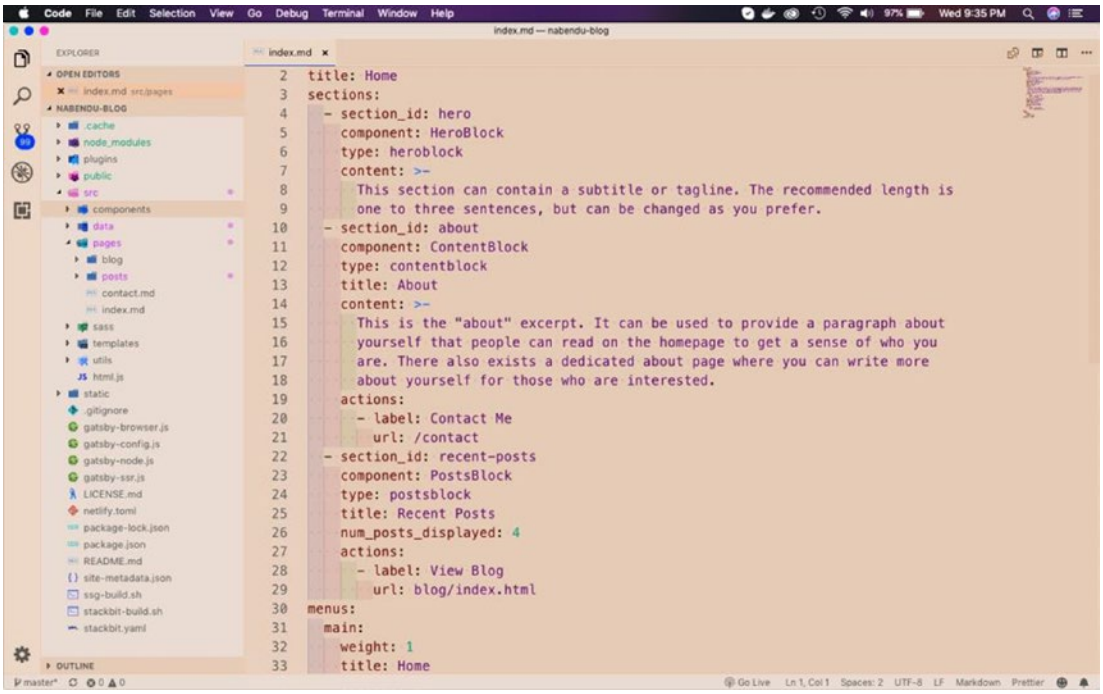
ABOUT

This is the "about" excerpt. It can be used to provide a paragraph about yourself that people can read on the homepage to get a sense of who you are. There also exists a dedicated about page where you can write more about yourself for those who are interested.

RECENT POSTS

Figure 2-8. *The localhost*

Now, let's open the project in VSCode. The main thing we need to edit is the home page, as shown in Figure 2-9. The content comes from `src` ➤ `pages` ➤ `index.md`.



```
title: Home
sections:
  - section_id: hero
    component: HeroBlock
    type: heroblock
    content: >-
      This section can contain a subtitle or tagline. The recommended length is
      one to three sentences, but can be changed as you prefer.
  - section_id: about
    component: ContentBlock
    type: contentblock
    title: About
    content: >-
      This is the "about" excerpt. It can be used to provide a paragraph about
      yourself that people can read on the homepage to get a sense of who you
      are. There also exists a dedicated about page where you can write more
      about yourself for those who are interested.
    actions:
      - label: Contact Me
        url: /contact
  - section_id: recent-posts
    component: PostsBlock
    type: postsblock
    title: Recent Posts
    num_posts_displayed: 4
    actions:
      - label: View Blog
        url: blog/index.html
menus:
  main:
    weight: 1
    title: Home
```

Figure 2-9. *The index.md file*

We will now update the content (marked in bold) in `index.md`, as shown in Listing 2-1.

Listing 2-1. The index.md File

```
---
title: Home
sections:
  - section_id: hero
    component: HeroBlock
    type: heroblock
    content: >-
      UI Developer | Tech Blogger | Team Lead
  - section_id: about
    component: ContentBlock
    type: contentblock
    title: About
    content: >-
```

```
      I live in Bengaluru (India), with my kid and wife. Love Coding,
      blogging  & running. Always keep updating my technical skills in the
      ever changing technology workspace.
    actions:
      - label: Contact Me
        url: /contact
  - section_id: recent-posts
    component: PostsBlock
    type: postsblock
    title: Recent Posts
    num_posts_displayed: 4
    actions:
      - label: View Blog
        url: blog/index.html
menus:
  main:
    weight: 1
    title: Home
template: home
---
```

Let’s commit the changes and push it to our GitHub with the `git push origin master` command. It will also start updating the site in Netlify, as shown in Figure 2-10.

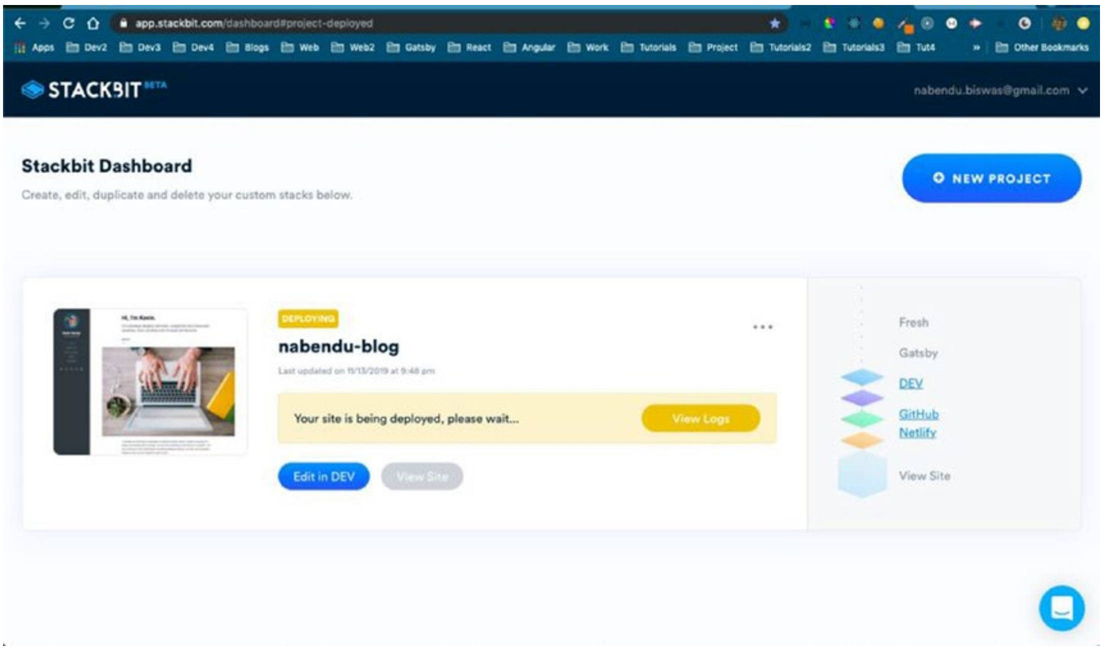


Figure 2-10. *The site is updated*

And, after some time, the changes are pushed to production, as shown in Figure 2-11.

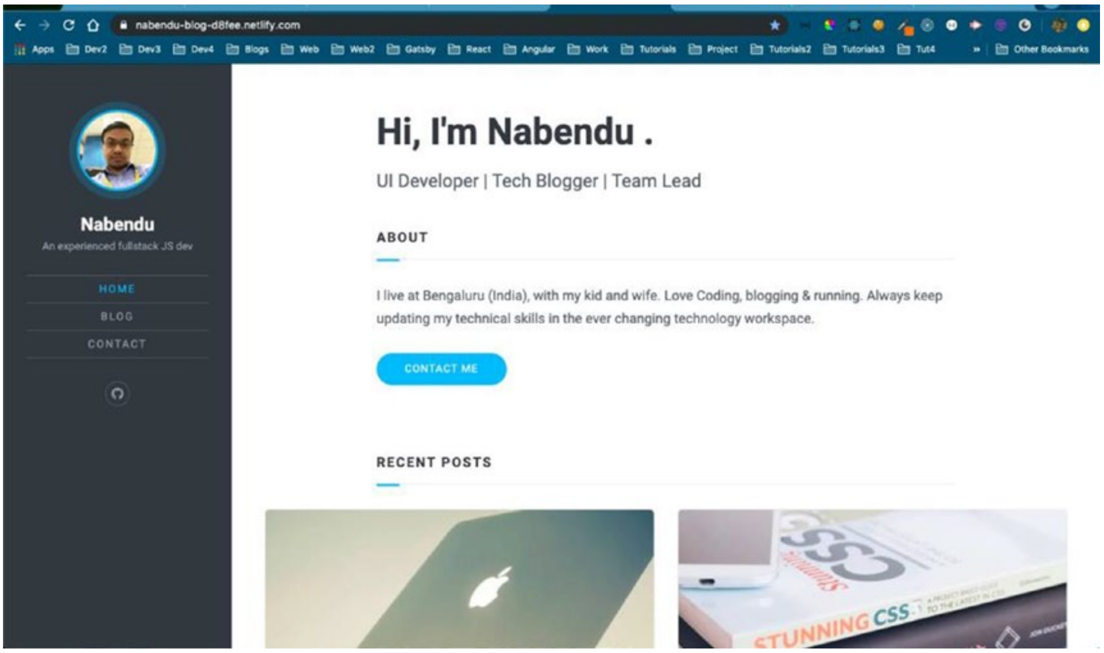

Nabendu
An experienced fullstack JS dev
HOME
BLOG
CONTACT
Hi, I'm Nabendu .
UI Developer | Tech Blogger | Team Lead
ABOUT
I live at Bengaluru (India), with my kid and wife. Love Coding, blogging & running. Always keep updating my technical skills in the ever changing technology workspace.
CONTACT ME
RECENT POSTS

Figure 2-11. *The site is updated*

Adding a Domain

In this section, we go through the process of buying a domain name from `namecheap.com` and configuring it in Netlify. It's time to buy a domain name and add the details in Netlify. To add a custom domain, I generally follow steps from this awesome Brad Traversy[9] video[10].

The first thing to do is buy a domain name. We will use the most popular service—`namecheap`.[11] Once you log in to your account, you will see the screen shown in Figure 2-12.

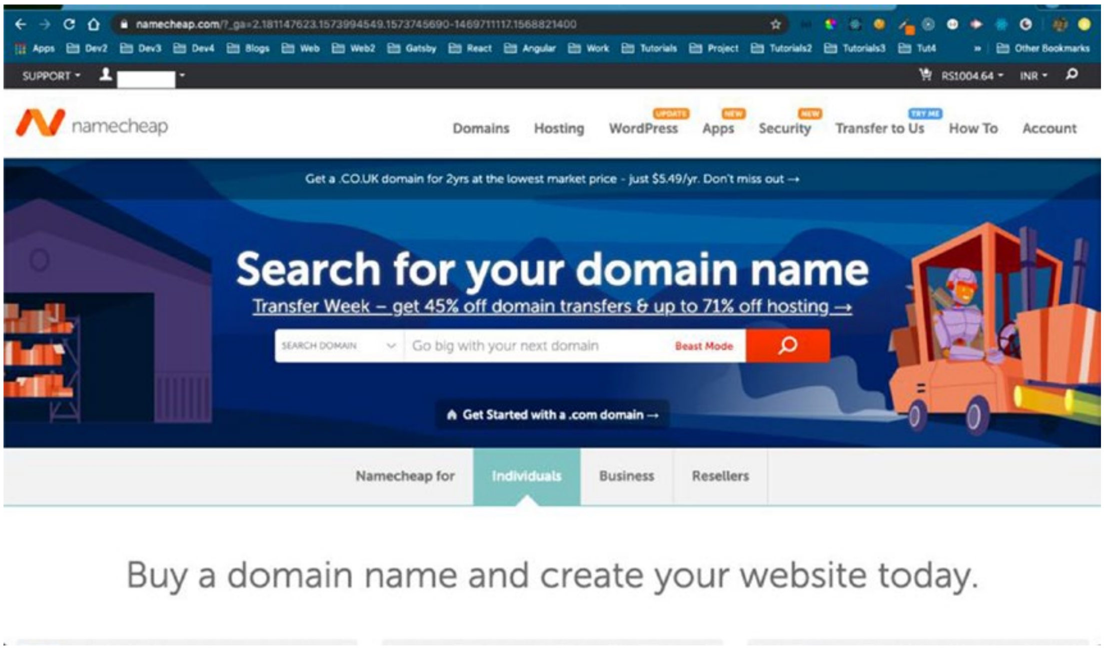

namecheap
Domains
Hosting
WordPress
Apps
Security
Transfer to Us
How To
Account
Get a .CO.UK domain for 2yrs at the lowest market price - just $5.49/yr. Don't miss out →
Search for your domain name
Transfer Week – get 45% off domain transfers & up to 71% off hosting →
SEARCH DOMAIN
Go big with your next domain
Beast Mode
Get Started with a .com domain →
Namecheap for
Individuals
Business
Resellers
Buy a domain name and create your website today.

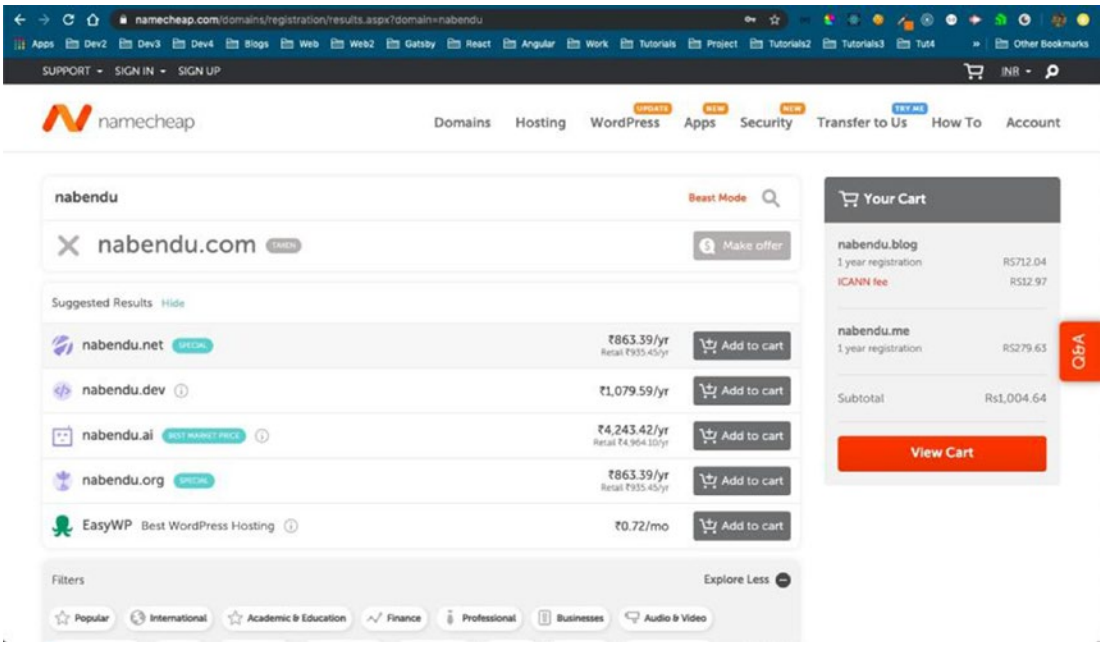
Figure 2-12. *The namecheap login*

I searched for my name and added the domain to my cart, as shown in Figure 2-13.

[9] `https://twitter.com/traversymedia`

[10] `https://www.youtube.com/watch?v=bjVUqvcCnxM&list=PLuOLMA-n-nN9TOxtU11gNLaMZBXxCtvWD&index=19&t=1000s`

[11] `http://www.namecheap.com`



nabendu

nabendu.com TAKEN

nabendu.net SPECIAL	₹863.39/yr Retail ₹935.45/yr
nabendu.dev	₹1,079.59/yr
nabendu.ai BEST MARKET PRICE	₹4,243.42/yr Retail ₹4,964.10/yr
nabendu.org SPECIAL	₹863.39/yr Retail ₹935.45/yr
EasyWP Best WordPress Hosting	₹0.72/mo

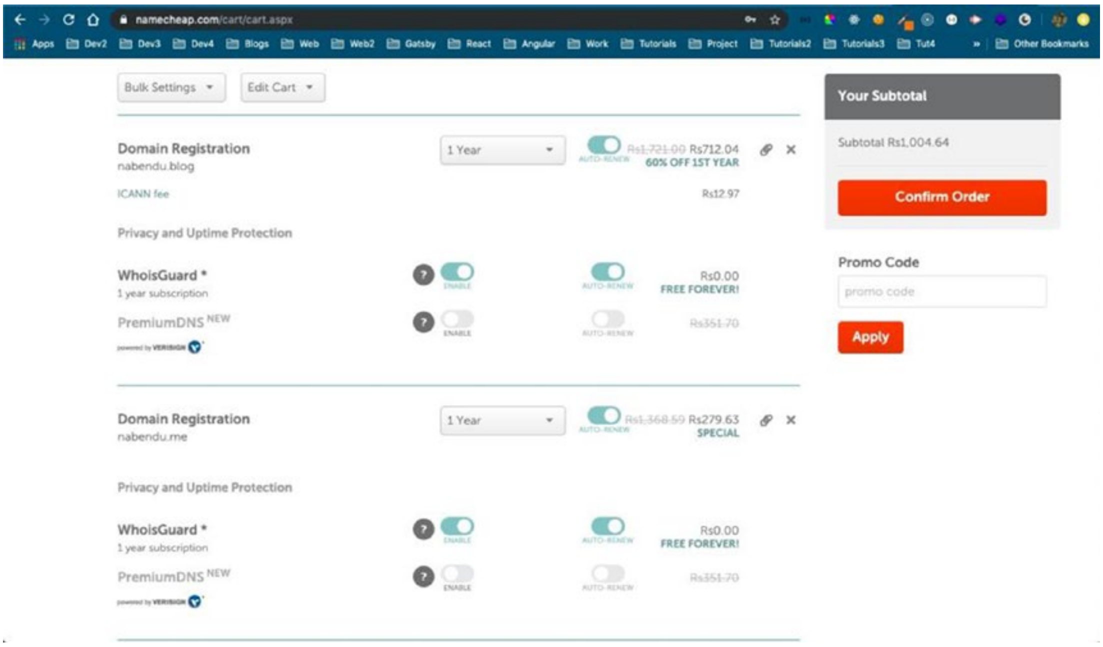
Your Cart

nabendu.blog	
1 year registration	RS712.04
ICANN fee	RS12.97
nabendu.me	
1 year registration	RS279.63
Subtotal	Rs1,004.64

View Cart

Figure 2-13. *Buying domains*

Then click View Cart, which will show the next screen. See Figure 2-14.



Domain Registration
nabendu.blog　1 Year　~~Rs1,721.00~~ Rs712.04 60% OFF 1ST YEAR
ICANN fee　Rs12.97

Privacy and Uptime Protection
WhoisGuard * 1 year subscription　Rs0.00 FREE FOREVER!
PremiumDNS NEW powered by VERISIGN　~~Rs351.70~~

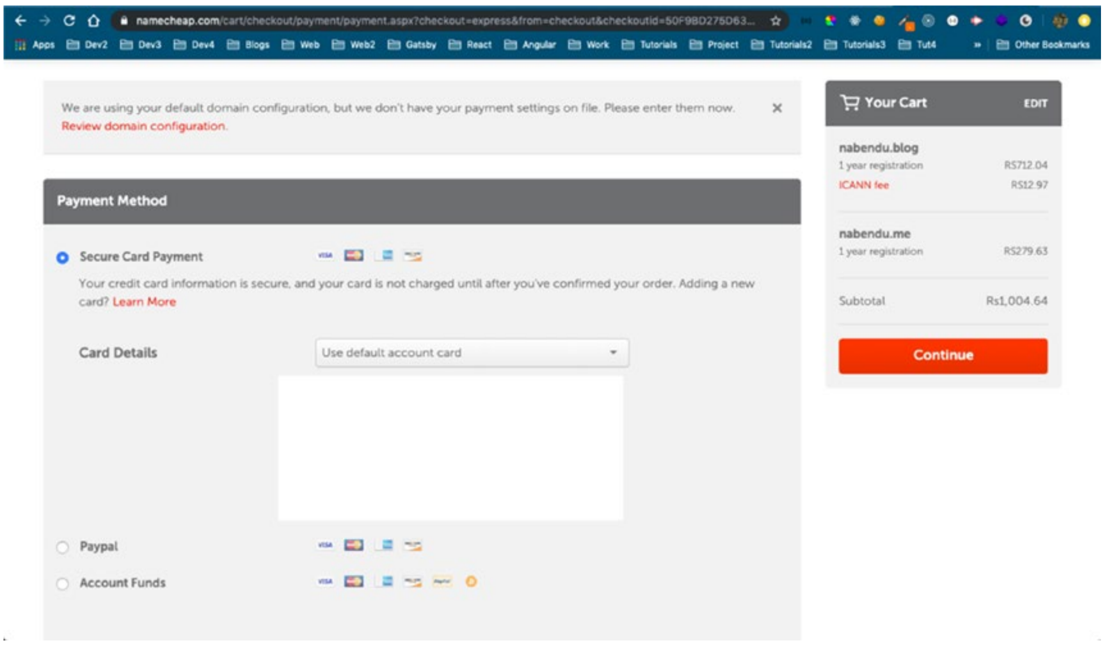
Domain Registration
nabendu.me　1 Year　~~Rs1,368.59~~ Rs279.63 SPECIAL

Privacy and Uptime Protection
WhoisGuard * 1 year subscription　Rs0.00 FREE FOREVER!
PremiumDNS NEW powered by VERISIGN　~~Rs351.70~~

Subtotal Rs1,004.64

Figure 2-14. *Buying a domain*

Once you click Confirm Order, you will be taken to the next screen, where you confirm your payment options, as shown in Figure 2-15.



namecheap.com/cart/checkout/payment/payment.aspx?checkout=express&from=checkout&checkoutid=50F9BD275D63...

We are using your default domain configuration, but we don't have your payment settings on file. Please enter them now.
Review domain configuration.

Payment Method

Secure Card Payment

Your credit card information is secure, and your card is not charged until after you've confirmed your order. Adding a new card? Learn More

Card Details: Use default account card

Paypal

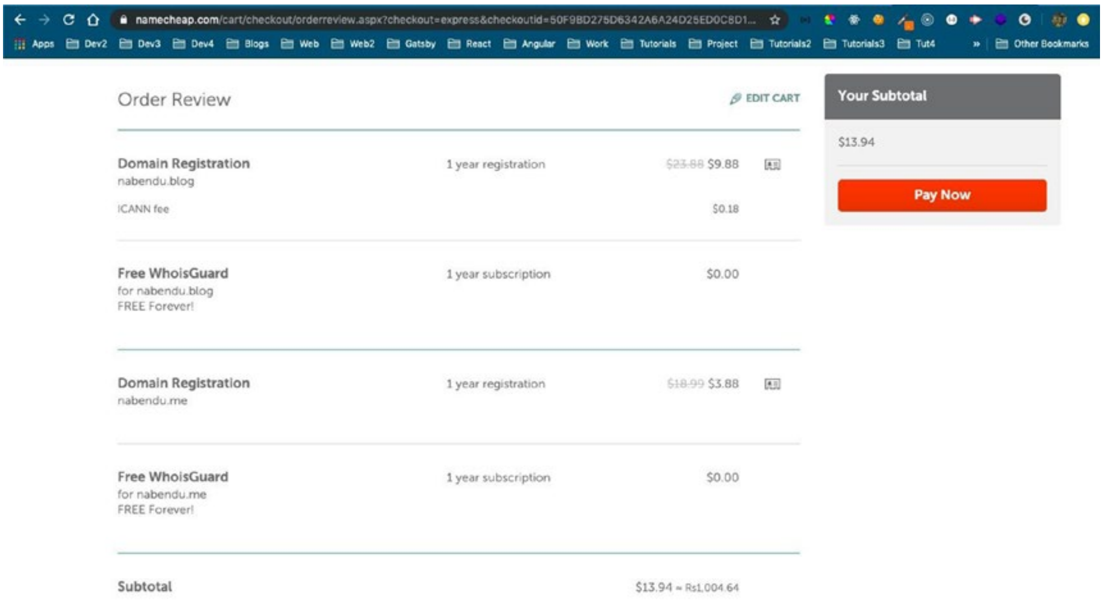
Account Funds

Your Cart EDIT

nabendu.blog	
1 year registration	RS712.04
ICANN fee	RS12.97
nabendu.me	
1 year registration	RS279.63
Subtotal	Rs1,004.64

Figure 2-15. *Payment options*

Once you click Continue, you will be taken to the screen in Figure 2-16, where you have to pay.



Order Review

EDIT CART

Domain Registration nabendu.blog	1 year registration	~~$23.88~~ $9.88	
ICANN fee		$0.18	
Free WhoisGuard for nabendu.blog FREE Forever!	1 year subscription	$0.00	
Domain Registration nabendu.me	1 year registration	~~$18.99~~ $3.88	
Free WhoisGuard for nabendu.me FREE Forever!	1 year subscription	$0.00	
Subtotal		$13.94 ≈ Rs1,004.64	

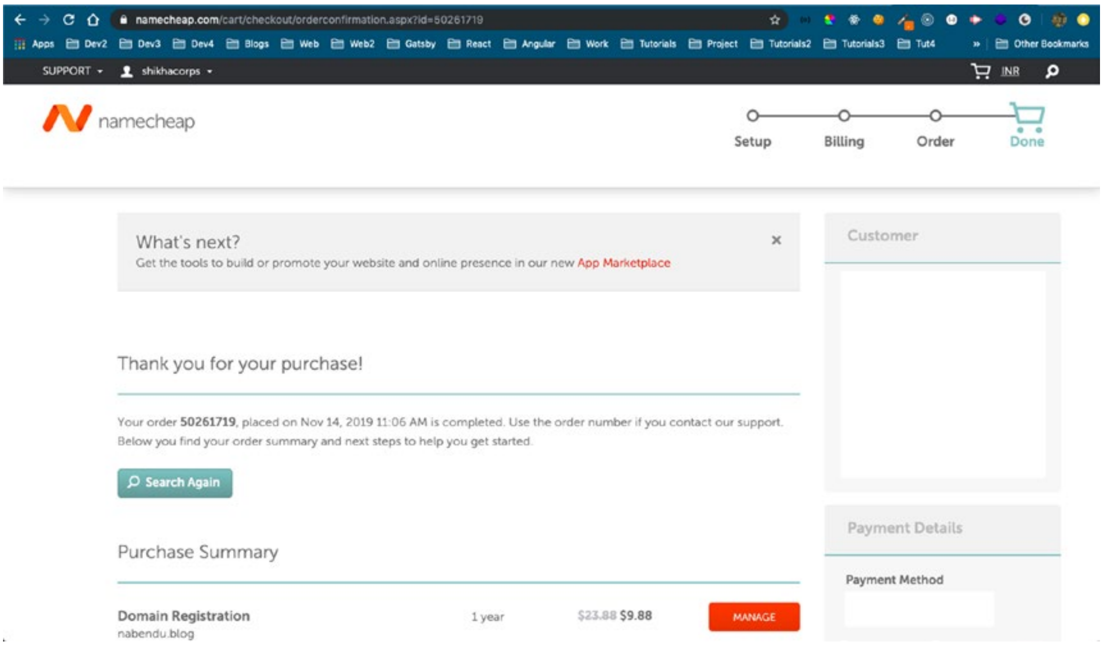
Your Subtotal

$13.94

Pay Now

Figure 2-16. The Pay screen

Once you click Pay Now, shown in Figure 2-17, the purchase will be completed.



What's next?
Get the tools to build or promote your website and online presence in our new App Marketplace

Thank you for your purchase!

Your order **50261719**, placed on Nov 14, 2019 11:06 AM is completed. Use the order number if you contact our support. Below you find your order summary and next steps to help you get started.

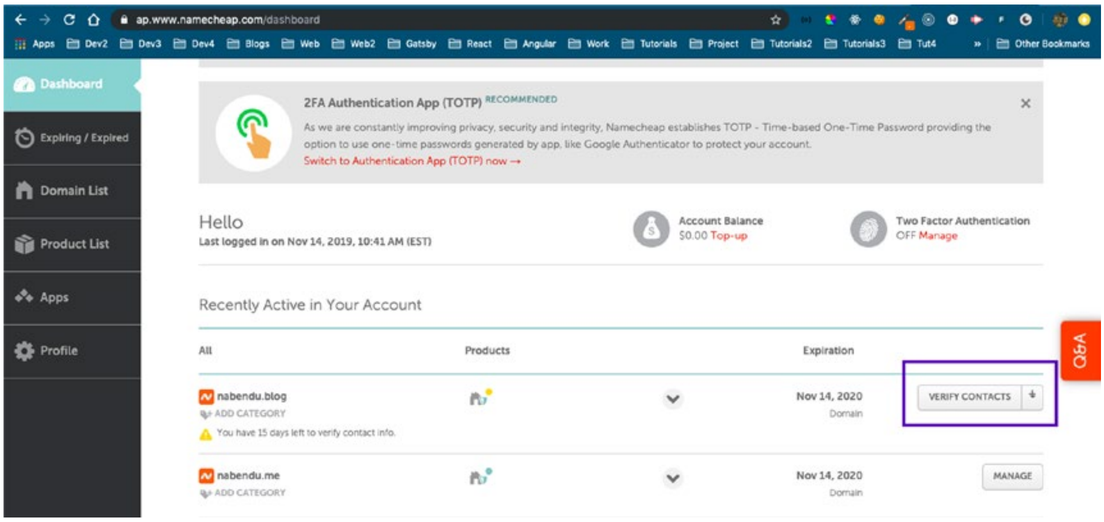
Search Again

Purchase Summary

Domain Registration nabendu.blog	1 year	~~$23.88~~ $9.88	MANAGE

Figure 2-17. The Purchase Completed screen

After this, we have to click the Manage button. This will take us to our dashboard, as shown in Figure 2-18. But strangely enough it is asking me to verify contacts. To resolve this, I clicked a link in the mail they sent.


ap.www.namecheap.com/dashboard
Dashboard
Expiring / Expired
Domain List
Product List
Apps
Profile
2FA Authentication App (TOTP) RECOMMENDED
Switch to Authentication App (TOTP) now →
Hello
Account Balance
$0.00 Top-up
Two Factor Authentication
OFF Manage
Q&A
VERIFY CONTACTS
MANAGE

As we are constantly improving privacy, security and integrity, Namecheap establishes TOTP - Time-based One-Time Password providing the option to use one-time passwords generated by app, like Google Authenticator to protect your account.

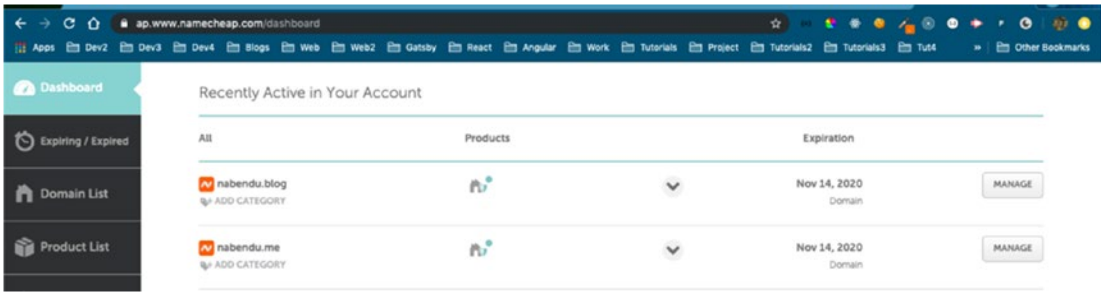
Last logged in on Nov 14, 2019, 10:41 AM (EST)

Recently Active in Your Account

All	Products	Expiration	
nabendu.blog ADD CATEGORY You have 15 days left to verify contact info.		Nov 14, 2020 Domain	VERIFY CONTACTS
nabendu.me ADD CATEGORY		Nov 14, 2020 Domain	MANAGE

Figure 2-18. *The error*

The error was resolved in an hour and I got a Manage button, as shown in Figure 2-19.



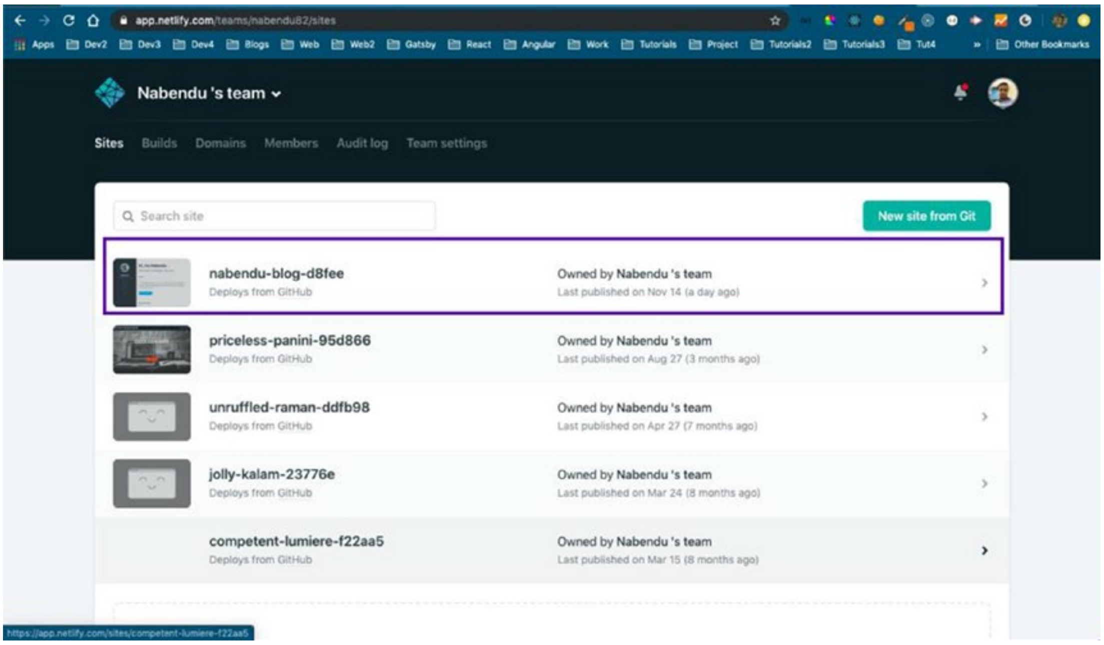
Recently Active in Your Account

All	Products	Expiration	
nabendu.blog ADD CATEGORY		Nov 14, 2020 Domain	MANAGE
nabendu.me ADD CATEGORY		Nov 14, 2020 Domain	MANAGE

Figure 2-19. *The Manage button*

Next, open Netlify[12] and log in to your account. You will see the blog site we created earlier, as shown in Figure 2-20.

[12]`https://app.netlify.com/`



nabendu-blog-d8fee — Deploys from GitHub — Owned by Nabendu 's team — Last published on Nov 14 (a day ago)

priceless-panini-95d866 — Deploys from GitHub — Owned by Nabendu 's team — Last published on Aug 27 (3 months ago)

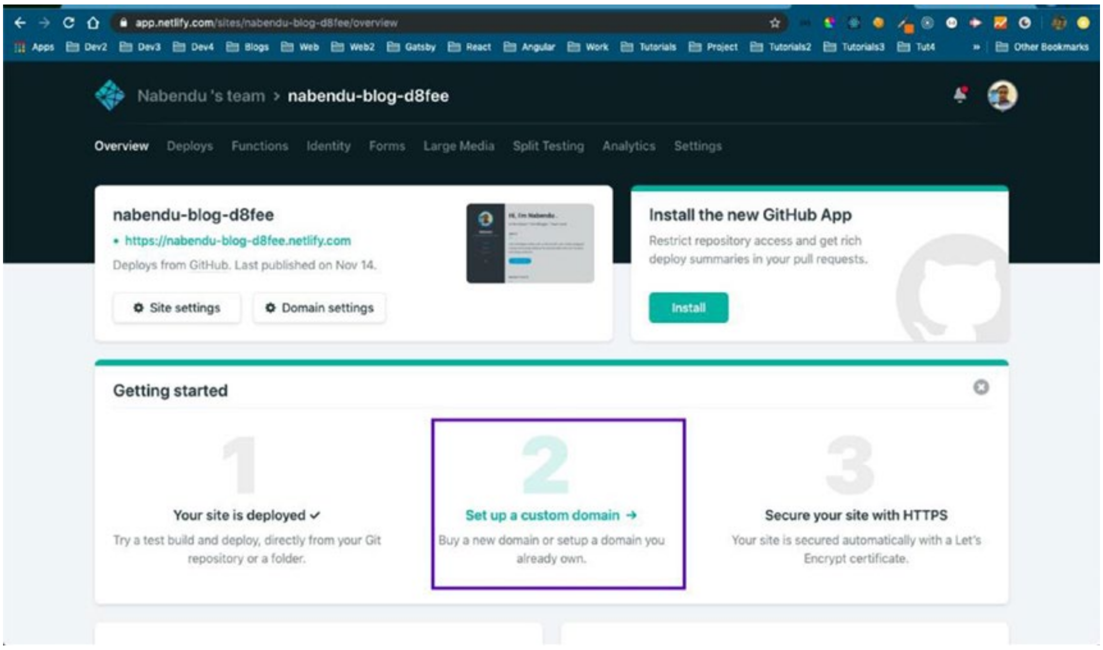
unruffled-raman-ddfb98 — Deploys from GitHub — Owned by Nabendu 's team — Last published on Apr 27 (7 months ago)

jolly-kalam-23776e — Deploys from GitHub — Owned by Nabendu 's team — Last published on Mar 24 (8 months ago)

competent-lumiere-f22aa5 — Deploys from GitHub — Owned by Nabendu 's team — Last published on Mar 15 (8 months ago)

Figure 2-20. *My blog*

In the next screen, click Set Up a Custom Domain, as shown in Figure 2-21.



nabendu-blog-d8fee
https://nabendu-blog-d8fee.netlify.com
Deploys from GitHub. Last published on Nov 14.

Install the new GitHub App
Restrict repository access and get rich deploy summaries in your pull requests.

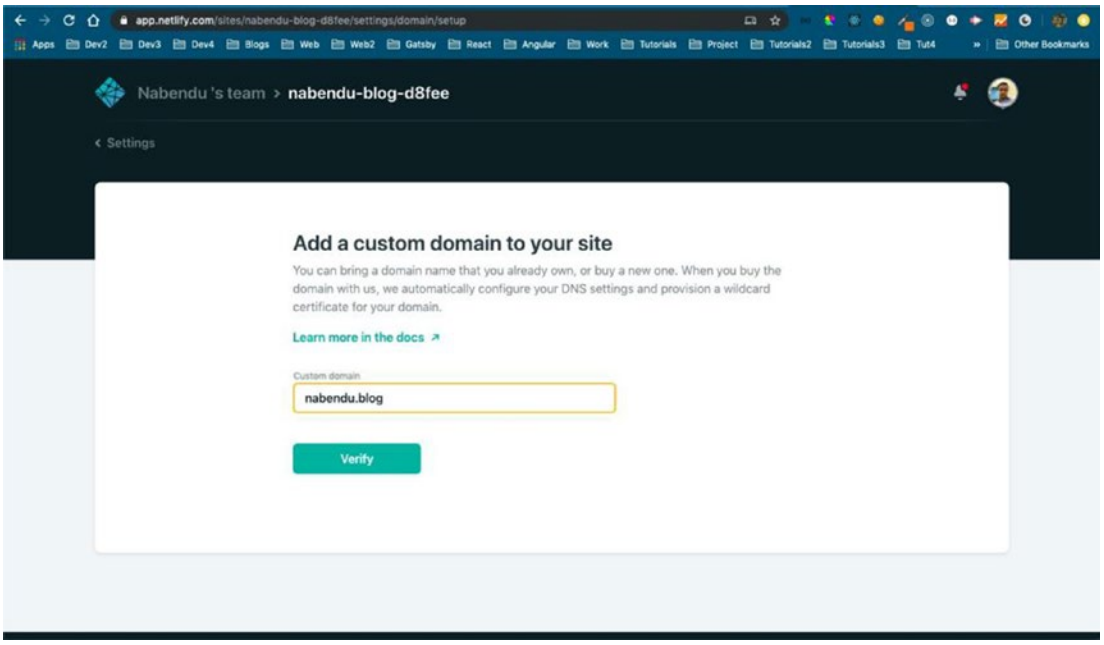
1 Your site is deployed ✓
Try a test build and deploy, directly from your Git repository or a folder.

2 Set up a custom domain →
Buy a new domain or setup a domain you already own.

3 Secure your site with HTTPS
Your site is secured automatically with a Let's Encrypt certificate.

Figure 2-21. *The Custom domain*

In the next screen, I will give the domain name (`nabendu.blog`), which I purchased from namecheap. Click the Verify button, as shown in Figure 2-22.


app.netlify.com/sites/nabendu-blog-d8fee/settings/domain/setup
Nabendu 's team > nabendu-blog-d8fee
Settings

Add a custom domain to your site

You can bring a domain name that you already own, or buy a new one. When you buy the domain with us, we automatically configure your DNS settings and provision a wildcard certificate for your domain.

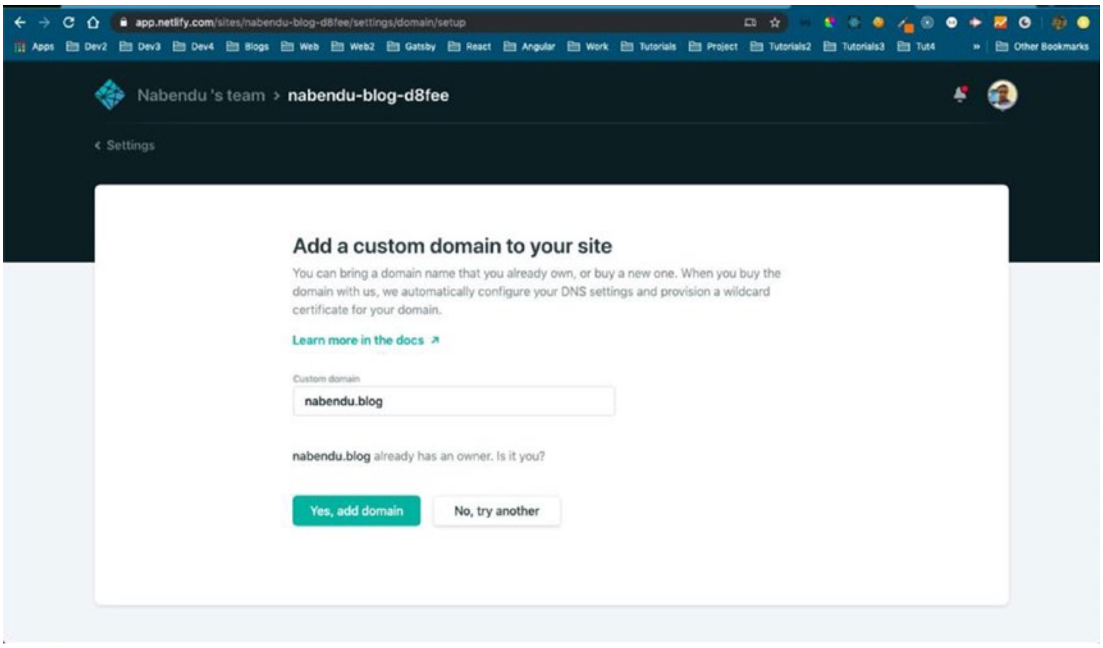
Learn more in the docs ↗

Custom domain: nabendu.blog

Verify

Figure 2-22. nabendu.blog

Next, it will ask whether I am the owner, so I will click the Yes, Add Domain button, as shown in Figure 2-23.


Add a custom domain to your site

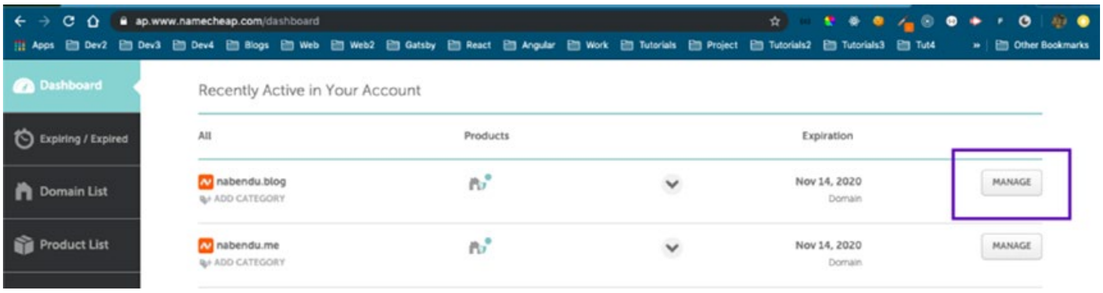
You can bring a domain name that you already own, or buy a new one. When you buy the domain with us, we automatically configure your DNS settings and provision a wildcard certificate for your domain.

Custom domain: nabendu.blog

nabendu.blog already has an owner. Is it you?

Figure 2-23. *Add the custom domain*

Now, it's time to go back to `namecheap.com` and click the Manage button in `nabendu.blog` on the dashboard, as shown in Figure 2-24.


Recently Active in Your Account

All	Products		Expiration	
nabendu.blog			Nov 14, 2020 Domain	MANAGE
nabendu.me			Nov 14, 2020 Domain	MANAGE

Figure 2-24. *The Manage button*

It will open the screen shown in Figure 2-25. Click the Advanced DNS tab.

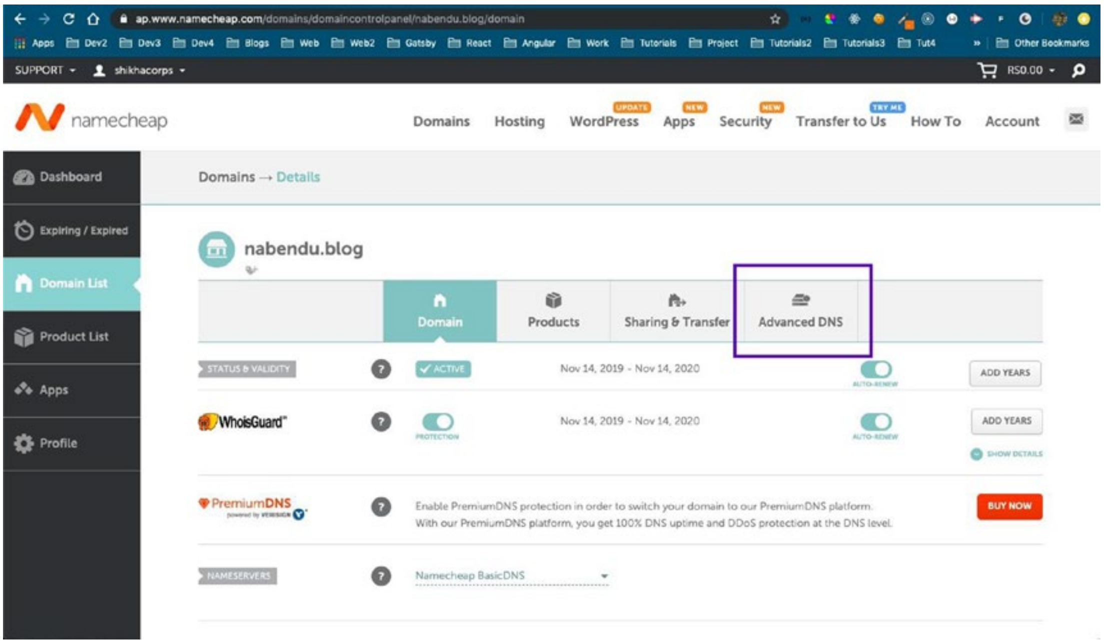
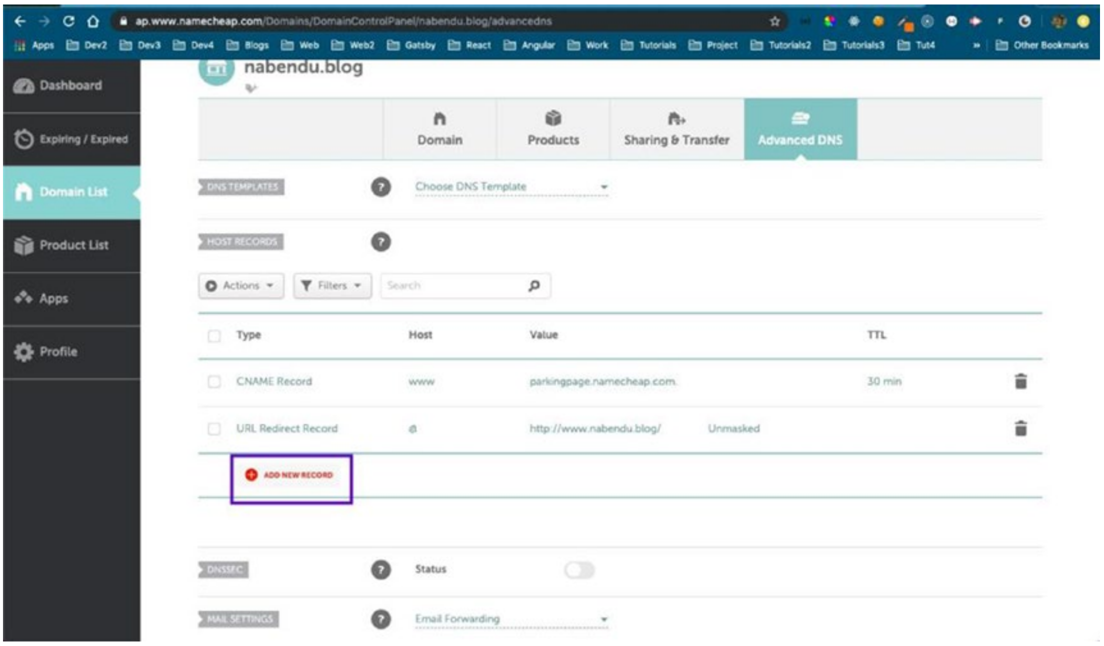


Figure 2-25. *The Advanced DNS tab*

We can use Netlify DNS to configure the DNS, but we will be using our service provider (namecheap) to do so. You can go through this[13] article for more details.

So, click the Add New Record button, as shown in Figure 2-26.

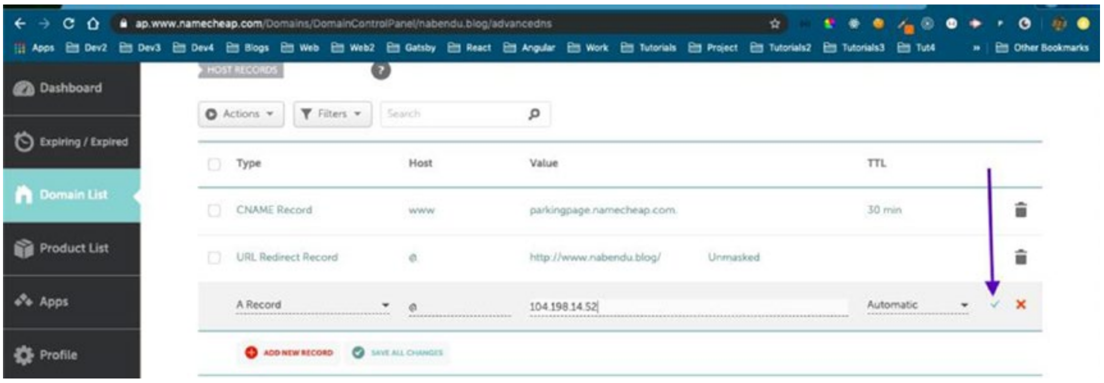
[13]https://docs.netlify.com/domains-https/custom-domains/configure-external-dns/#configure-a-subdomain



Type	Host	Value	TTL
CNAME Record	www	parkingpage.namecheap.com	30 min
URL Redirect Record	@	http://www.nabendu.blog/ Unmasked	

Figure 2-26. *Add new record*

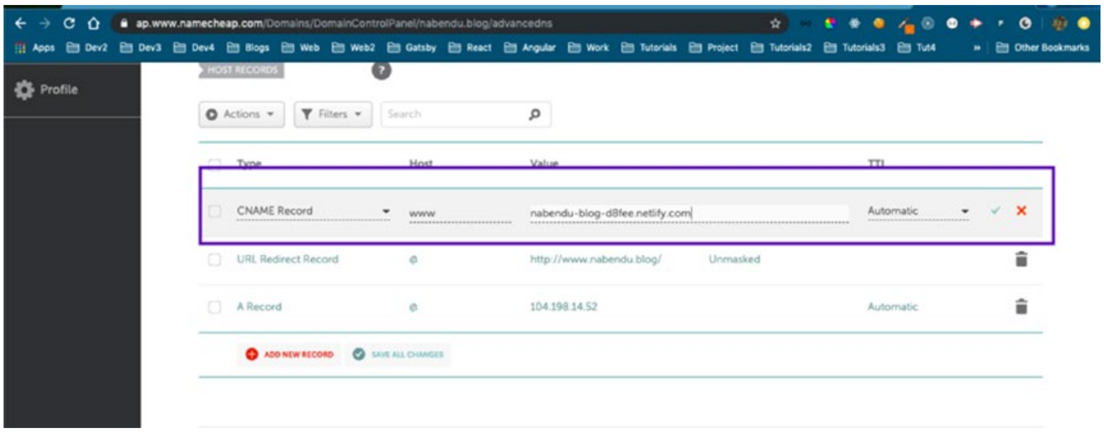
According to Netlify docs, we need to add an A record and add their load balancer IP address, which is `104.198.14.52`. Add the record and then click the small green tick mark, as shown in Figure 2-27.



Type	Host	Value	TTL
CNAME Record	www	parkingpage.namecheap.com	30 min
URL Redirect Record	@	http://www.nabendu.blog/ Unmasked	
A Record	@	104.198.14.52	Automatic

Figure 2-27. *Add an A record*

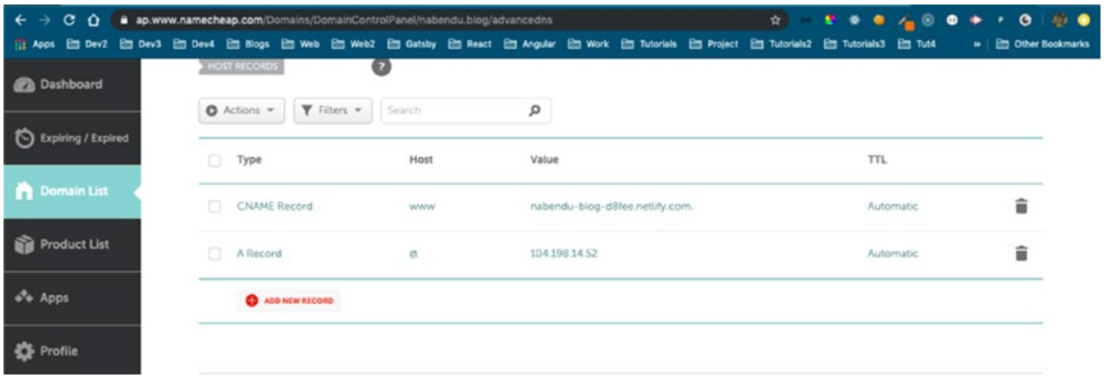
Next, we will add a CNAME record. Since I already had one, I edited it. The value should be equal to the random domain name, which Netlify gave us, as shown in Figure 2-28.



Type	Host	Value		TTL
CNAME Record	www	nabendu-blog-d8fee.netlify.com		Automatic
URL Redirect Record	@	http://www.nabendu.blog/	Unmasked	
A Record	@	104.198.14.52		Automatic

Figure 2-28. *The CNAME record*

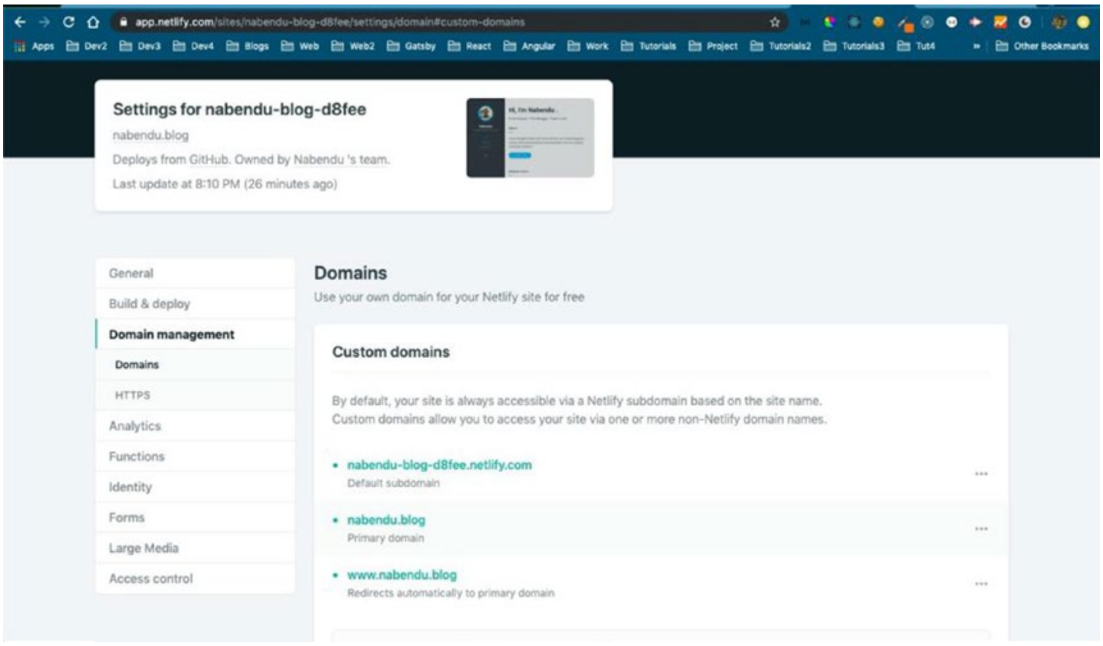
Also, let's get rid of the URL Redirect Record by clicking the Delete button in its row. The Advanced DNS page should now look like Figure 2-29.



Type	Host	Value	TTL
CNAME Record	www	nabendu-blog-d8fee.netlify.com.	Automatic
A Record	@	104.198.14.52	Automatic

Figure 2-29. *Final advanced DNS page*

Now, go to your Netlify account and refresh the web page. You will see Figure 2-30 if everything is okay.


app.netlify.com/sites/nabendu-blog-d8fee/settings/domain#custom-domains

Settings for nabendu-blog-d8fee
nabendu.blog
Deploys from GitHub. Owned by Nabendu 's team.
Last update at 8:10 PM (26 minutes ago)

Domains
Use your own domain for your Netlify site for free

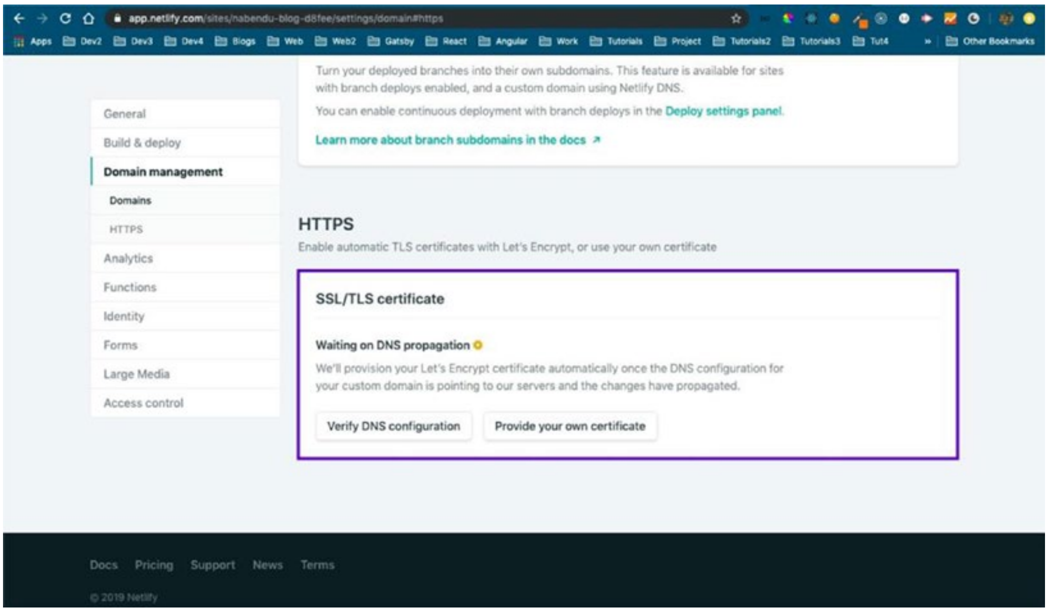
Custom domains

By default, your site is always accessible via a Netlify subdomain based on the site name.
Custom domains allow you to access your site via one or more non-Netlify domain names.

- nabendu-blog-d8fee.netlify.com
  Default subdomain
- nabendu.blog
  Primary domain
- www.nabendu.blog
  Redirects automatically to primary domain

Figure 2-30. *Everything is okay*

One of the very good features of Netlify is that it provides a secure HTTPS site. But it generally takes time. If you scroll down on the same page, it will show that it is waiting on DNS propagation, as shown in Figure 2-31.



Turn your deployed branches into their own subdomains. This feature is available for sites with branch deploys enabled, and a custom domain using Netlify DNS.

You can enable continuous deployment with branch deploys in the Deploy settings panel.

Learn more about branch subdomains in the docs ↗

HTTPS

Enable automatic TLS certificates with Let's Encrypt, or use your own certificate

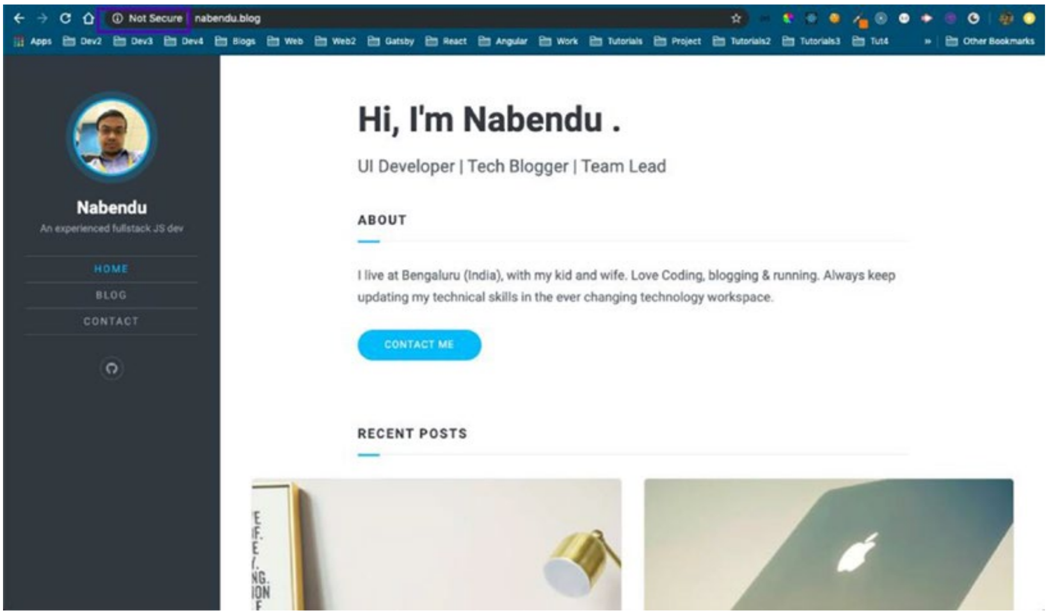
SSL/TLS certificate

Waiting on DNS propagation

We'll provision your Let's Encrypt certificate automatically once the DNS configuration for your custom domain is pointing to our servers and the changes have propagated.

Figure 2-31. *Waiting on DNS propagation*

It can take up to a day for DNS propagation. But we can view the HTTP version of our site, with the Not Secure warning in the title bar, as shown in Figure 2-32.



Hi, I'm Nabendu .

UI Developer | Tech Blogger | Team Lead

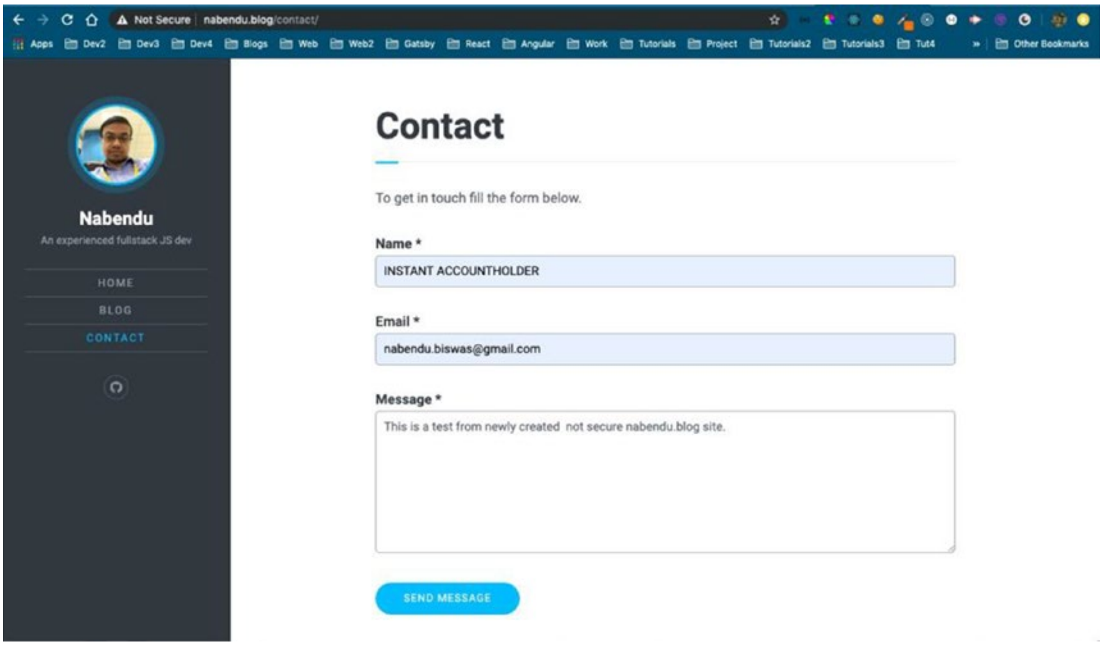
ABOUT

I live at Bengaluru (India), with my kid and wife. Love Coding, blogging & running. Always keep updating my technical skills in the ever changing technology workspace.

RECENT POSTS

Figure 2-32. *This site is not secure*

Netlify also provides a great contact form, which was given to us by Stackbit. We can check whether it is working by heading to the Contact page, as shown in Figure 2-33.


Not Secure nabendu.blog/contact/
Apps Dev2 Dev3 Dev4 Blogs Web Web2 Gatsby React Angular Work Tutorials Project Tutorials2 Tutorials3 Tut4 Other Bookmarks
Nabendu
An experienced fullstack JS dev
HOME
BLOG
CONTACT
SEND MESSAGE

Contact

To get in touch fill the form below.

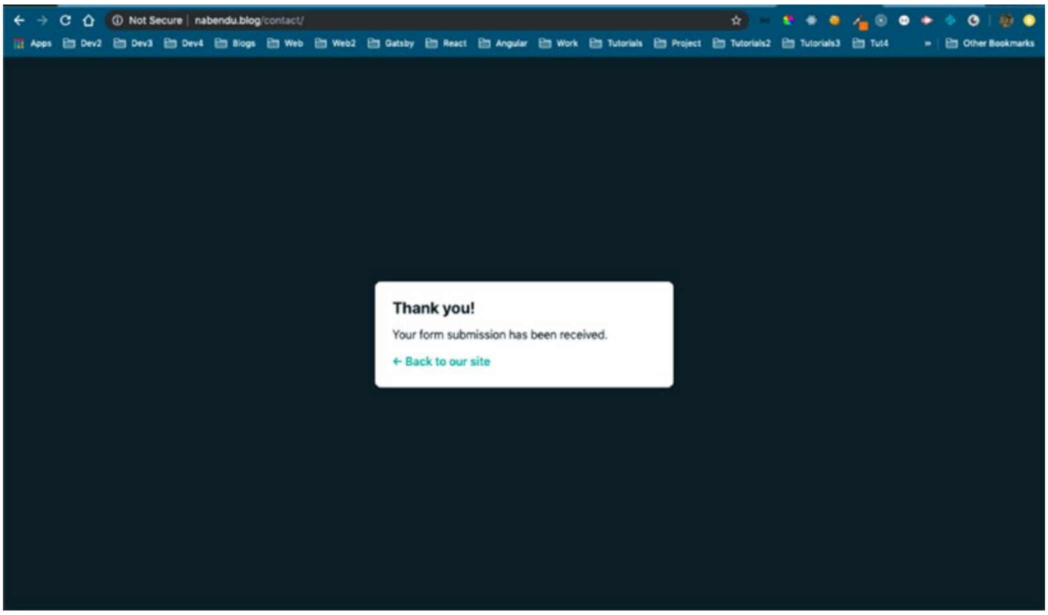
Name *
INSTANT ACCOUNTHOLDER

Email *
nabendu.biswas@gmail.com

Message *
This is a test from newly created not secure nabendu.blog site.

Figure 2-33. *Submitting a form*

If it is successful, we will get the screen shown in Figure 2-34.


Not Secure | nabendu.blog/contact/

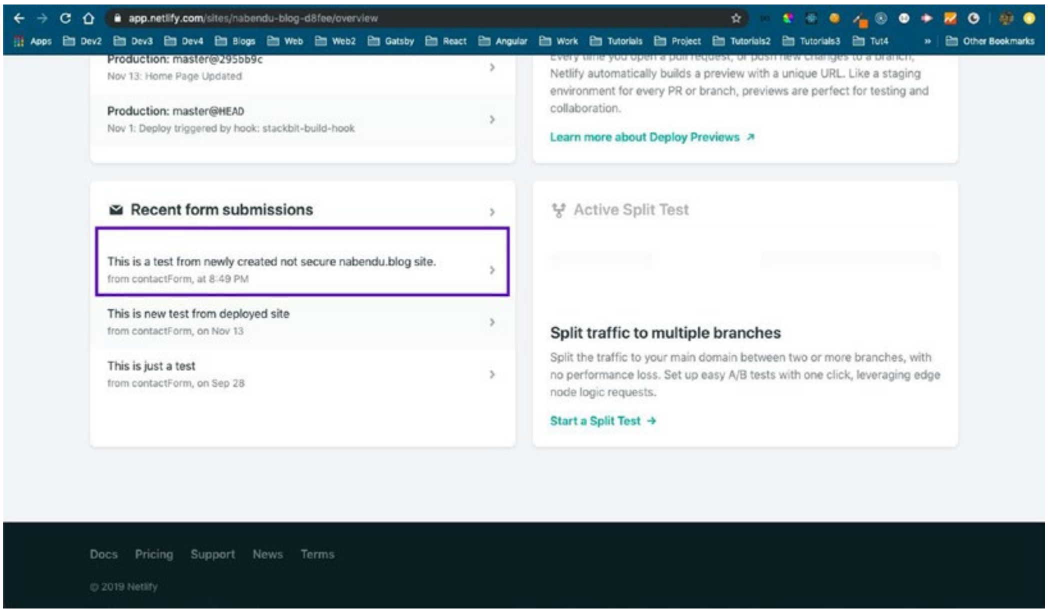
Thank you!

Your form submission has been received.

← Back to our site

Figure 2-34. *Successful form submission*

Now, head over to Overview in Netlify and scroll down. You can see the newly sent message, as shown in Figure 2-35.


app.netlify.com/sites/nabendu-blog-d8fee/overview

Production: master@295bb9c
Nov 13: Home Page Updated

Production: master@HEAD
Nov 1: Deploy triggered by hook: stackbit-build-hook

Netlify automatically builds a preview with a unique URL. Like a staging environment for every PR or branch, previews are perfect for testing and collaboration.

Learn more about Deploy Previews ↗

Recent form submissions

This is a test from newly created not secure nabendu.blog site.
from contactForm, at 8:49 PM

This is new test from deployed site
from contactForm, on Nov 13

This is just a test
from contactForm, on Sep 28

Active Split Test

Split traffic to multiple branches

Split the traffic to your main domain between two or more branches, with no performance loss. Set up easy A/B tests with one click, leveraging edge node logic requests.

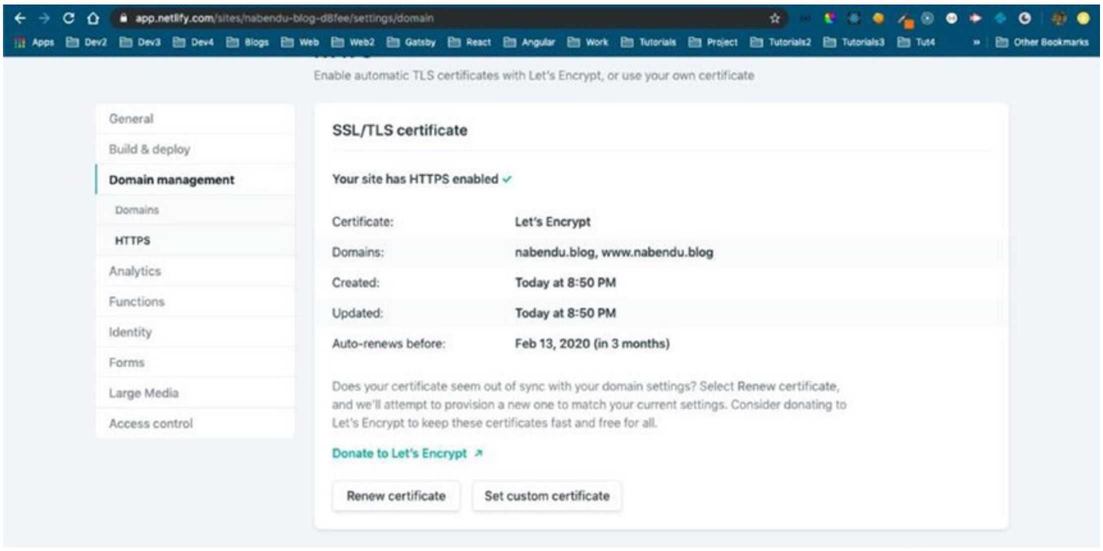
Start a Split Test →

Docs Pricing Support News Terms

© 2019 Netlify

Figure 2-35. *Contact form message*

I checked my HTTPS and it was activated in fewer than 30 minutes, as shown in Figure 2-36.



Enable automatic TLS certificates with Let's Encrypt, or use your own certificate

SSL/TLS certificate

Your site has HTTPS enabled ✓

Certificate:	Let's Encrypt
Domains:	nabendu.blog, www.nabendu.blog
Created:	Today at 8:50 PM
Updated:	Today at 8:50 PM
Auto-renews before:	Feb 13, 2020 (in 3 months)

Does your certificate seem out of sync with your domain settings? Select Renew certificate, and we'll attempt to provision a new one to match your current settings. Consider donating to Let's Encrypt to keep these certificates fast and free for all.

Figure 2-36. *The HTTPS site*

Now, if we head over to `https://nabendu.blog/`,[14] we won't get any errors. Finally my blog site is live with HTTPS, as shown in Figure 2-37.

[14]`https://nabendu.blog/`

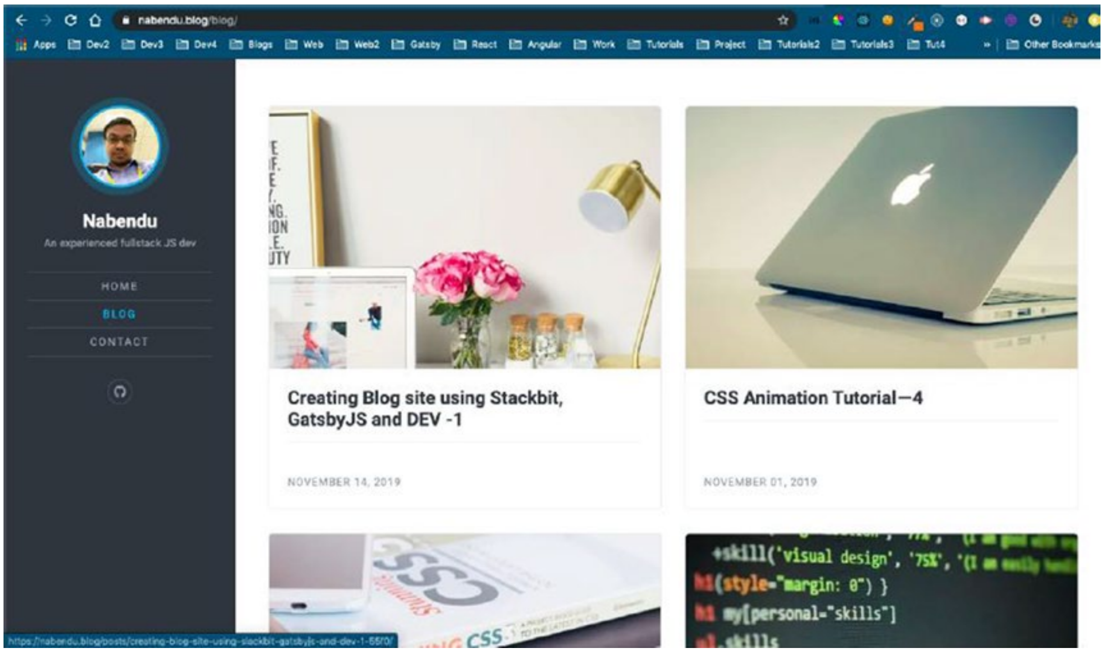


Figure 2-37. *Secure blog site, with HTTPS*

Adding Gatsby Plugins

In this section, we are going to add Gatsby plugins to our site, which are an easy way to add functionality to a Gatsby site. We are also going to add a plugin for Google analytics, for which you need a Google account.

We will start by adding Gatsby plugins to my newly created blog site[15]. I found a great article by Emeruche Cole[16] about this process.

So, let's head over to the terminal and run the command to get the latest data from Stackbit on our local machine. I got the following information from my `github repo`[17] for my blog site, which was initially created by Stackbit.

```
                npx @stackbit/stackbit-pull --stackbit-pull-api-
url=https://api.stackbit.com/pull/5d8d8fee6476dc00105e91ac
```

I will be adding two SEO plugins first.

15 `https://nabendu.blog/`
16 `https://dev.to/cole_ruche/my-top-plugins-for-a-gatsbyjs-powered-blog-1oo1`
17 `https://github.com/nabendu82/nabendu-blog`

gatsby-plugin-robots-txt

So, let's head over to the plugin page[18]. As the name suggests, it creates a `robots.txt` file for the Gatsby site. It's good for SEO as it tells Google bots on how to crawl your site.

We will follow the instructions and run an `npm install` in our directory, as shown here:

```
npm install --save gatsby-plugin-robots-txt
```

In the `gatsby-config.js` file, we need to add the code shown in Listing 2-2 to our plugin array.

Listing 2-2. gatsby-config.js

```
        {
            resolve: `gatsby-plugin-robots-txt`,
            options: {
                host: 'https://nabendu.blog/',
                sitemap: 'https://nabendu.blog/sitemap.xml',
                policy: [{ userAgent: '*', allow: '/' }]
            }
        }
```

gatsby-plugin-sitemap

This plugin generates a sitemap for your site, which is very important for SEO purposes. As per the instructions on the plugin page[19], let's run `npm install`:

```
npm install --save gatsby-plugin-sitemap
```

Now, this plugin is easy to use. We need to add the plugin name to the `gatsby-config.js` file inside the plugins array:

```
`gatsby-plugin-sitemap`
```

[18]https://www.gatsbyjs.org/packages/gatsby-plugin-robots-txt/?=gatsby-plugin-robots-txt

[19]https://www.gatsbyjs.org/packages/gatsby-plugin-sitemap/?=

We also need to have a `siteUrl` in our `siteMetaData`. But Stackbit creates a JSON for us. So, we need to add it there. We will also add some other useful keywords (see Listing 2-3) for SEO to the `site-metadata.json` file.

I found a helpful list at the Emeruche Cole GitHub link.[20]

Listing 2-3. The site-metadata.json File

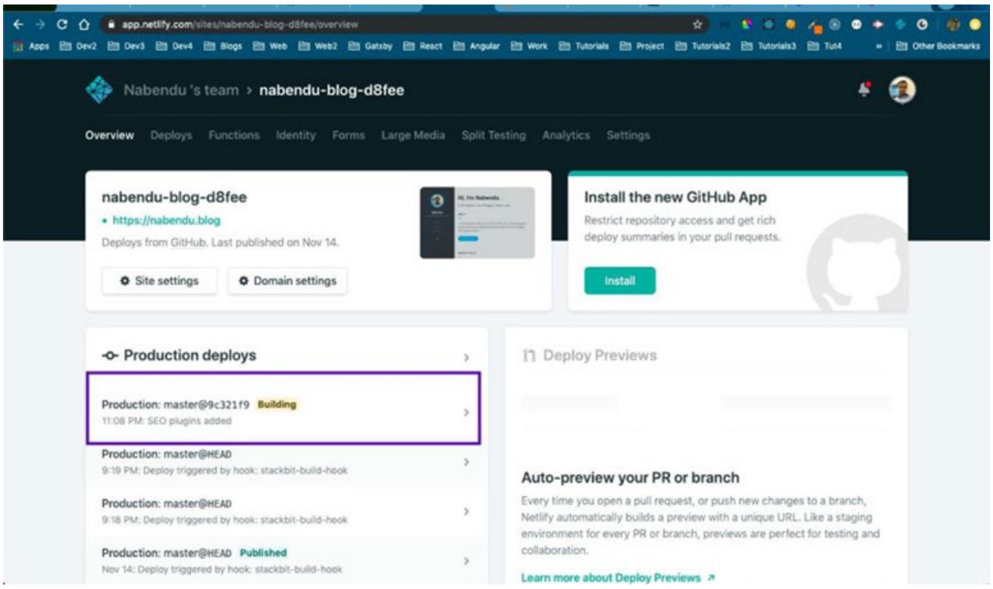
```
"title": "Nabendu's Blog",
"author": "Nabendu Biswas",
"siteUrl": "https://nabendu.blog/",
"description": "Blog posts started by Nabendu Biswas to share stuff I
learned in my Web development journey",
"keywords": [
    "Nabendu",
    "Biswas",
    "UI Lead",
    "GatsbyJs",
    "HTML",
    "CSS",
    "JavaScript",
    "ReactJs",
    "React developer",
    "Front-end Engineer"
]
```

Next, let's start our server to see if there are any errors. Start with `npm run develop`.

I have also checked `http://localhost:8000/`[21] and there were no errors. Next, let's add these changes to our production. With Netlify, it's simply a matter of pushing the changes to GitHub. So, I just committed my changes and pushed them to GitHub with `git push origin master`. When I head over to my Netlify dashboard, I see that it's building, as in Figure 2-38.

[20] https://github.com/kingingcole/myblog/blob/master/gatsby-config.js

[21] http://localhost:8000/



nabendu-blog-d8fee
https://nabendu.blog
Deploys from GitHub. Last published on Nov 14.

Install the new GitHub App
Restrict repository access and get rich deploy summaries in your pull requests.

Production: master@9c321f9 Building
11:08 PM: SEO plugins added

Production: master@HEAD
9:19 PM: Deploy triggered by hook: stackbit-build-hook

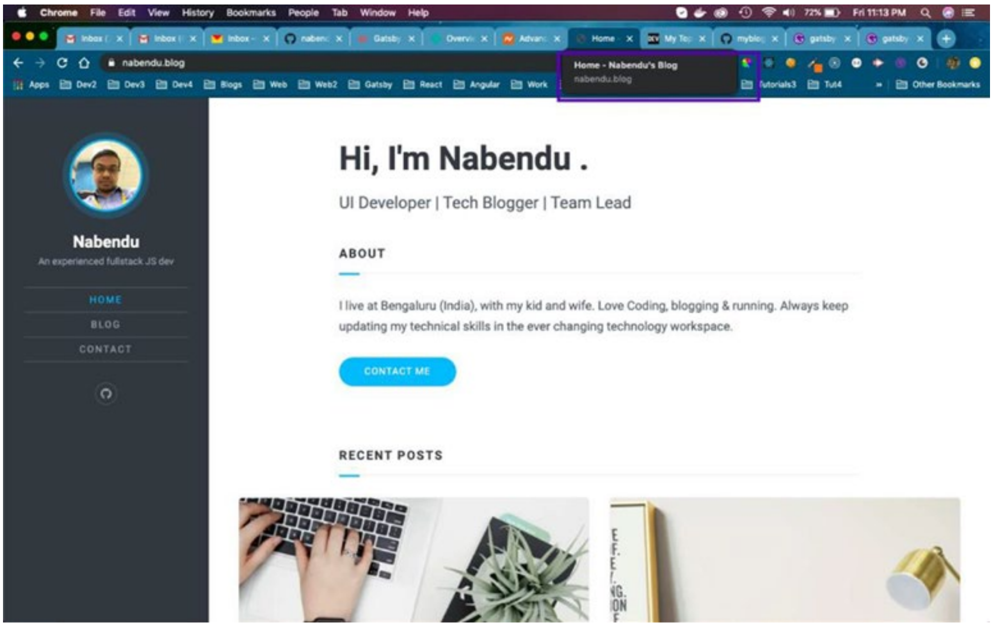
Production: master@HEAD
9:18 PM: Deploy triggered by hook: stackbit-build-hook

Production: master@HEAD Published
Nov 14: Deploy triggered by hook: stackbit-build-hook

Auto-preview your PR or branch
Every time you open a pull request, or push new changes to a branch, Netlify automatically builds a preview with a unique URL. Like a staging environment for every PR or branch, previews are perfect for testing and collaboration.

Figure 2-38. *Building*

It was published in less than five minutes. Now, when I head over to my site, my title change is there, as shown in Figure 2-39. Everything else is behind the scenes.



Hi, I'm Nabendu .

UI Developer | Tech Blogger | Team Lead

ABOUT

I live at Bengaluru (India), with my kid and wife. Love Coding, blogging & running. Always keep updating my technical skills in the ever changing technology workspace.

RECENT POSTS

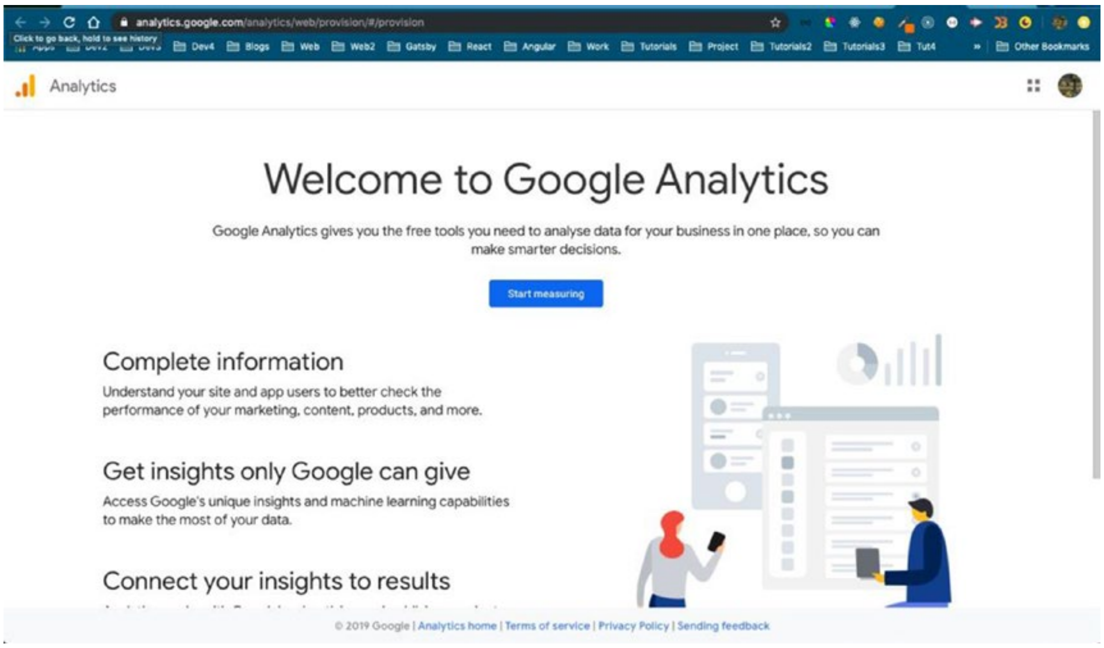
Figure 2-39. *The title has changed*

Next, we will add one of the most important plugins, one for analytics purposes.

gatsby-plugin-google-analytics

This plugin will add Google Analytics to our website. As per the Gatsby docs[22] on setting up Google Analytics, we have to first set up a Google Analytics account and then get the tracking ID.

Let's open Google Analytics[23]. If you are not logged in, log in to your Google account. It will show the screen in Figure 2-40.


analytics.google.com/analytics/web/provision/#/provision
Analytics
Start measuring
© 2019 Google | Analytics home | Terms of service | Privacy Policy | Sending feedback

Welcome to Google Analytics

Google Analytics gives you the free tools you need to analyse data for your business in one place, so you can make smarter decisions.

Complete information

Understand your site and app users to better check the performance of your marketing, content, products, and more.

Get insights only Google can give

Access Google's unique insights and machine learning capabilities to make the most of your data.

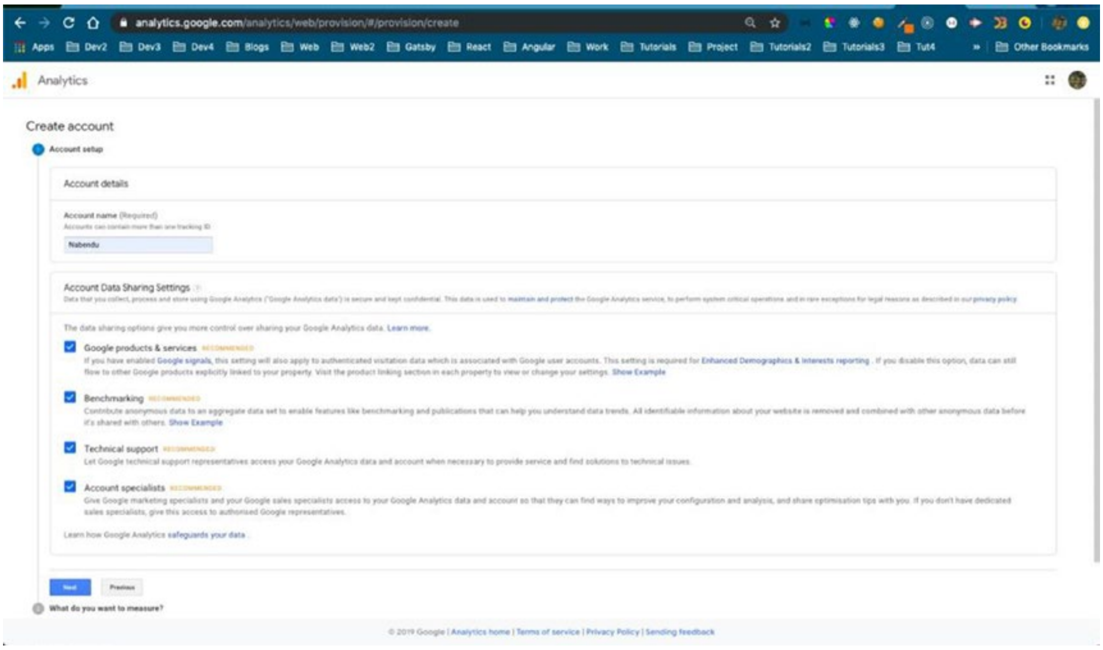
Connect your insights to results

Figure 2-40. *The Google Analytics Welcome page*

Once you click the Start Measuring button, it will take you to the page shown in Figure 2-41. You will be asked for your account name. Click the Next button when you're done.

[22] https://www.gatsbyjs.org/docs/adding-analytics/

[23] https://analytics.google.com/analytics/web/provision/#/provision



analytics.google.com/analytics/web/provision/#/provision/create

Analytics

Create account

Account setup

Account details

Account name (Required)
Accounts can contain more than one tracking ID
Nabendu

Account Data Sharing Settings
Data that you collect, process and store using Google Analytics ("Google Analytics data") is secure and kept confidential. This data is used to maintain and protect the Google Analytics service, to perform system critical operations and in rare exceptions for legal reasons as described in our privacy policy.

The data sharing options give you more control over sharing your Google Analytics data. Learn more.

☑ Google products & services RECOMMENDED
If you have enabled Google signals, this setting will also apply to authenticated visitation data which is associated with Google user accounts. This setting is required for Enhanced Demographics & Interests reporting. If you disable this option, data can still flow to other Google products explicitly linked to your property. Visit the product linking section in each property to view or change your settings. Show Example

☑ Benchmarking RECOMMENDED
Contribute anonymous data to an aggregate data set to enable features like benchmarking and publications that can help you understand data trends. All identifiable information about your website is removed and combined with other anonymous data before it's shared with others. Show Example

☑ Technical support RECOMMENDED
Let Google technical support representatives access your Google Analytics data and account when necessary to provide service and find solutions to technical issues.

☑ Account specialists RECOMMENDED
Give Google marketing specialists and your Google sales specialists access to your Google Analytics data and account so that they can find ways to improve your configuration and analysis, and share optimisation tips with you. If you don't have dedicated sales specialists, give this access to authorised Google representatives.

Learn how Google Analytics safeguards your data.

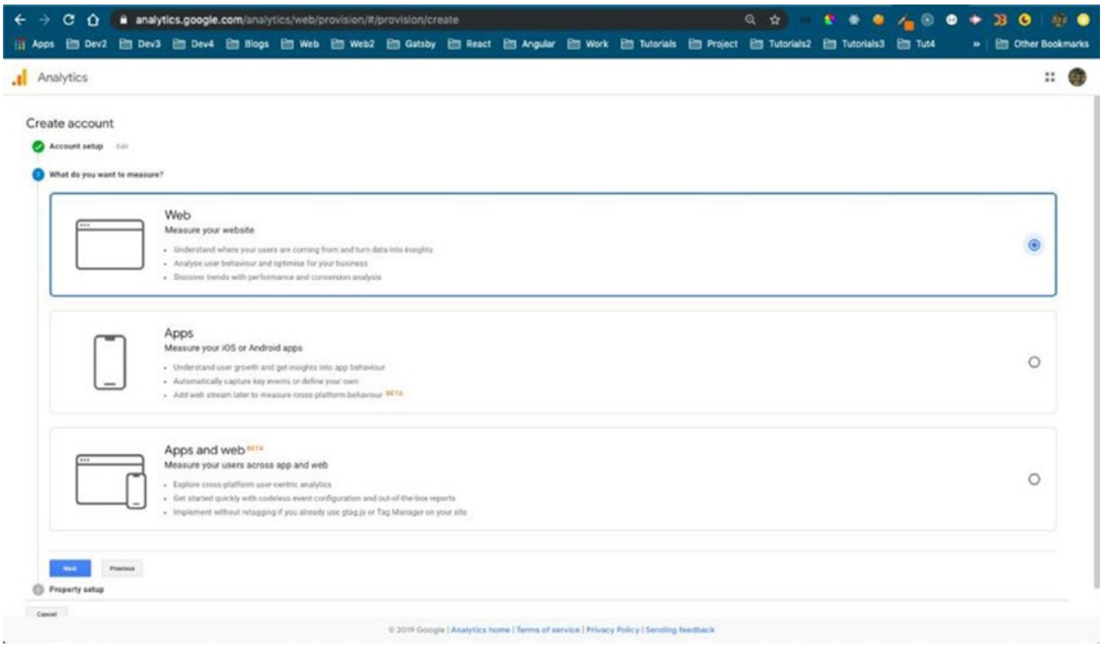
Next Previous

What do you want to measure?

© 2019 Google | Analytics home | Terms of service | Privacy Policy | Sending feedback

Figure 2-41. *Account name*

In the screen in Figure 2-42, select Web and then click Next.



analytics.google.com/analytics/web/provision/#/provision/create

Analytics

Create account

Account setup Edit

What do you want to measure?

[✓ Web]
Measure your website
- Understand where your users are coming from and turn data into insights
- Analyse user behaviour and optimise for your business
- Discover trends with performance and conversion analysis

Apps
Measure your iOS or Android apps
- Understand user growth and get insights into app behaviour
- Automatically capture key events or define your own
- Add web stream later to measure cross-platform behaviour BETA

Apps and web BETA
Measure your users across app and web
- Explore cross-platform user-centric analytics
- Get started quickly with codeless event configuration and out-of-the-box reports
- Implement without retagging if you already use gtag.js or Tag Manager on your site

Next Previous

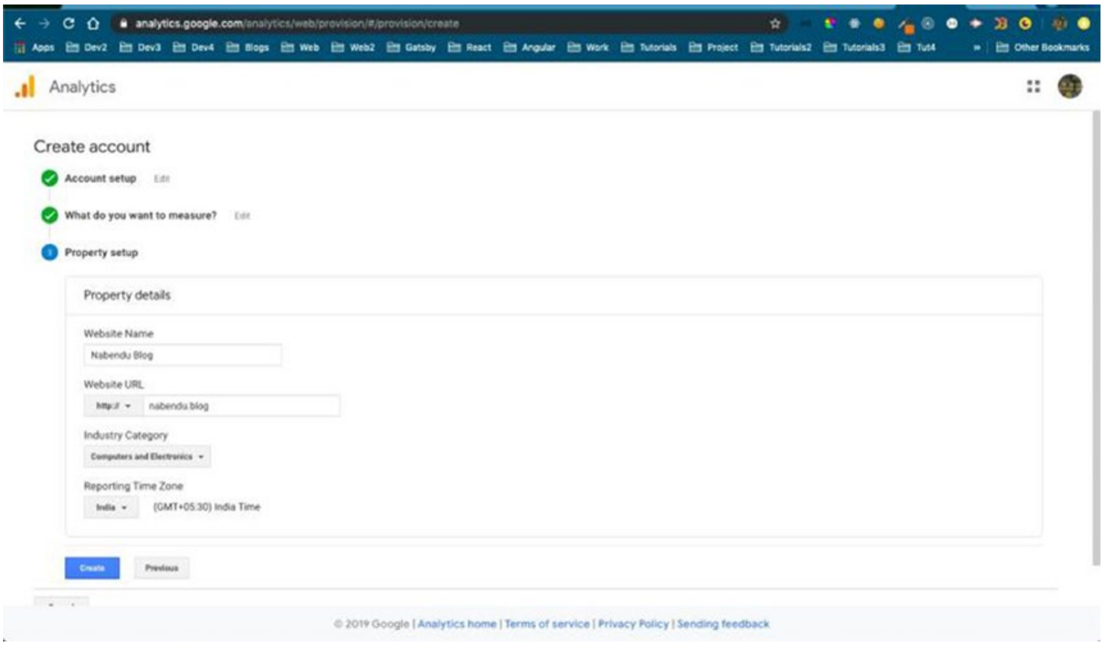
Property setup

Cancel

© 2019 Google | Analytics home | Terms of service | Privacy Policy | Sending feedback

Figure 2-42. *Choose Web here*

In the screen in Figure 2-43, you have to provide some basic site information, including the site's URL.


analytics.google.com/analytics/web/provision/#/provision/create
Analytics

Create account

Account setup Edit

What do you want to measure? Edit

Property setup

Property details

Website Name: Nabendu Blog

Website URL: http:// nabendu.blog

Industry Category: Computers and Electronics

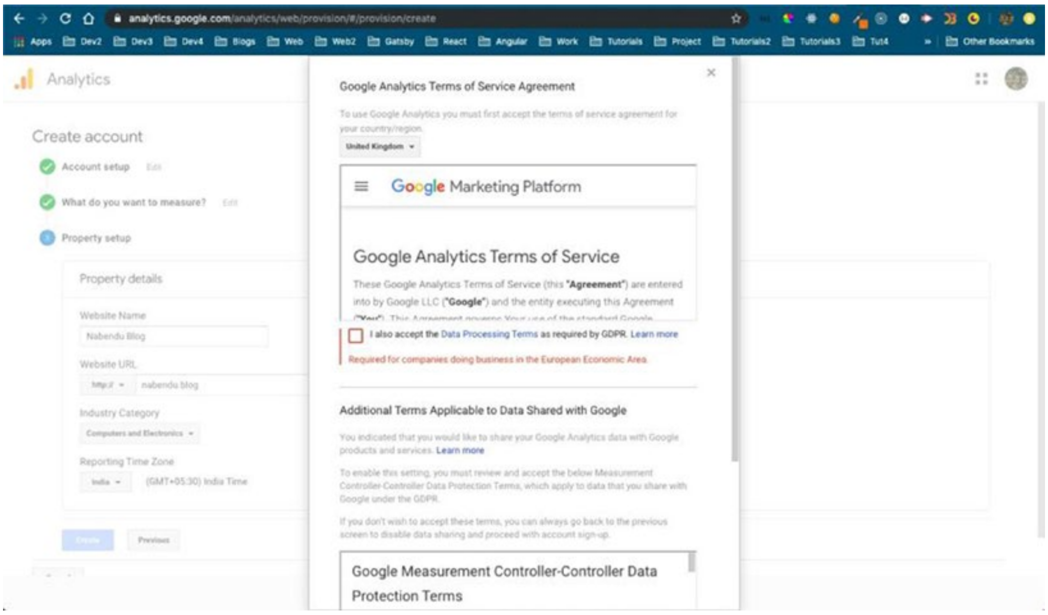
Reporting Time Zone: India (GMT+05:30) India Time

Create Previous

© 2019 Google | Analytics home | Terms of service | Privacy Policy | Sending feedback

Figure 2-43. *Basic site information*

Once you click the Create button, a popup will appear, as shown in Figure 2-44.


Google Analytics Terms of Service Agreement

To use Google Analytics you must first accept the terms of service agreement for your country/region.

United Kingdom

Google Analytics Terms of Service

These Google Analytics Terms of Service (this **"Agreement"**) are entered into by Google LLC (**"Google"**) and the entity executing this Agreement

I also accept the Data Processing Terms as required by GDPR. Learn more

Required for companies doing business in the European Economic Area

Additional Terms Applicable to Data Shared with Google

You indicated that you would like to share your Google Analytics data with Google products and services. Learn more

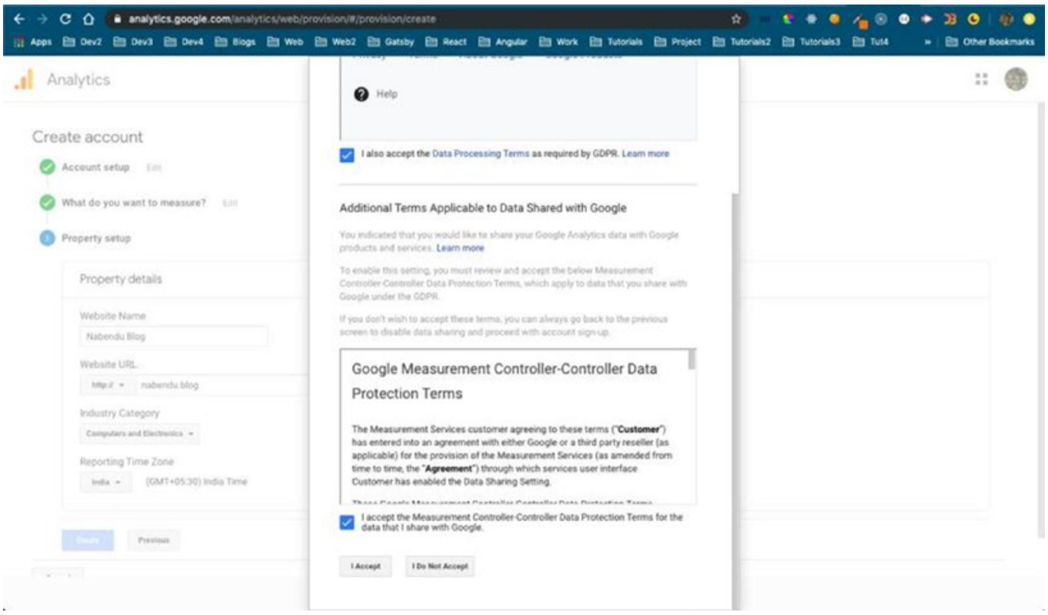
To enable this setting, you must review and accept the below Measurement Controller-Controller Data Protection Terms, which apply to data that you share with Google under the GDPR.

If you don't wish to accept these terms, you can always go back to the previous screen to disable data sharing and proceed with account sign-up.

Google Measurement Controller-Controller Data Protection Terms

Figure 2-44. *The popup asks you to accept the terms*

Here, you have to accept the terms and conditions. Click the Accept buttons, as shown in Figure 2-45.


Help

☑ I also accept the Data Processing Terms as required by GDPR. Learn more

Additional Terms Applicable to Data Shared with Google

You indicated that you would like to share your Google Analytics data with Google products and services. Learn more

To enable this setting, you must review and accept the below Measurement Controller-Controller Data Protection Terms, which apply to data that you share with Google under the GDPR.

If you don't wish to accept these terms, you can always go back to the previous screen to disable data sharing and proceed with account sign-up.

Google Measurement Controller-Controller Data Protection Terms

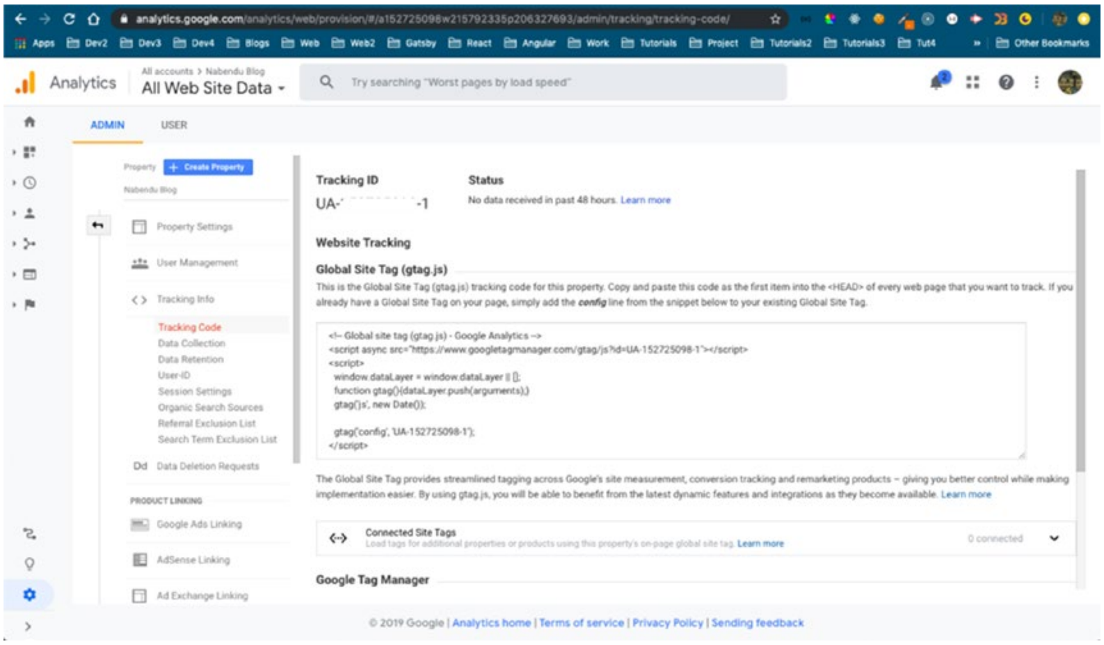
The Measurement Services customer agreeing to these terms (**"Customer"**) has entered into an agreement with either Google or a third party reseller (as applicable) for the provision of the Measurement Services (as amended from time to time, the **"Agreement"**) through which services user interface Customer has enabled the Data Sharing Setting.

☑ I accept the Measurement Controller-Controller Data Protection Terms for the data that I share with Google.

I Accept | I Do Not Accept

Figure 2-45. *You have to accept the terms*

The next screen, shown in Figure 2-46, will contain your tracking ID. It is in the format of UA-XXXXXXXXX-X.



Tracking ID
UA- -1

Status
No data received in past 48 hours. Learn more

Website Tracking

Global Site Tag (gtag.js)

This is the Global Site Tag (gtag.js) tracking code for this property. Copy and paste this code as the first item into the <HEAD> of every web page that you want to track. If you already have a Global Site Tag on your page, simply add the ***config*** line from the snippet below to your existing Global Site Tag.

```
<!-- Global site tag (gtag.js) - Google Analytics -->
<script async src="https://www.googletagmanager.com/gtag/js?id=UA-152725098-1"></script>
<script>
  window.dataLayer = window.dataLayer || [];
  function gtag(){dataLayer.push(arguments);}
  gtag('js', new Date());

  gtag('config', 'UA-152725098-1');
</script>
```

The Global Site Tag provides streamlined tagging across Google's site measurement, conversion tracking and remarketing products – giving you better control while making implementation easier. By using gtag.js, you will be able to benefit from the latest dynamic features and integrations as they become available. Learn more

Connected Site Tags
Load tags for additional properties or products using this property's on-page global site tag. Learn more
0 connected

Google Tag Manager

© 2019 Google | Analytics home | Terms of service | Privacy Policy | Sending feedback

Figure 2-46. *Tracking ID*

Now it's time to install the Gatsby Google Analytics plugin by executing `npm install --save gatsby-plugin-google-analytics` in the terminal.

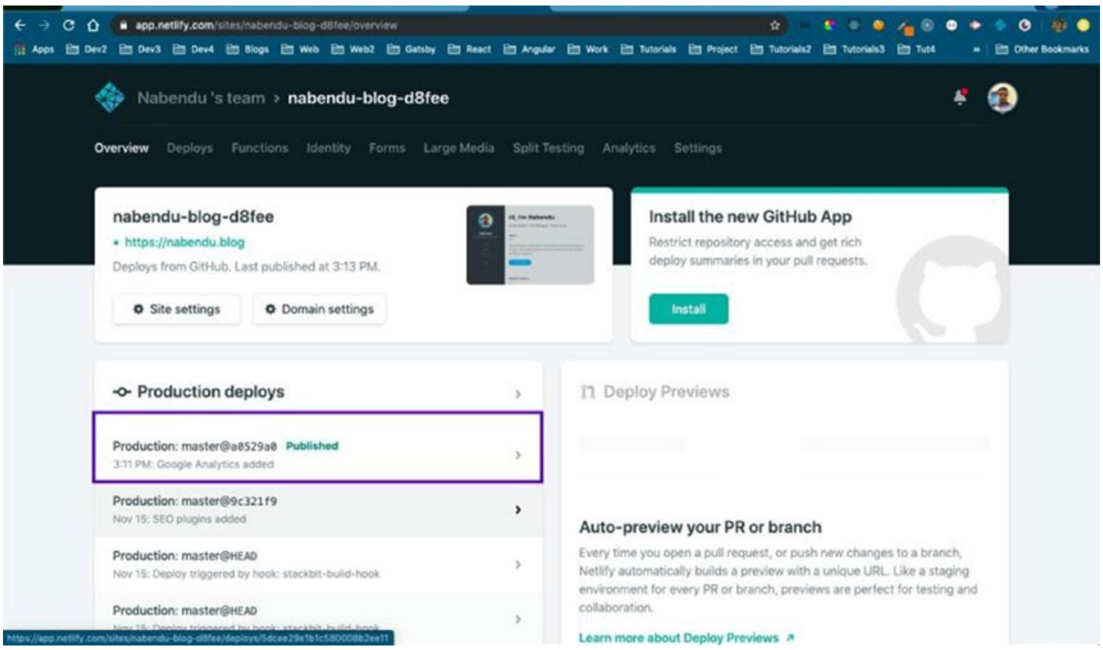
Next, we need to add the plugin to the `gatsby-config.js` file (see Listing 2-4). You have to enter the Tracking ID you got from Google.

Listing 2-4. gatsby-config.js

```
{
    resolve: `gatsby-plugin-google-analytics`,
    options: {
        trackingId: "UA-XXXXXXXXX-1",
    }
}
```

Next, let's start the DEV server to see if there are any issues. We need to restart the server by running the `npm run develop` command on the terminal. I also checked `http://localhost:8000/`[24] again and there were no errors. Next, let's add these changes to our production.

If you commit your changes and push them to GitHub, it will build the production site in Netlify again, as shown in Figure 2-47.



nabendu-blog-d8fee
https://nabendu.blog
Deploys from GitHub. Last published at 3:13 PM.

Restrict repository access and get rich deploy summaries in your pull requests.

Production: master@a0529a0 Published
3:11 PM: Google Analytics added

Production: master@9c321f9
Nov 15: SEO plugins added

Production: master@HEAD
Nov 15: Deploy triggered by hook: stackbit-build-hook

Production: master@HEAD
Nov 15: Deploy triggered by hook: stackbit-build-hook

Every time you open a pull request, or push new changes to a branch, Netlify automatically builds a preview with a unique URL. Like a staging environment for every PR or branch, previews are perfect for testing and collaboration.

Figure 2-47. Changes have been published

Once the site is published, go back to the Google Analytics page and click the Home tab. You will see your site data, as shown in Figure 2-48.

[24] `http://localhost:8000/`

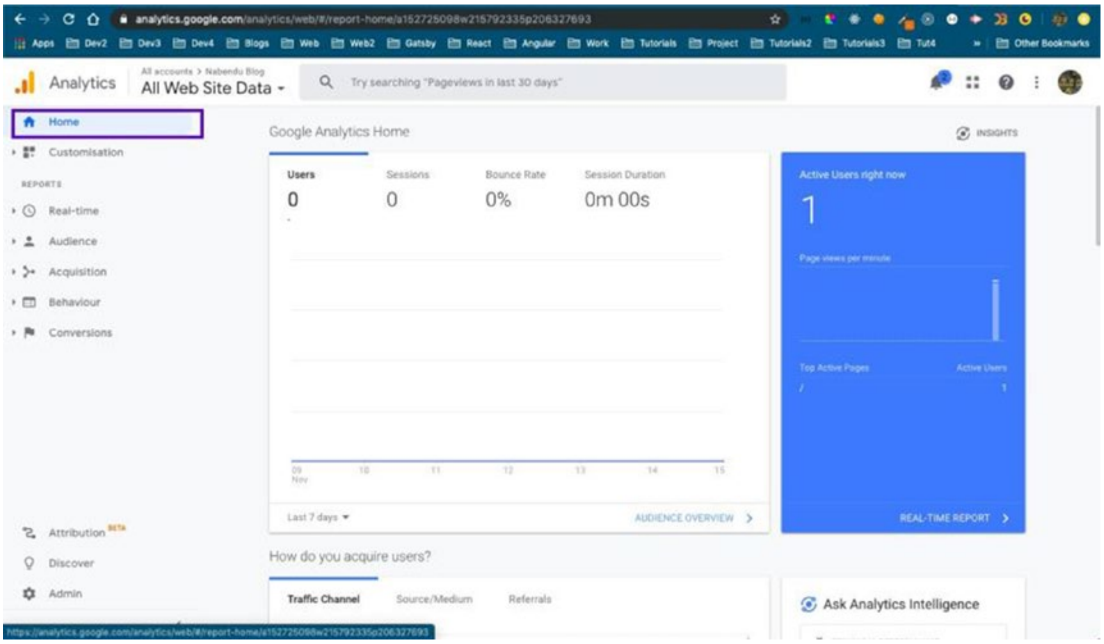


Figure 2-48. *Site data*

One more important thing I want to do to my personal blog site is to change the canonical link in my Medium and DEV posts.

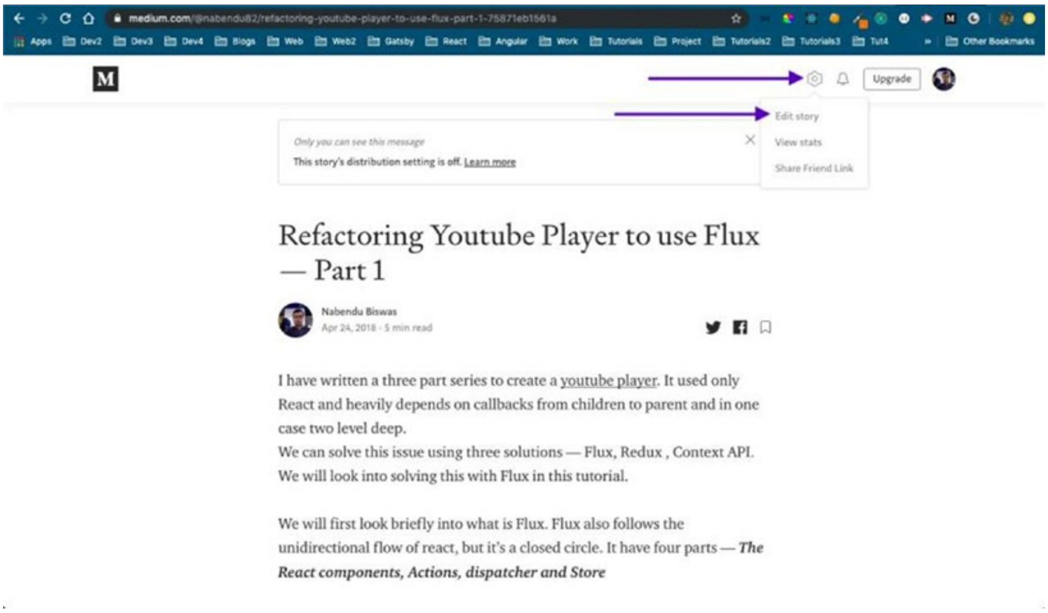
If a post occurs in more than one place, the Google bots give greater priority to the original post. Now, it doesn't matter where you published the blog first, because all big blogging platforms have ways to give canonical links.

My process is to create the blog in Medium first because that is what I have been doing for several years. After that, I use an awesome package called Medium Exporter[25] to change the post to a Markdown file.

After that, I had to paste the contents of the Markdown file to a new DEV post. Since Dev is also serving as my headless CMS for my site, due to the awesome service of Stackbit, it is published on my personal site.

I will show the process on one of my earlier blogs. First, we have to add a link to a Medium post. I opened one of my earlier Medium posts. Since I am logged in, I also get the Edit button. I first click the Settings icon and then choose the Edit Story option, as shown in Figure 2-49.

[25] https://medium.com/@macropus/export-your-medium-posts-to-markdown-b5ccc8cb0050



Only you can see this message

This story's distribution setting is off. Learn more

Refactoring Youtube Player to use Flux — Part 1

Nabendu Biswas
Apr 24, 2018 · 5 min read

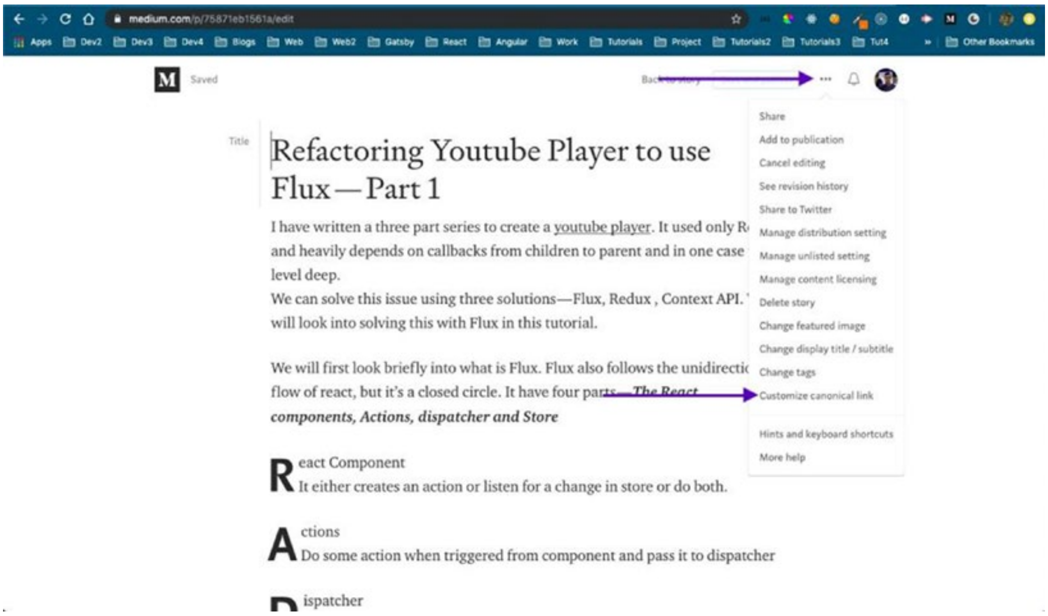
I have written a three part series to create a youtube player. It used only React and heavily depends on callbacks from children to parent and in one case two level deep.
We can solve this issue using three solutions — Flux, Redux , Context API.
We will look into solving this with Flux in this tutorial.

We will first look briefly into what is Flux. Flux also follows the unidirectional flow of react, but it's a closed circle. It have four parts — ***The React components, Actions, dispatcher and Store***

medium.com/@nabendu82/refactoring-youtube-player-to-use-flux-part-1-75871eb1561a
Upgrade
Edit story
View stats
Share Friend Link

Figure 2-49. *Editing Medium*

After that, I click the three dots and then choose the Customize Canonical Link, as shown in Figure 2-50.



Refactoring Youtube Player to use Flux — Part 1

I have written a three part series to create a youtube player. It used only R… and heavily depends on callbacks from children to parent and in one case level deep.
We can solve this issue using three solutions—Flux, Redux , Context API. … will look into solving this with Flux in this tutorial.

We will first look briefly into what is Flux. Flux also follows the unidirecti… flow of react, but it's a closed circle. It have four parts—***The React components, Actions, dispatcher and Store***

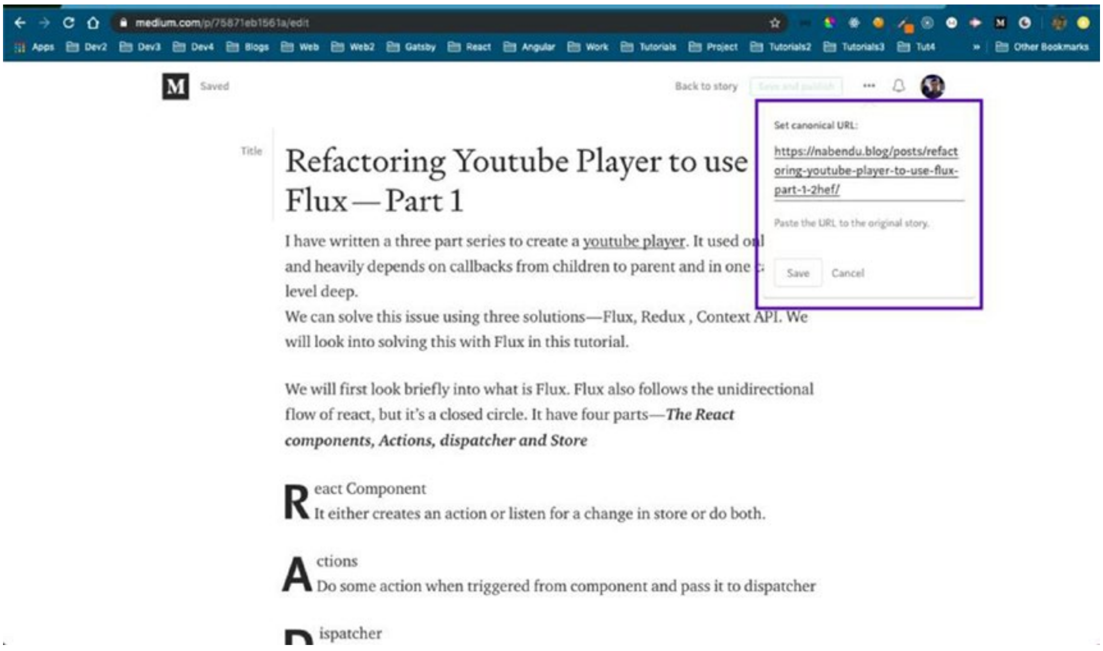
React Component
It either creates an action or listen for a change in store or do both.

Actions
Do some action when triggered from component and pass it to dispatcher

Dispatcher

Figure 2-50. *More editing*

After that, I need to paste the original link into the Set Canonical URL: field. Here I gave the link from my blog site. After that, I click the Save button. See Figure 2-51.



Set canonical URL:
https://nabendu.blog/posts/refactoring-youtube-player-to-use-flux-part-1-2hef/
Paste the URL to the original story.

Refactoring Youtube Player to use Flux — Part 1

I have written a three part series to create a youtube player. It used o
and heavily depends on callbacks from children to parent and in one
level deep.
We can solve this issue using three solutions—Flux, Redux , Context API. We will look into solving this with Flux in this tutorial.

We will first look briefly into what is Flux. Flux also follows the unidirectional flow of react, but it's a closed circle. It have four parts—***The React components, Actions, dispatcher and Store***

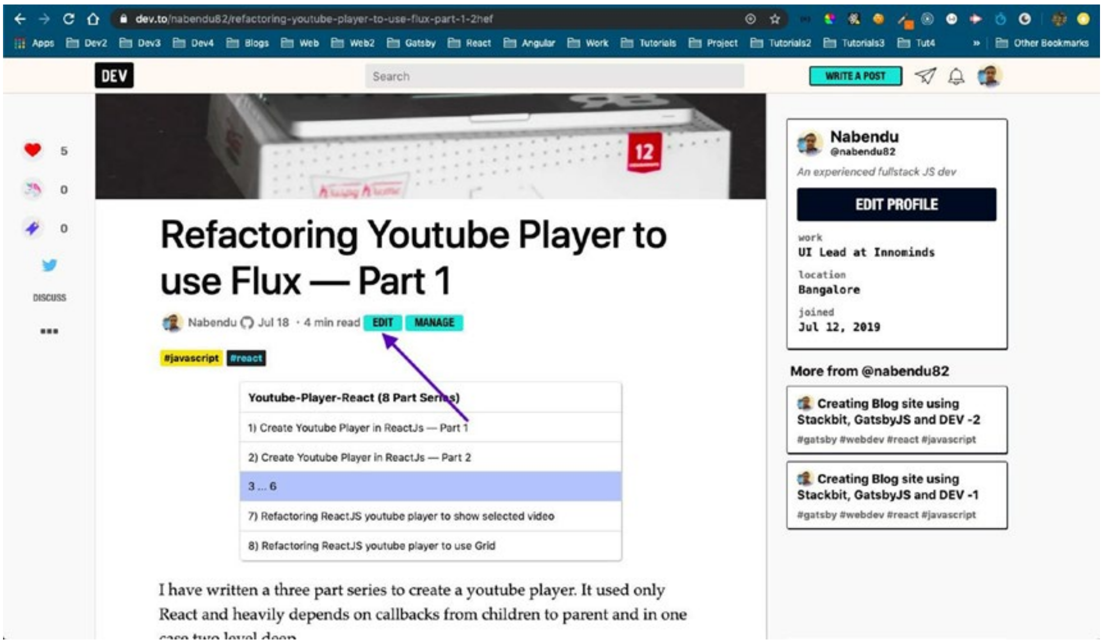
React Component
It either creates an action or listen for a change in store or do both.

Actions
Do some action when triggered from component and pass it to dispatcher

Dispatcher

Figure 2-51. *Setting the canonical URL*

Next it's time to head over to `dev.to` and change the canonical link there also. When I head over to my same blog in DEV, I get the Edit button, as shown in Figure 2-52.


dev.to/nabendu82/refactoring-youtube-player-to-use-flux-part-1-2hef
DEV
Search
WRITE A POST
DISCUSS
EDIT
MANAGE
EDIT PROFILE

Refactoring Youtube Player to use Flux — Part 1

Nabendu Jul 18 · 4 min read

#javascript #react

Youtube-Player-React (8 Part Series)

1) Create Youtube Player in ReactJs — Part 1

2) Create Youtube Player in ReactJs — Part 2

3 ... 6

7) Refactoring ReactJS youtube player to show selected video

8) Refactoring ReactJS youtube player to use Grid

I have written a three part series to create a youtube player. It used only React and heavily depends on callbacks from children to parent and in one case two level deep.

Nabendu
@nabendu82
An experienced fullstack JS dev

work
UI Lead at Innominds
location
Bangalore
joined
Jul 12, 2019

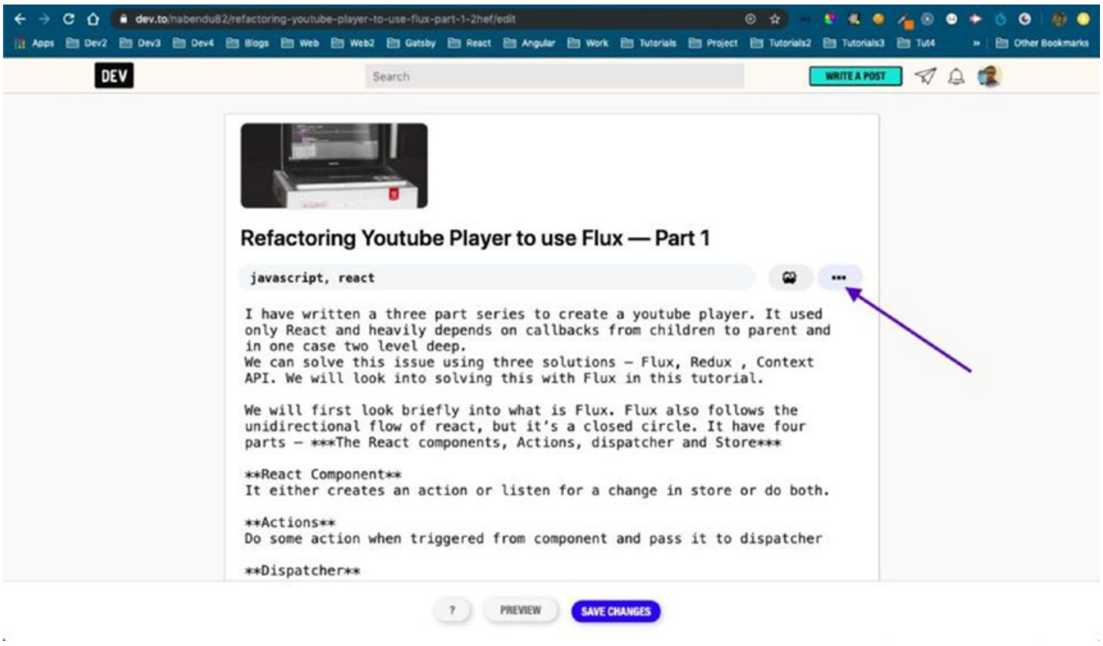
More from @nabendu82

Creating Blog site using Stackbit, GatsbyJS and DEV -2
#gatsby #webdev #react #javascript

Creating Blog site using Stackbit, GatsbyJS and DEV -1
#gatsby #webdev #react #javascript

Figure 2-52. *DEV editing*

Next, I click the three dots, as shown in Figure 2-53.


dev.to/nabendu82/refactoring-youtube-player-to-use-flux-part-1-2hef/edit
DEV
Search
WRITE A POST
?
PREVIEW
SAVE CHANGES

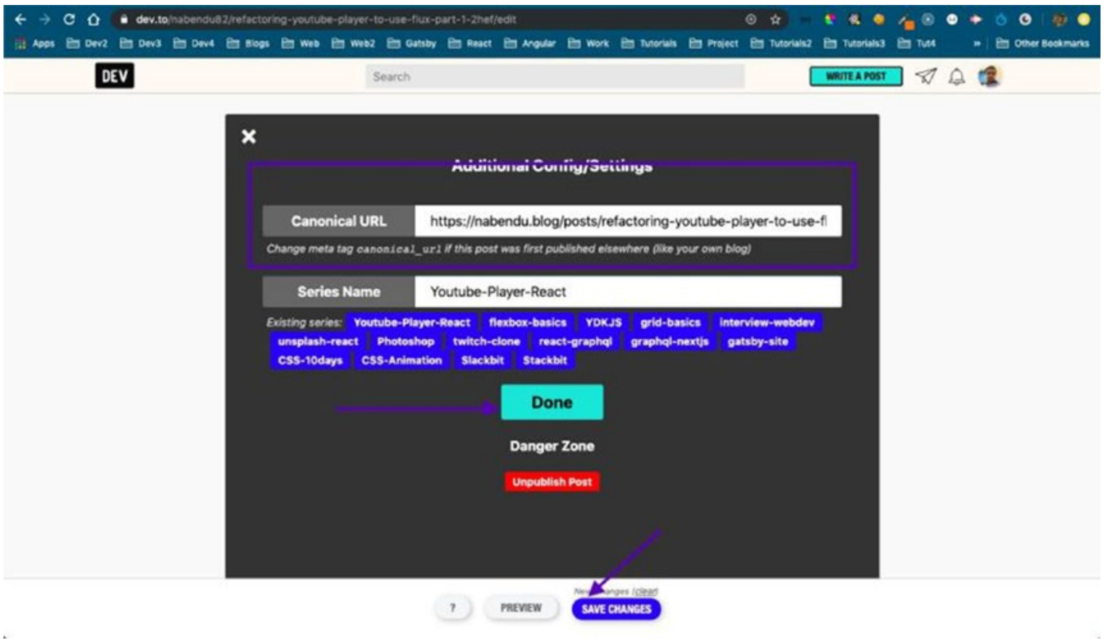
Refactoring Youtube Player to use Flux — Part 1

javascript, react

```
I have written a three part series to create a youtube player. It used
only React and heavily depends on callbacks from children to parent and
in one case two level deep.
We can solve this issue using three solutions — Flux, Redux , Context
API. We will look into solving this with Flux in this tutorial.

We will first look briefly into what is Flux. Flux also follows the
unidirectional flow of react, but it's a closed circle. It have four
parts — ***The React components, Actions, dispatcher and Store***

**React Component**
It either creates an action or listen for a change in store or do both.

**Actions**
Do some action when triggered from component and pass it to dispatcher

**Dispatcher**
```

Figure 2-53. *Click the three dots*

It will open a popup, where we have to give the canonical link. I again give it the link from my blog site in Canonical URL. After that, click the Done button, which will close this popup. Click the Save Changes button next. This is shown in Figure 2-54.


Additional Config/Settings

Canonical URL: https://nabendu.blog/posts/refactoring-youtube-player-to-use-fl

Change meta tag `canonical_url` if this post was first published elsewhere (like your own blog)

Series Name: Youtube-Player-React

Existing series: Youtube-Player-React, flexbox-basics, YDKJS, grid-basics, interview-webdev, unsplash-react, Photoshop, twitch-clone, react-graphql, graphql-nextjs, gatsby-site, CSS-10days, CSS-Animation, Slackbit, Stackbit

dev.to/nabendu82/refactoring-youtube-player-to-use-flux-part-1-2hef/edit
DEV
Search
WRITE A POST
Done
Danger Zone
Unpublish Post
?
PREVIEW
SAVE CHANGES

Figure 2-54. *Canonical URL*

Now I have to repeat the process of changing the canonical links of my 200 blogs manually for my site[26] to get better SEO results. As you might know, I built my personal site with Stackbit, which publishes all my `dev.to`[27] blogs to `nabendu.blog`[28].

It's an amazing service and I created the site in less than an hour. But, as with most things that automate processes, one of the major drawbacks is that my posts are exact replicas of the `dev.to` posts and cannot be edited.

I need to make some minor changes to the site, which I will discuss further and then will try to do them. It is not possible for me to change from Stackbit now, as my site gets 1,000 visitors per month. I also manually updated the site canonical links (130 blogs to date) to Medium and DEV.

[26] `https://nabendu.blog/`
[27] `https://dev.to/nabendu82`
[28] `https://nabendu.blog/`

There are five things that I need to change on my blog:

- Add the `disqus` plugin to my blog, so that people can add comment and likes
- Add advertisement to the home page (not anywhere else, as I also don’t like more ads)
- Add more social links to the home page
- Change my picture on the home page
- Change the favicon of the site

I was searching for a way to add the `disqus` plugin in the code, but I found a way to add social links to the home page.

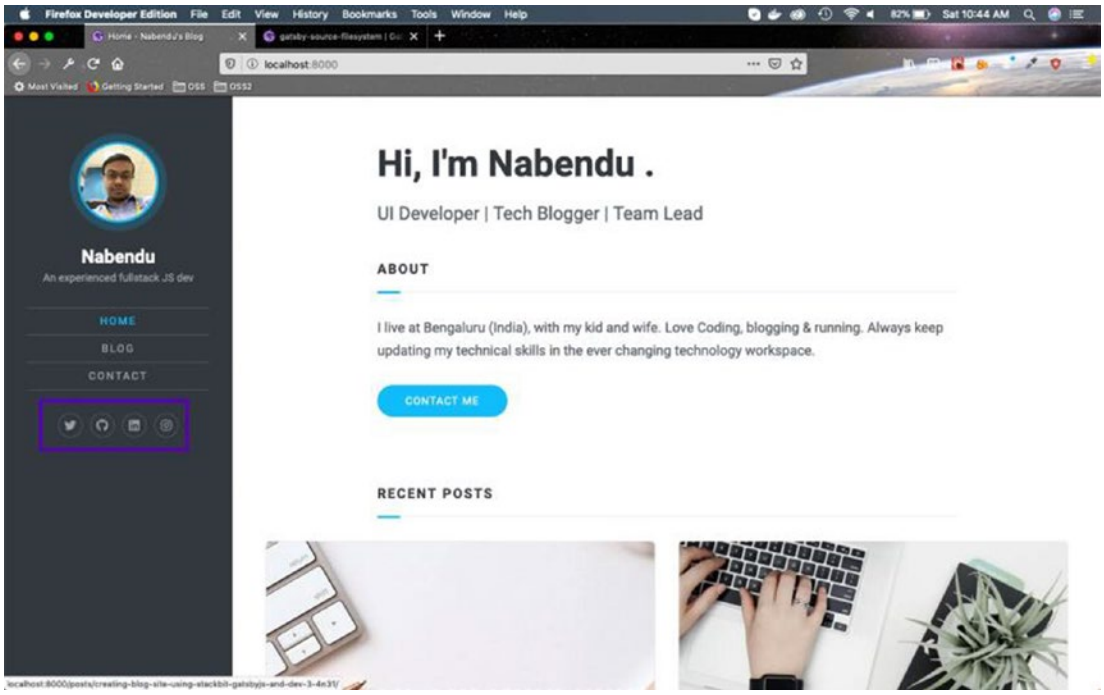
Adding Social Links

By default, we only have the GitHub social link on the home page. To add other social links, we need to add the code in Listing 2-5 to the `social.json` file.

Listing 2-5. The social.json File

```
{
    "links": [
        {
            "type": "twitter",
            "title": "Twitter",
            "icon": "fa-twitter",
            "url": "https://twitter.com/nabendu82"
        },
        {
            "type": "github",
            "title": "GitHub",
            "icon": "fa-github",
            "url": "https://github.com/nabendu82"
        },
        {
            "type": "linkedin",
            "title": "Linkedin",
            "icon": "fa-linkedin",
            "url": "https://www.linkedin.com/in/nabendu-biswas-42aa4522/"
        },
        {
            "type": "instagram",
            "title": "Instagram",
            "icon": "fa-instagram",
            "url": "https://www.instagram.com/nabendu82/"
        }
    ]
}
```

In my local development server, the changes were not reflected, so I had to restart the `gatsby develop`. After that, the social links were updated, as shown in Figure 2-55.



Firefox Developer Edition File Edit View History Bookmarks Tools Window Help
localhost:8000
Nabendu
An experienced fullstack JS dev
HOME
BLOG
CONTACT
Hi, I'm Nabendu .
UI Developer | Tech Blogger | Team Lead
ABOUT
CONTACT ME
RECENT POSTS

I live at Bengaluru (India), with my kid and wife. Love Coding, blogging & running. Always keep updating my technical skills in the ever changing technology workspace.

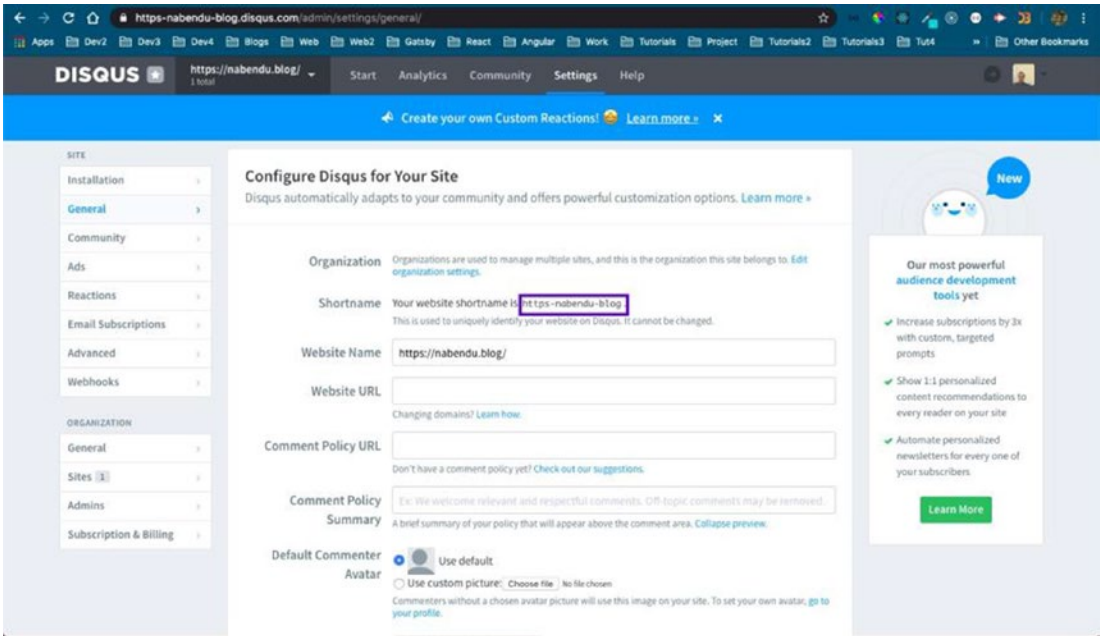
Figure 2-55. *The social links*

Adding the Disqus Plugin

After doing some research, I can install the `Disqus` plugin, as per the documentation[29] inside my `project` directory:

```
npm install -S gatsby-plugin-disqus
```

We need to register to `disqus` and get the website short name. You can get it from the screen shown in Figure 2-56.


https-nabendu-blog.disqus.com/admin/settings/general/
Apps Dev2 Dev3 Dev4 Blogs Web Web2 Gatsby React Angular Work Tutorials Project Tutorials2 Tutorials3 Tut4 Other Bookmarks
DISQUS
https://nabendu.blog/
1 total
Start Analytics Community Settings Help
Create your own Custom Reactions! Learn more »
SITE
Installation
General
Community
Ads
Reactions
Email Subscriptions
Advanced
Webhooks
ORGANIZATION
General
Sites 1
Admins
Subscription & Billing
New
Learn More
Choose file
No file chosen

Configure Disqus for Your Site

Disqus automatically adapts to your community and offers powerful customization options. Learn more »

Organization: Organizations are used to manage multiple sites, and this is the organization this site belongs to. Edit organization settings.

Shortname: Your website shortname is `https-nabendu-blog`. This is used to uniquely identify your website on Disqus. It cannot be changed.

Website Name: https://nabendu.blog/

Website URL: Changing domains? Learn how.

Comment Policy URL: Don't have a comment policy yet? Check out our suggestions.

Comment Policy Summary: Ex: We welcome relevant and respectful comments. Off-topic comments may be removed. A brief summary of your policy that will appear above the comment area. Collapse preview.

Default Commenter Avatar: [✓ Use default] Use custom picture: Commenters without a chosen avatar picture will use this image on your site. To set your own avatar, go to your profile.

Our most powerful audience development tools yet

- Increase subscriptions by 3x with custom, targeted prompts
- Show 1:1 personalized content recommendations to every reader on your site
- Automate personalized newsletters for every one of your subscribers

Figure 2-56. *The short name*

Next, it's time to add the content in Listing 2-6 to the `gatsby-config.js` file.

Listing 2-6. gatsby-config.js

```
{
    resolve: `gatsby-plugin-disqus`,
```

[29] https://www.gatsbyjs.org/packages/gatsby-plugin-disqus/

```
            options: {
                shortname: `https-nabendu-blog`
            }
        }
```

We need to add disqus to the post.js template. After some research and help from this GitHub[30] belonging to Cole Emeruche,[31]I was able to add disqus. We need to add the URL, the identifier, and the style to disqusConfig. To do this, we need a bit of JavaScript.

We need to add the bold code in post.js, as shown in Listing 2-7.

Listing 2-7. The post.js File

```
...
...
import {htmlToReact, safePrefix} from '../utils';
import { Disqus, CommentCount } from "gatsby-plugin-disqus";
export default class Post extends React.Component {
    render() {
      const url = "https://nabendu.blog";
      const siteTitle = _.get(this.props, 'pageContext.site.siteMetadata.
      title');
      const blogIdentity = this.props.location.pathname.split("/")[2];
      let disqusConfig = {
        url: `${url}${this.props.location.pathname}`,
        identifier: blogIdentity,
        title: siteTitle,
      }
        return (
                          ...
                           ...
                    );
    }
}
```

[30]https://github.com/kingingcole/myblog

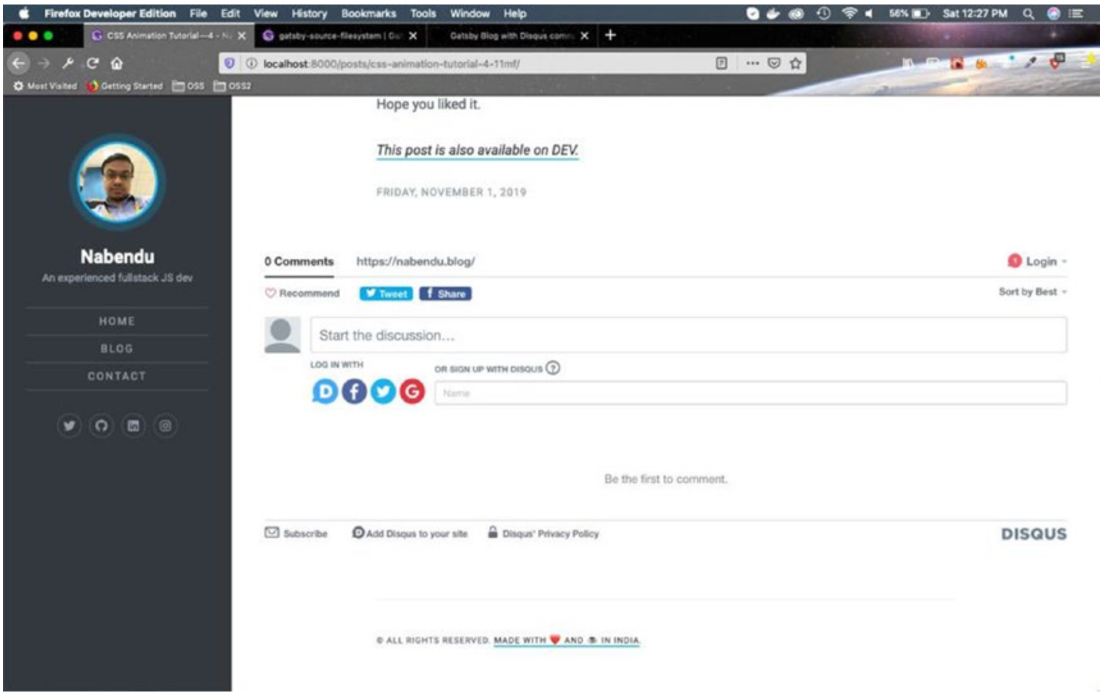
[31]https://coleruche.com/

It's time to add `CommentCount` and `Disqus` before and after your content, as shown Listing 2-8.

Listing 2-8. The post.js File

```
return (
    <Layout {...this.props}>
      <article className="post post-full">
      <CommentCount config={disqusConfig} placeholder={'...'} />
      ...
       ...
      </article>
      <Subscribe heading={true} />
      <Disqus config={disqusConfig} />
    </Layout>
);
```

The Comment section has started to appear in `localhost`, as shown in Figure 2-57.



Hope you liked it.

This post is also available on DEV.

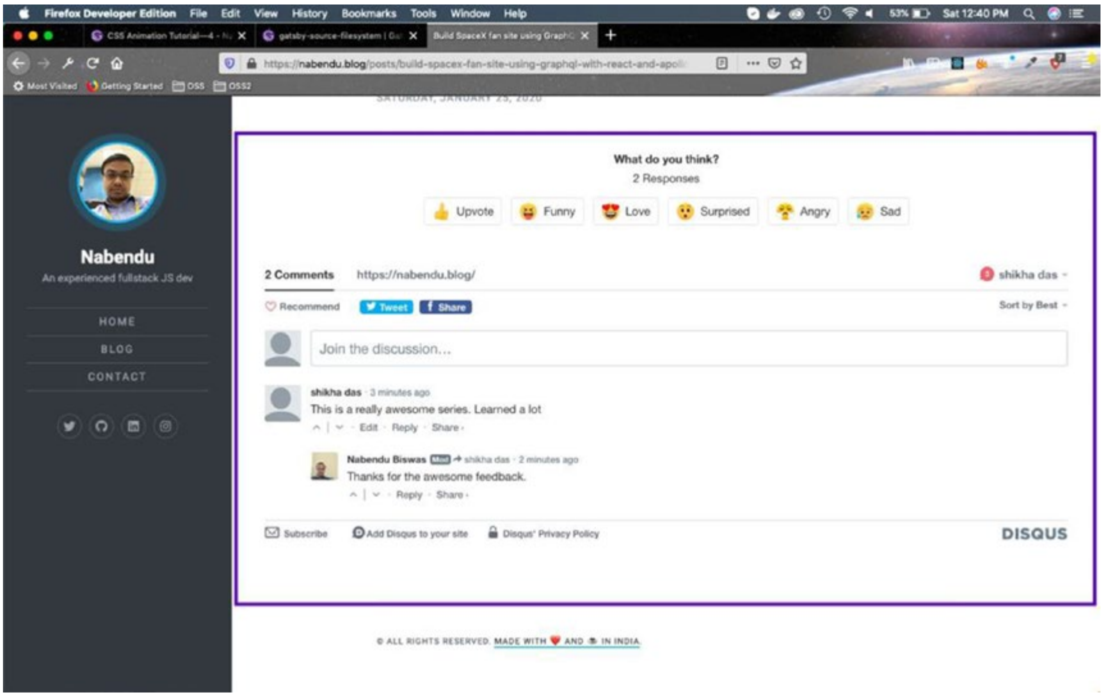
FRIDAY, NOVEMBER 1, 2019

Be the first to comment.

© ALL RIGHTS RESERVED. MADE WITH ♥ AND IN INDIA

Figure 2-57. *localhost disqus*

Next, it's time to push it to GitHub so it's automatically deployed to Netlify. After the deployment, it looks awesome and my site now has a Comment and Like section, as shown in Figure 2-58.



shikha das · 3 minutes ago
This is a really awesome series. Learned a lot

Nabendu Biswas Mod → shikha das · 2 minutes ago
Thanks for the awesome feedback.

Figure 2-58. *Just awesome*

Go ahead and start showing your love at `https://nabendu.blog/`.[32] You can find the code for this at my GitHub repo[33].

Adding More Site Features

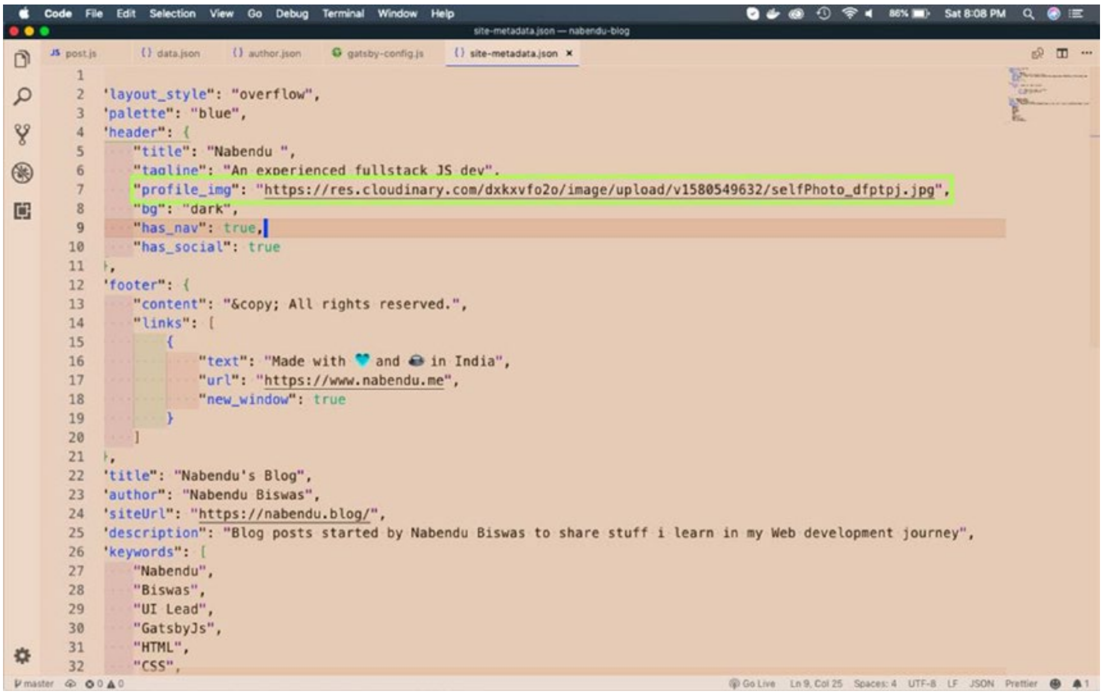
I have four features to add to my site. They are as follows:

- Change my picture on the home page
- Add a Google ad to the project

[32]`https://nabendu.blog/`
[33]`https://github.com/nabendu82/nabendu-blog`

- Integrate Mailchimp for subscriptions
- Change the favicon of the site

Let's start by changing my profile picture on the `sitemy` blog site. For this, we just need to change the `profile_img` in `site-metadata.json`, as shown in Figure 2-59. I saved my image on Cloudinary, but it can be saved on any platform on the web, which gives you access to a link.



```
'layout_style": "overflow",
'palette": "blue",
'header": {
    "title": "Nabendu ",
    "tagline": "An experienced fullstack JS dev",
    "profile_img": "https://res.cloudinary.com/dxkxvfo2o/image/upload/v1580549632/selfPhoto_dfptpj.jpg",
    "bg": "dark",
    "has_nav": true,
    "has_social": true
},
'footer": {
    "content": "&copy; All rights reserved.",
    "links": [
        {
            "text": "Made with 💙 and ☕ in India",
            "url": "https://www.nabendu.me",
            "new_window": true
        }
    ]
},
'title": "Nabendu's Blog",
'author": "Nabendu Biswas",
'siteUrl": "https://nabendu.blog/",
'description": "Blog posts started by Nabendu Biswas to share stuff i learn in my Web development journey",
'keywords": [
    "Nabendu",
    "Biswas",
    "UI Lead",
    "GatsbyJs",
    "HTML",
    "CSS",
```

Figure 2-59. *The metadata*

Adding Google AdSense to a GatsbyJS site is a bit tricky and I didn't find many articles on it. If you don't have an AdSense account activated, it's a different process.

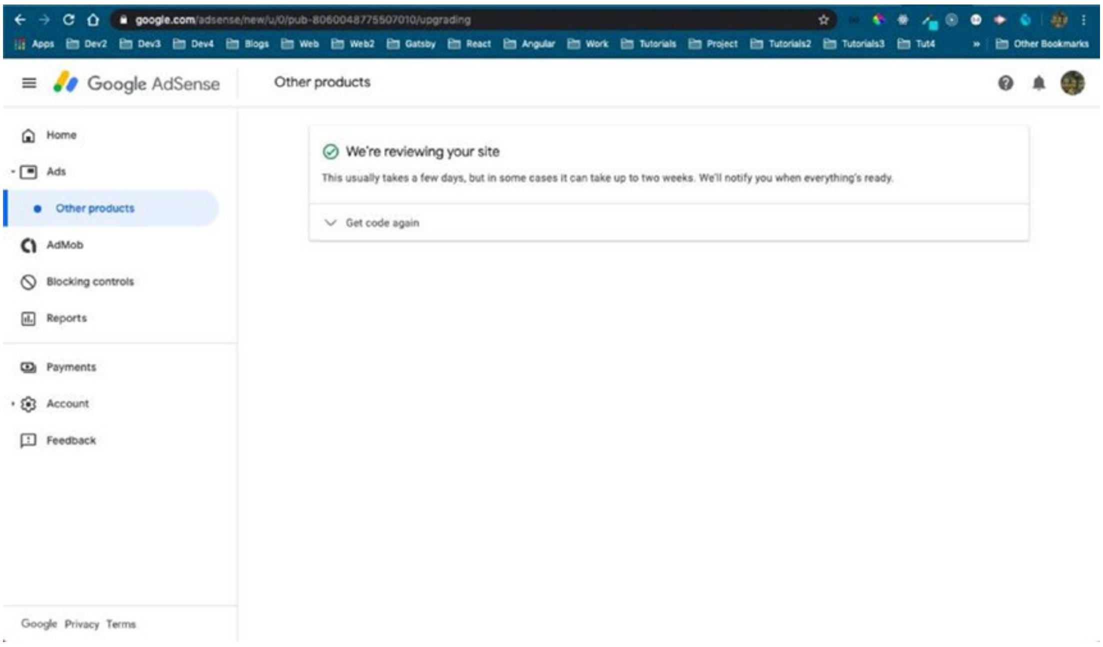
First, copy the `html.js` file in the `src` folder, using `cp .cache/default-html.js src/html.js` in the main directory.

Next, for AdSense activation, you need to add the script in Listing 2-9 inside the `<head>` tag. You need to use your own `google_ad_client`. The additions are shown in bold.

Listing 2-9. The Updated html.js File

```
import React from "react"
import PropTypes from "prop-types"

export default function HTML(props) {
  return (
    <html {...props.htmlAttributes}>
      <head>
        <meta charSet="utf-8" />
        <meta httpEquiv="x-ua-compatible" content="ie=edge" />
        <meta
          name="viewport"
          content="width=device-width, initial-scale=1, shrink-to-fit=no"
        />
        {props.headComponents}
                             <script async src="https://pagead2.
                             googlesyndication.com/pagead/js/adsbygoogle.js">
                             </script>
        <script>
          (adsbygoogle = window.adsbygoogle || []).push({
            google_ad_client: "ca-pub-8XXXXXXXXXXX",
            enable_page_level_ads: true
          });
        </script>
      </head>
      <body {...props.bodyAttributes}>
        {props.preBodyComponents}
        <noscript key="noscript" id="gatsby-noscript">
          This app works best with JavaScript enabled.
        </noscript>
        <div
          key={`body`}
          id="___gatsby"
          dangerouslySetInnerHTML={{ __html: props.body }}
        />
        {props.postBodyComponents}
      </body>
    </html>
  )
}

HTML.propTypes = {
  htmlAttributes: PropTypes.object,
  headComponents: PropTypes.array,
  bodyAttributes: PropTypes.object,
  preBodyComponents: PropTypes.array,
  body: PropTypes.string,
  postBodyComponents: PropTypes.array,
}
```

I deploy the site to Netlify and click a check box in Google AdSense to complete the process. It gives me the message shown in Figure 2-60, which means I need to wait a few days.



We're reviewing your site

This usually takes a few days, but in some cases it can take up to two weeks. We'll notify you when everything's ready.

Get code again

Figure 2-60. *Waiting for Google AdSense approval*

Now it's time to add Mailchimp to the blog. I need to register for a Mailchimp account first. After that, I followed the instructions on this blog[34] to get my end point.

Then you need to install the Mailchimp plugin by running the following command. This command needs to be run in the terminal in the root directory of your project.

```
npm install --save gatsby-plugin-mailchimp
```

After that, you need to add the code in Listing 2-10 to the `gatsby-config.js` file.

Listing 2-10. Code for the gatsby-config.js File

```
        {
            resolve: "gatsby-plugin-mailchimp",
            options: {
                endpoint: "https://blog.us4.list-manage.com/subscribe/
                post?u=5c156d7649897240e9c994d38&id=67f40851e0",
            },
        }
```

You now need to create a Subscribe component, but I want to use `sass` for styling in the project. So, I will first add a plugin for that by running the next command. This command needs to be run in the terminal in the root directory of your project.

```
npm install --save node-sass gatsby-plugin-sass
```

Next, add `gatsby-plugin-sass` to the `gatsby-config.js` file. Then create a file called `subscribe.js` inside the `components` folder and update it with the content in Listing 2-11. Here, we are just creating a class-based component and submitting it to the `add Mailchimp` function.

Listing 2-11. The subscribe.js File

```
import React, { Component } from 'react';
import addToMailchimp from "gatsby-plugin-mailchimp";
import * as styles from './subscribe.module.scss';
```

[34] https://thetrevorharmon.com/blog/email-signup-forms-with-gatsby-and-mailchimp

```
class Subscribe extends Component  {
    state = {
        email: "",
        statusMsg: "",
        statusMsgColor: "green",
        subscribing: false,
    }

    handleSubmit = (e) => {
        e.preventDefault();
        this.setState({ statusMsg: "", subscribing: true });
        let { email } = this.state;
        addToMailchimp(email)
            .then(data => {
            data.result === "success"
                ? this.setState({
                    statusMsg: "Your subscription was successful!",
                    statusMsgColor: "green",
                    email: "",
                    subscribing: false,
                    })
                : this.setState({
                    statusMsg: "This email has already been subscribed.",
                    statusMsgColor: "red",
                    subscribing: false,
                    })
            })
            .catch(err => {
                this.setState({
                statusMsg: "An error occurred. Please re-try",
                statusMsgColor: "red",
                subscribing: false,
                })
        })

    };
    render() {
        let { statusMsg, subscribing } = this.state;
        let btnCTA = subscribing ? "Subscribing" : "Subscribe";
        return (
            <form onSubmit={this.handleSubmit} className={styles.
            EmailListForm}>
                <h2>Subscribe to receive updates on new posts!</h2>
                <div className={styles.Wrapper}>
                    <input
                        placeholder="Email address"
                        name="email"
                        type="text"
                        value={this.state.email}
                        onChange={e => this.setState({ email: e.target.
                        value })}
                    />
                    <button type="submit">{btnCTA}</button>
                    {statusMsg && (
                        <div className="col-12">
                            <p
                            className="text-left"
                            style={{ color: this.state.statusMsgColor }}
                            >
                            {statusMsg}
                            </p>
                        </div>
                    )}
                </div>
            </form>
        );
    }
};

export default Subscribe;
```

Next, create a `subscribe.module.scss` module in the same directory and add the content from Listing 2-12.

Listing 2-12. subscribe.module.scss

```
.EmailListForm {
    display: flex;
    flex-direction: column;
    background: transparent;
    color: #2a2a2a;
    font-family: -apple-system, Helvetica, Arial, sans-serif;

    h2 {
        margin-top: 0;
        margin-bottom: 1rem;
    }

    .Wrapper {
        display: flex;
        flex-direction: column;
    }

    input {
        color: #2a2a2a;
        width: 100%;
        border: none;
    }

    input {
        padding: 1rem 1.5rem;
    }

    button {
        display: inline-block;
        border: none;
        background-image: none;
        background-color: #DD0505 !important;
        color: white;
        border-radius: 5px;
        letter-spacing: 1px;
        transition: all 0.1s linear;
        margin-top:1%;

        &:hover {
            cursor: pointer;
            background: darken(#DD0505, 15%) !important;
        }
    }
}
```

Next, we will add the Subscribe component to our `post.js` file, so that it is displayed after every post. It is marked in bold in Listing 2-13.

Listing 2-13. The post.js File

```
...
...
import {Layout} from '../components/index';
import Subscribe from '../components/subscribe';
import {htmlToReact, safePrefix} from '../utils';
import { Disqus, CommentCount } from "gatsby-plugin-disqus";

export default class Post extends React.Component {
    render() {
           ...
           ...
      }
        return (
            <Layout {...this.props}>
                        ...
                        ...
              <Subscribe />
              <Disqus config={disqusConfig} />
            </Layout>
        );
    }
}
```

Once it's pushed to GitHub and deployed to Netlify, people can subscribe to my blog site, as shown in Figure 2-61.

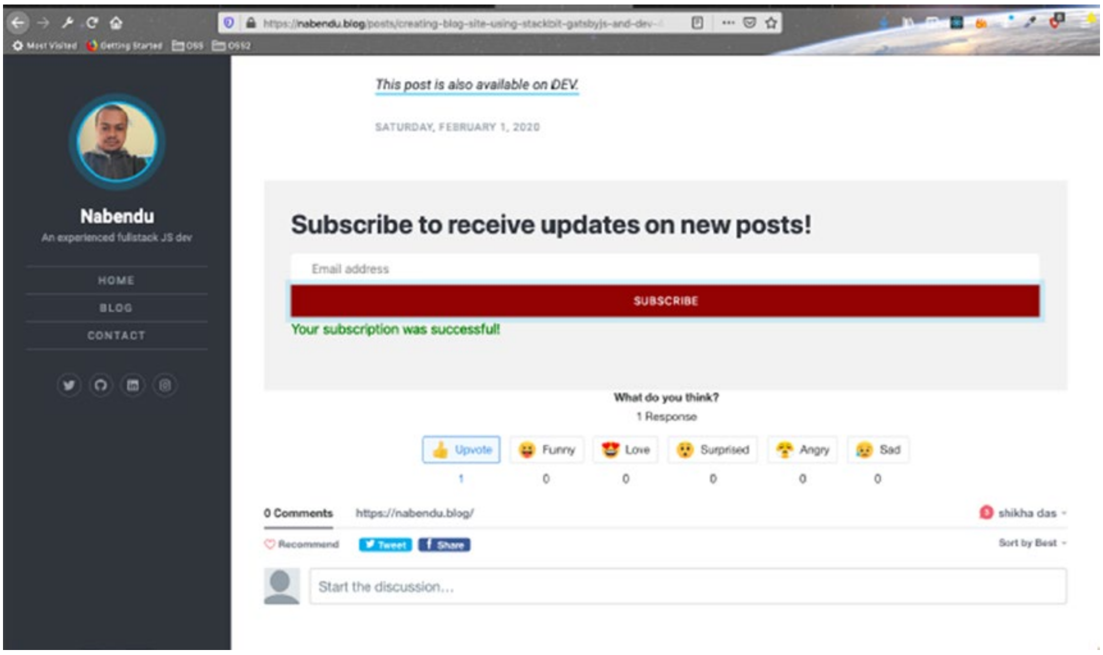


Figure 2-61. *Subscription*

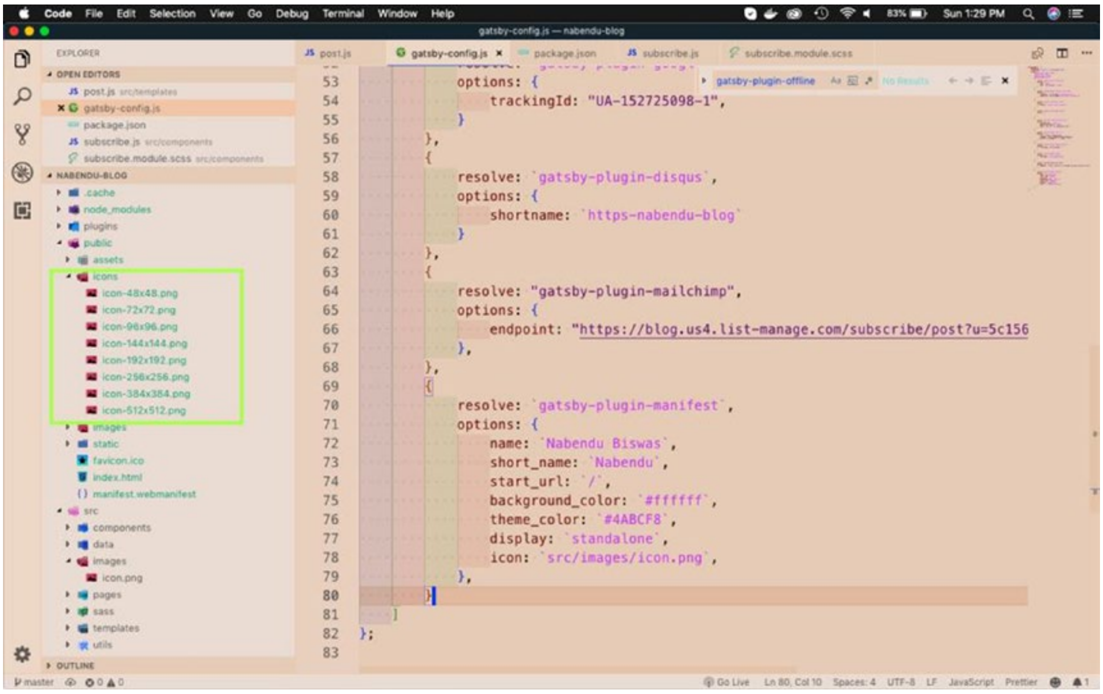
Now, we will add favicons to the site. For this, we need to use `gatsby-plugin-manifest`. Let's first install it by typing `npm install --save gatsby-plugin-manifest` in the root directory.

Then we need to add a favicon to any folder. I generated a 512x512 PNG image and created an `images` folder in `src` and placed it there. Also, I need to add the contents of Listing 2-14 to the `gatsby-config.js` file.

Listing 2-14. The gatsby-config.js File

```
{
    resolve: `gatsby-plugin-manifest`,
    options: {
        name: `Nabendu Biswas`,
        short_name: `Nabendu`,
        start_url: `/`,
        background_color: `#ffffff`,
                theme_color: `#4ABCF8`,
                display: `standalone`,
                icon: `src/images/icon.png`,
            },
        }
```

One of the best things about this plugin is that it auto-generates favicons of other sizes, as shown in Figure 2-62.



```
EXPLORER
OPEN EDITORS
    post.js src/templates
  × gatsby-config.js
    package.json
    subscribe.js src/components
    subscribe.module.scss src/components
NABENDU-BLOG
  .cache
  node_modules
  plugins
  public
    assets
    icons
      icon-48x48.png
      icon-72x72.png
      icon-96x96.png
      icon-144x144.png
      icon-192x192.png
      icon-256x256.png
      icon-384x384.png
      icon-512x512.png
    images
    static
    favicon.ico
    index.html
    manifest.webmanifest
  src
    components
    data
    images
      icon.png
    pages
    sass
    templates
    utils
OUTLINE
```

```
            options: {
                trackingId: "UA-152725098-1",
            }
        },
        {
            resolve: `gatsby-plugin-disqus`,
            options: {
                shortname: `https-nabendu-blog`
            }
        },
        {
            resolve: "gatsby-plugin-mailchimp",
            options: {
                endpoint: "https://blog.us4.list-manage.com/subscribe/post?u=5c156
            },
        },
        {
            resolve: `gatsby-plugin-manifest`,
            options: {
                name: `Nabendu Biswas`,
                short_name: `Nabendu`,
                start_url: `/`,
                background_color: `#ffffff`,
                theme_color: `#4ABCF8`,
                display: `standalone`,
                icon: `src/images/icon.png`,
            },
        }
    ]
};
```

Figure 2-62. *Other sizes of favicons*

The favicon is there, as in Figure 2-63 on my site (`https://nabendu.blog/`).[35]

[35]`https://nabendu.blog/`

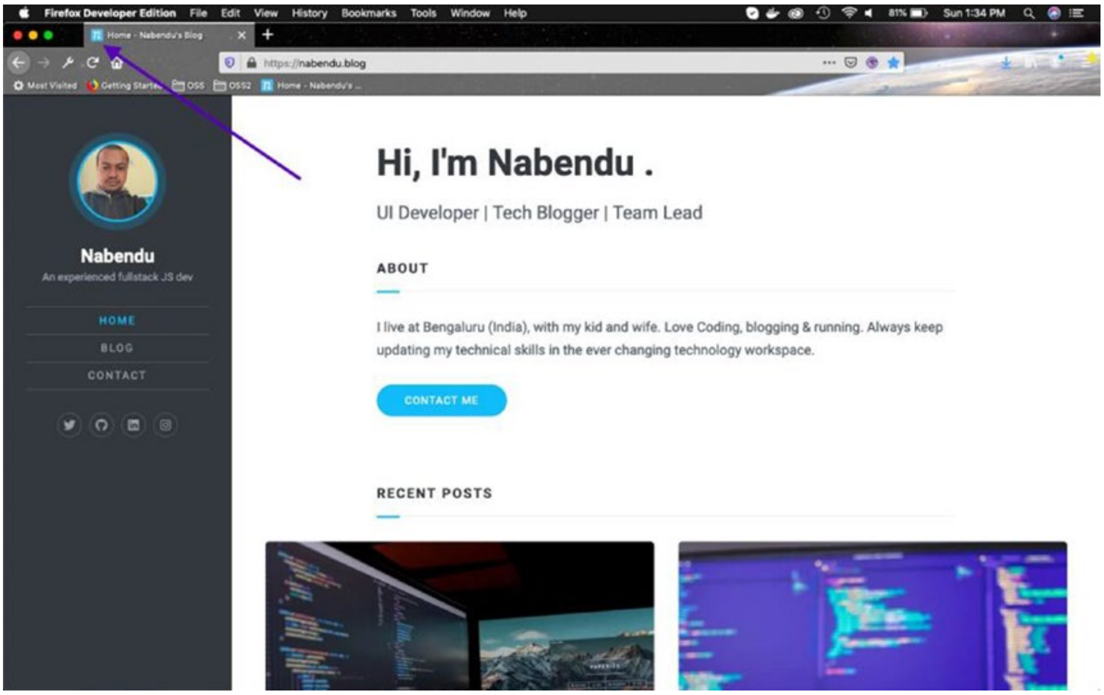

Home - Nabendu's Blog
https://nabendu.blog
Hi, I'm Nabendu .
UI Developer | Tech Blogger | Team Lead
ABOUT
I live at Bengaluru (India), with my kid and wife. Love Coding, blogging & running. Always keep updating my technical skills in the ever changing technology workspace.
CONTACT ME
RECENT POSTS
Nabendu
An experienced fullstack JS dev
HOME
BLOG
CONTACT

Figure 2-63. *The favicon is displayed*

Adding Advertisements

One of the main tasks remaining from the last post is to add advertisements to the site. I should tell you it's a great pain to add advertisements to a Gatsby site, as all the HTML files are generated on build. Most advertising networks provide you with JavaScript scripts to place in the head and body tags.

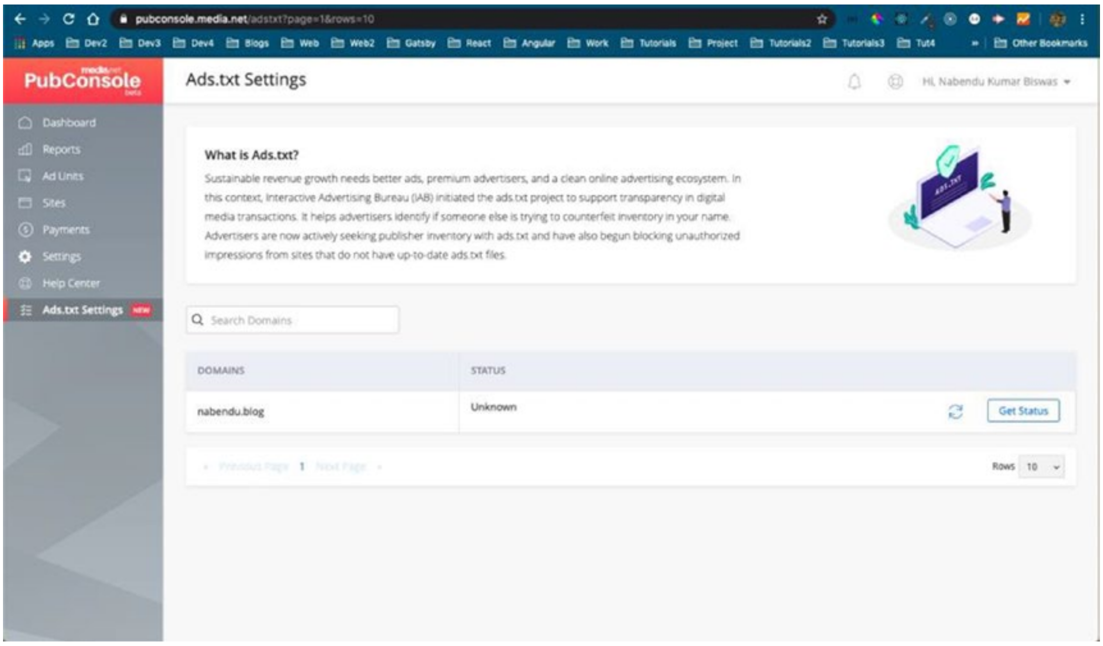
In the previous section, I applied for a Google AdSense account, but as my content is on three sites (even though I had canonical links to my primary domain `https://nabendu.blog/`[36] from DEV and Medium), it was rejected.

After searching the Internet, I found this[37] useful article with details of other popular ad networks.

[36]`https://nabendu.blog/`
[37]`https://www.adpushup.com/blog/the-best-ad-networks-for-publishers/`

I applied to Media.net, InfoLinks, and RevenueHits and got approval from all of them. I also applied to Carbon Ads[38], which is nowadays very popular with developers, but didn't get the approval yet.

I decided to add Media.net ads to my site, as they serve ads from the Yahoo network. Once you log in to the Media.net dashboard, you need to add a file called `Ads.txt` to your site root directory, as shown in Figure 2-64.



Ads.txt Settings

What is Ads.txt?

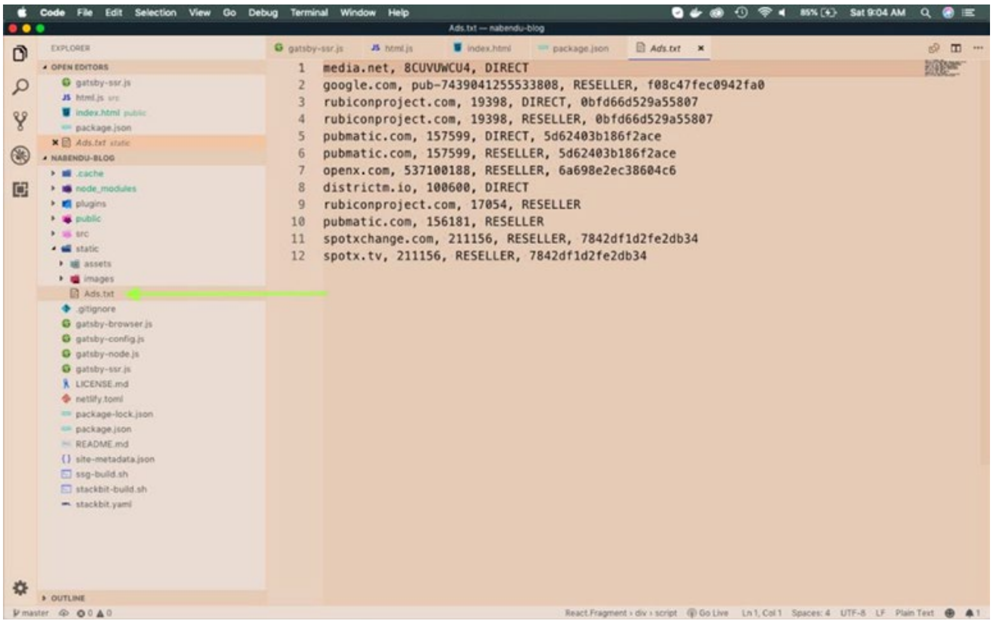
Sustainable revenue growth needs better ads, premium advertisers, and a clean online advertising ecosystem. In this context, Interactive Advertising Bureau (IAB) initiated the ads.txt project to support transparency in digital media transactions. It helps advertisers identify if someone else is trying to counterfeit inventory in your name. Advertisers are now actively seeking publisher inventory with ads.txt and have also begun blocking unauthorized impressions from sites that do not have up-to-date ads.txt files.

DOMAINS	STATUS
nabendu.blog	Unknown

Figure 2-64. *The Ads.txt file*

The Gatsby site must be placed inside the `static` folder, as shown in Figure 2-65.

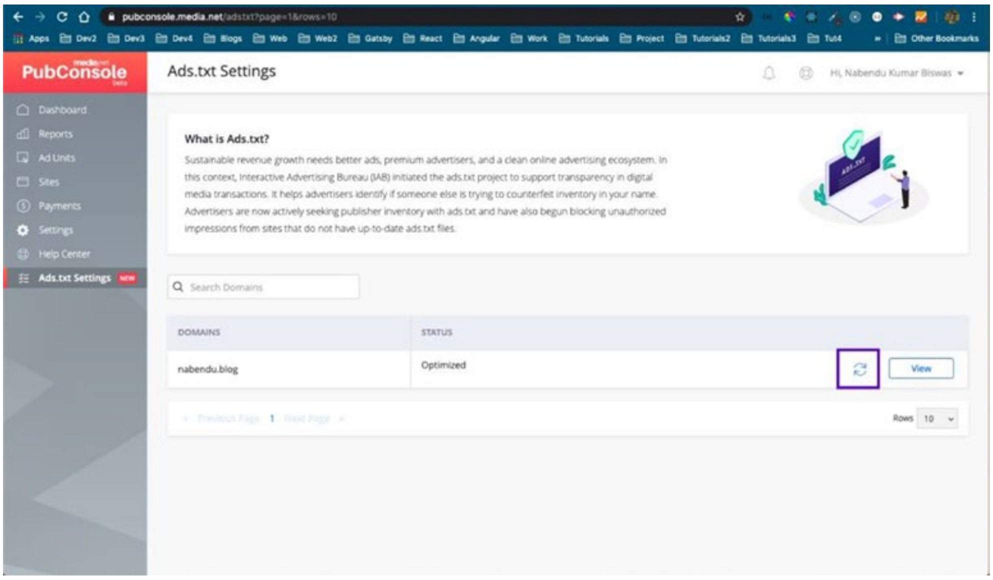
[38] `https://www.carbonads.net/`



```
media.net, 8CUVUWCU4, DIRECT
google.com, pub-7439041255533808, RESELLER, f08c47fec0942fa0
rubiconproject.com, 19398, DIRECT, 0bfd66d529a55807
rubiconproject.com, 19398, RESELLER, 0bfd66d529a55807
pubmatic.com, 157599, DIRECT, 5d62403b186f2ace
pubmatic.com, 157599, RESELLER, 5d62403b186f2ace
openx.com, 537100188, RESELLER, 6a698e2ec38604c6
districtm.io, 100600, DIRECT
rubiconproject.com, 17054, RESELLER
pubmatic.com, 156181, RESELLER
spotxchange.com, 211156, RESELLER, 7842df1d2fe2db34
spotx.tv, 211156, RESELLER, 7842df1d2fe2db34
```

Figure 2-65. *Static folder*

After deploying to Netlify, you can click the Refresh button and it will show the optimized domain, as shown in Figure 2-66.



Ads.txt Settings

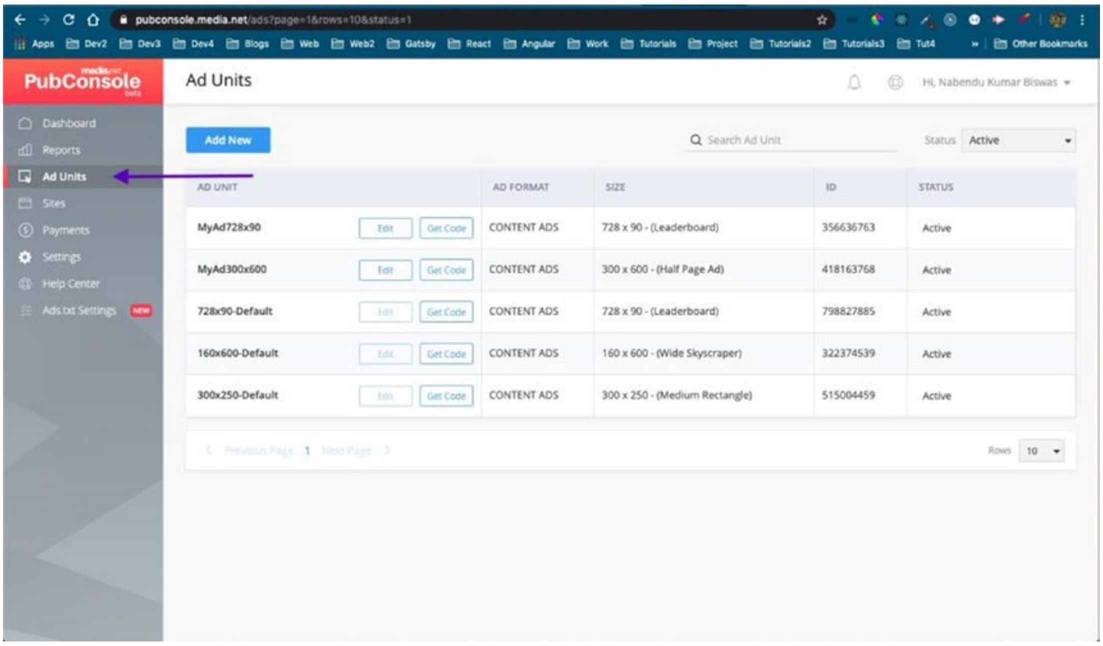
What is Ads.txt?

Sustainable revenue growth needs better ads, premium advertisers, and a clean online advertising ecosystem. In this context, Interactive Advertising Bureau (IAB) initiated the ads.txt project to support transparency in digital media transactions. It helps advertisers identify if someone else is trying to counterfeit inventory in your name. Advertisers are now actively seeking publisher inventory with ads.txt and have also begun blocking unauthorized impressions from sites that do not have up-to-date ads.txt files.

DOMAINS	STATUS
nabendu.blog	Optimized

Figure 2-66. *Optimized domain*

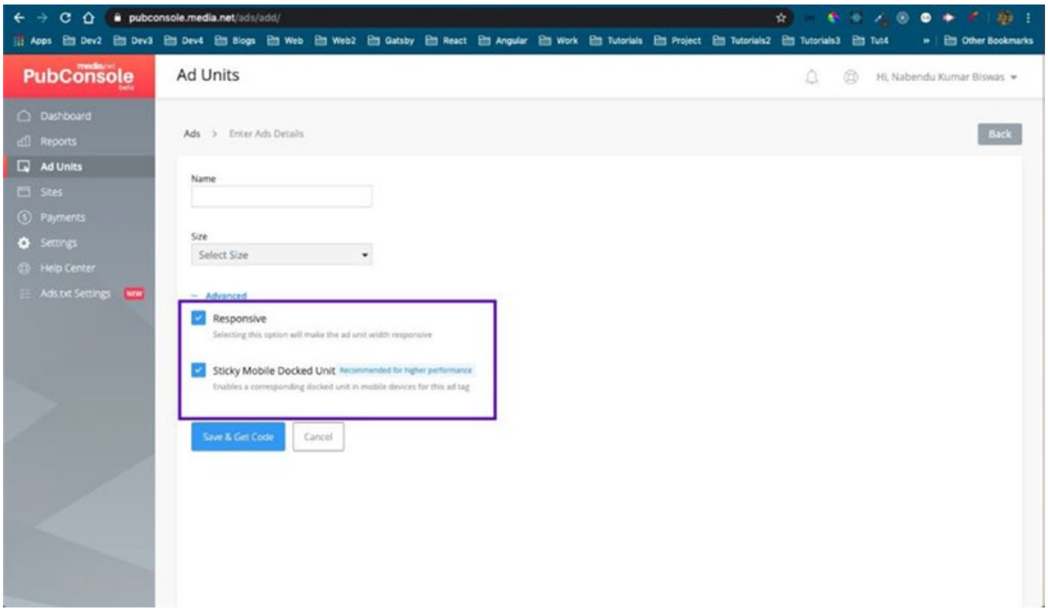
Go to the Ad Units tab to see all the ads, as shown in Figure 2-67.



AD UNIT		AD FORMAT	SIZE	ID	STATUS
MyAd728x90	Edit Get Code	CONTENT ADS	728 x 90 - (Leaderboard)	356636763	Active
MyAd300x600	Edit Get Code	CONTENT ADS	300 x 600 - (Half Page Ad)	418163768	Active
728x90-Default	Edit Get Code	CONTENT ADS	728 x 90 - (Leaderboard)	798827885	Active
160x600-Default	Edit Get Code	CONTENT ADS	160 x 600 - (Wide Skyscraper)	322374539	Active
300x250-Default	Edit Get Code	CONTENT ADS	300 x 250 - (Medium Rectangle)	515004459	Active

Figure 2-67. *Ad units*

I had a bit of a problem with the provided default ads, as they were not showing up on the desktop due to the layout of my website. So I created my own ads. One thing to keep in mind is that you should select the two check boxes shown in Figure 2-68 while creating an ad.

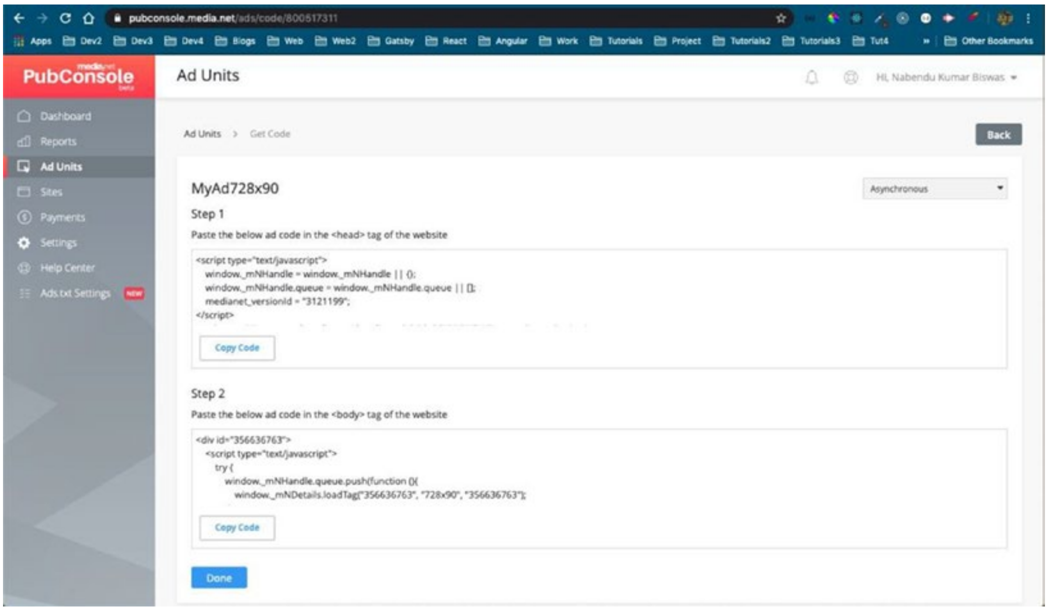


Responsive [✓]
Selecting this option will make the ad unit width responsive

Sticky Mobile Docked Unit [✓] Recommended for higher performance
Enables a corresponding docked unit in mobile devices for this ad tag

Figure 2-68. *Responsive ads*

Now, when you go inside any ad unit, you will get two scripts to be placed in the head and body tags of your website, as shown in Figure 2-69.



MyAd728x90

Step 1

Paste the below ad code in the <head> tag of the website

```
<script type="text/javascript">
  window._mNHandle = window._mNHandle || {};
  window._mNHandle.queue = window._mNHandle.queue || [];
  medianet_versionId = "3121199";
</script>
```

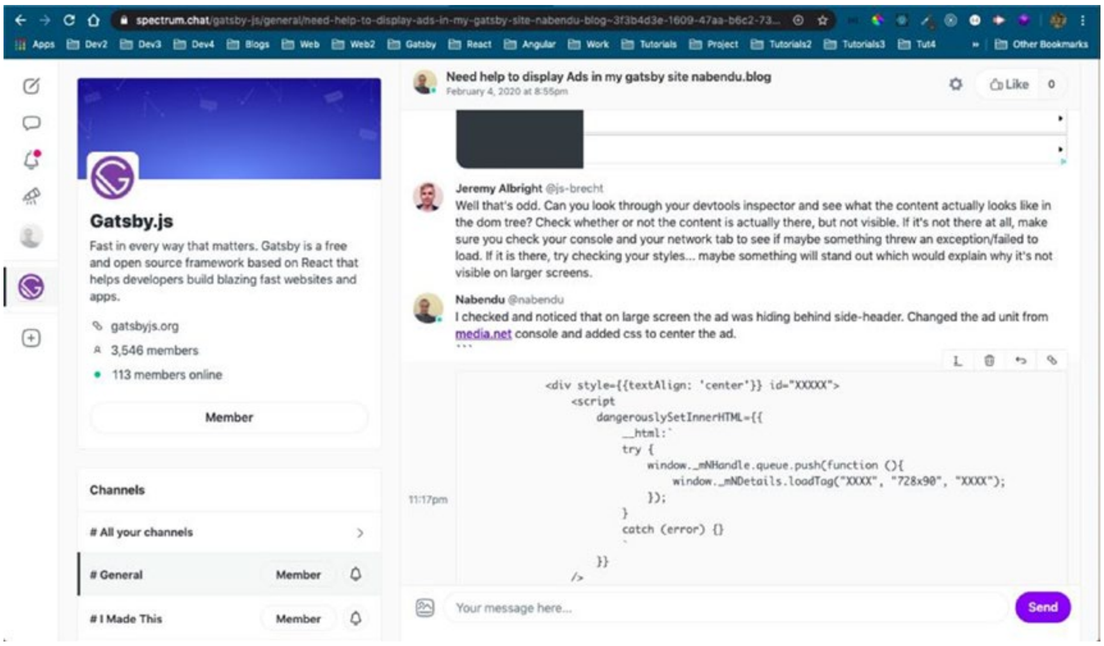
Step 2

Paste the below ad code in the <body> tag of the website

```
<div id="356636763">
  <script type="text/javascript">
    try {
      window._mNHandle.queue.push(function (){
        window._mNDetails.loadTag("356636763", "728x90", "356636763");
```

Figure 2-69. *Scripts for the head and body tags*

I was not able to figure out where to put these scripts in my Gatsby site. The tech support of Media.net and Stack Overflow also were not able to help me, as Gatsby is quite new. But I got awesome help from folks at Gatsby and came to know about their *spectrum chat*[39]. I posted in the General forum and Jeremy Albright[40] helped me through the complete process. Thanks, Jeremy, for your help (see Figure 2-70).



Fast in every way that matters. Gatsby is a free and open source framework based on React that helps developers build blazing fast websites and apps.

Need help to display Ads in my gatsby site nabendu.blog
February 4, 2020 at 8:55pm

Jeremy Albright @js-brecht
Well that's odd. Can you look through your devtools inspector and see what the content actually looks like in the dom tree? Check whether or not the content is actually there, but not visible. If it's not there at all, make sure you check your console and your network tab to see if maybe something threw an exception/failed to load. If it is there, try checking your styles... maybe something will stand out which would explain why it's not visible on larger screens.

Nabendu @nabendu
I checked and noticed that on large screen the ad was hiding behind side-header. Changed the ad unit from media.net console and added css to center the ad.

11:17pm

```
<div style={{textAlign: 'center'}} id="XXXXX">
    <script
        dangerouslySetInnerHTML={{
            __html:`
            try {
                window._mNHandle.queue.push(function (){
                    window._mNDetails.loadTag("XXXX", "728x90", "XXXX");
                });
            }
            catch (error) {}
            `
        }}
    />
```

Figure 2-70. Gatsby help

I was pointed to a great article[41] about how to insert scripts in the head and body tags of the site. You need to update the `gatsby-ssr.js` file in the root directory.

You must have `setHeadComponents`, `setPostBodyComponents`, and `setPreBodyComponents` in the `onRenderBody` function. The head tag script will come in `setHeadComponents`, but it needs to be updated to have `dangerouslySetInnerHTML`.

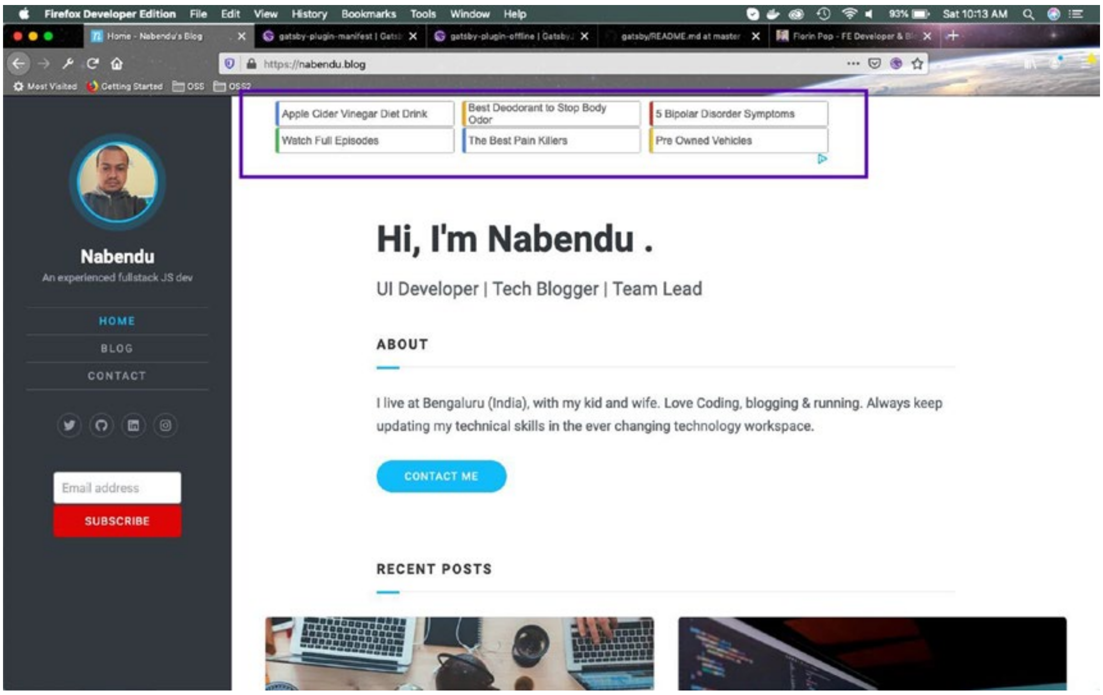
[39] https://spectrum.chat/gatsby-js/general?tab=posts
[40] https://spectrum.chat/users/js-brecht
[41] https://uxworks.online/how-to-add-a-script-in-head-or-body-tag-in-your-gatsby-website/

Similarly, the body tag script will come in `setPreBodyComponents` with `dangerouslySetInnerHTML`. Also, notice that I put an inline `style={{textAlign: 'center'}}` to center the ad. I did this because my ads were not properly lining up on the desktop. The updated `gatsby-ssr.js` file is shown in Listing 2-15.

Listing 2-15. The Updated gatsby-ssr.js File

```
const React = require("react");
const safePrefix = require("./src/utils/safePrefix").default;

exports.onRenderBody = function({ setHeadComponents, setPostBodyComponents,
setPreBodyComponents }) {

    setHeadComponents([
        <script
        dangerouslySetInnerHTML={{
            __html:`
            window._mNHandle = window._mNHandle || {};
            window._mNHandle.queue = window._mNHandle.queue || [];
            medianet_versionId = "3121199";
            `
        }}
    />,
    <script src="//contextual.media.net/dmedianet.js?cid=8CUVUWCU4"
    async="async" />
    ]);

    setPreBodyComponents([
            <div style={{maxWidth: '54vw', margin: '0 auto'}} id="356636763">
                <script
                    dangerouslySetInnerHTML={{
                        __html:`
                        try {
                            window._mNHandle.queue.push(function (){
                                window._mNDetails.loadTag("356636763",
                                "728x90", "356636763");
                            });
                        }
                        catch (error) {}
                        `
                    }}
                />
            </div>
    ]);

    setPostBodyComponents([
        <React.Fragment>
            <script src={safePrefix('assets/js/plugins.js')}/>
            <script src={safePrefix('assets/js/main.js')}/>
            <div style={{maxWidth: '54vw', margin: '0 auto'}}
            id="104240845">
                <script
                    dangerouslySetInnerHTML={{
                        __html:`
                        try {
                            window._mNHandle.queue.push(function (){
                                window._mNDetails.loadTag("104240845",
                                "300x250", "104240845");
                            });
                        }
                        catch (error) {}
                        `
                    }}
                />
            </div>
            <div id="172542266">
                <script
                    dangerouslySetInnerHTML={{
                        __html:`
                        try {
                            window._mNHandle.queue.push(function (){
                                window._mNDetails.loadTag("172542266",
                                "320x50", "172542266");
                            });
            }
            catch (error) {}
            `

          }}
        />
      </div>
    </React.Fragment>
  ]);
};
```

Commit the changes and deploy in Netlify. You will get ads shown in all pages and blogs before it, as shown in Figure 2-71.



Hi, I'm Nabendu .

UI Developer | Tech Blogger | Team Lead

ABOUT

I live at Bengaluru (India), with my kid and wife. Love Coding, blogging & running. Always keep updating my technical skills in the ever changing technology workspace.

RECENT POSTS

Figure 2-71. *Ads*

You will get ads shown in all pages and blogs after it, as shown in Figure 2-72.

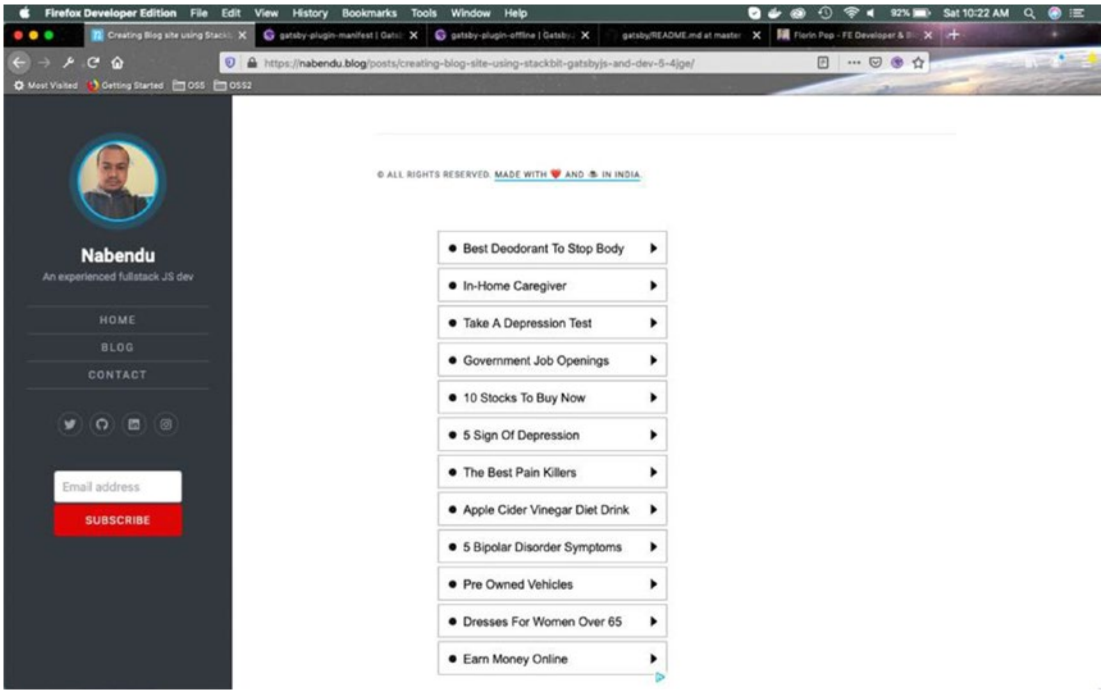


Figure 2-72. *After ads*

Making Minor Updates

I made one more update to my site, as you might have noticed in Figure 2-72. I added a subscription component to my sidebar. Earlier, my subscription component was shown after each blog.

In the `subscribe.js` file, I am expecting `props` now, which will decide whether to show the headings. Update the line marked in bold in Listing 2-16.

Listing 2-16. The Updated subscribe.js File

```
render() {
    let { statusMsg, subscribing } = this.state;
    let btnCTA = subscribing ? "Subscribing" : "Subscribe";
    return (
        <form onSubmit={this.handleSubmit} className={styles.
        EmailListForm}>
            {this.props.heading && <h2>Subscribe to receive updates on
            new posts!</h2>}
```

```
                <div className={styles.Wrapper}>
                                    ...
                                    ...
                </div>
            </form>
        );
    }
```

One of the other changes I made was to make the background transparent in `subscribe.module.scss`. This change is marked in bold in Listing 2-17.

Listing 2-17. The Updated subscribe.module.scss File

```
.EmailListForm {
    display: flex;
    flex-direction: column;
    background: transparent;
    color: #2a2a2a;
    font-family: -apple-system, Helvetica, Arial, sans-serif;
    ...
    ...
    ...
}
```

It's time to update it, first in `post.js`, and pass the new `props` as `true`. These changes are marked in bold in Listing 2-18.

Listing 2-18. The Updated post.js File

```
        return (
            <Layout {...this.props}>
                        ...
                        ...
              <Subscribe heading={true} />
              <Disqus config={disqusConfig} />
            </Layout>
        );
```

This will now show the updated Subscribe component without any background color after every post, as shown in Figure 2-73.

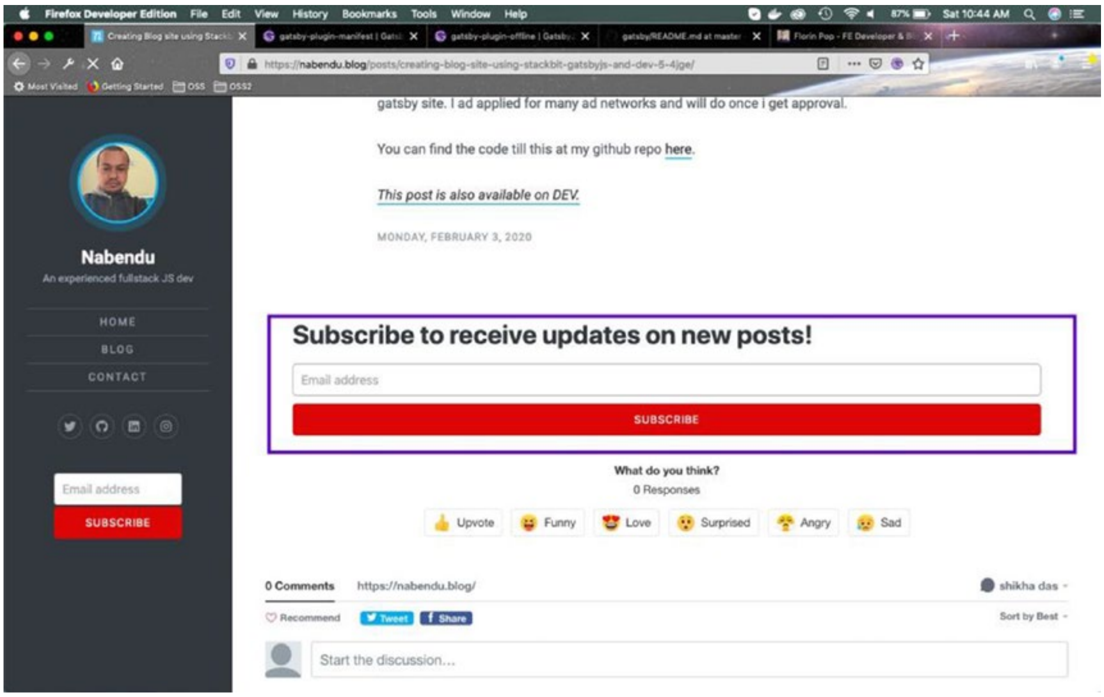

gatsby site. I ad applied for many ad networks and will do once i get approval.
You can find the code till this at my github repo here.
This post is also available on DEV.
MONDAY, FEBRUARY 3, 2020
Nabendu
An experienced fullstack JS dev
HOME
BLOG
CONTACT
Email address
SUBSCRIBE
Subscribe to receive updates on new posts!
Email address
SUBSCRIBE
What do you think?
0 Responses
Upvote
Funny
Love
Surprised
Angry
Sad
0 Comments
https://nabendu.blog/
Recommend
Tweet
Share
Sort by Best
Start the discussion...

Figure 2-73. *The updated Subscribe button*

To show the Subscribe component in the sidebar, I updated the `Social.js` file. I included it in the file, followed by adding a React fragment, and added it by passing the `props` heading as `false`. It is marked in bold in Listing 2-19.

Listing 2-19. Social.js

```
import React from 'react';
import _ from 'lodash';
import {Link} from '../utils';
import Subscribe from './subscribe';
import Advert from './Advert';
```

```
export default class Social extends React.Component {
    render() {
        return (

            <div className="social-links">
           ...
            ...
            </div>
            <Advert />
            <Subscribe heading={false} />

        );
    }
}
```

It is showing the mini-`subscribe` component in the sidebar. Subscribe to my weekly newsletter, which contains the updates on my posts. You can find this code at my GitHub repo.[42]

Summary

This completes Chapter 2 and the blog site with Stackbit project. You can use this chapter to create your own personal blog site with your `dev.to` posts. We covered the following topics in this chapter:

- Creating a Gatsby blog site from your `dev.to` posts using Stackbit
- Buying a domain from namecheap and configuring it
- Adding Gatsby plugins to the site, which add easy feature integration to the site
- Adding advertisements to the site using the Media.net ad network

In the next chapter, we are going to learn how to create a tourism site with a backend system that uses Contentful, an awesome CMS.

[42]https://github.com/nabendu82/nabendu-blog

CHAPTER 3

Creating a Tourism Site with Contentful: Part One

After creating two sites with GatsbyJS in the past two chapters, it's time to create a site about the World Heritage place in India, known as *Hampi*. We will be using a CMS called Contentful in the project to display the data stored in it. We will also store the blogs in Contentful, which will be used in the site.

The Setup

Let's head over to a terminal and create a new Gatsby project called `gatsbyTourism`, using the `hello-world` starter kit. The command is shown in Listing 3-1.

Listing 3-1. The npm install Command

```
gatsby new gatsbyTourism https://github.com/gatsbyjs/gatsby-starter-hello-
world
```

Next, we will change to the directory and run `gatsby develop` (see Listing 3-2) to start our project on the localhost.

Listing 3-2. Change the Directory

```
cd gatsbyTourism
gatsby develop
```

It will start our basic `hello-world` starter, as shown in Figure 3-1, which will just show `Hello World!` on `http://localhost:8000/`[1]

[1]`http://localhost:8000/`

© Nabendu Biswas 2021
N. Biswas, *Foundation Gatsby Projects*, https://doi.org/10.1007/978-1-4842-6558-1_3

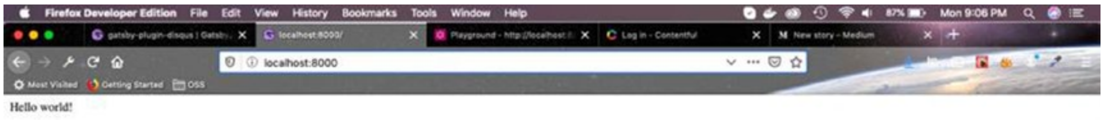


Figure 3-1. *Hello World*

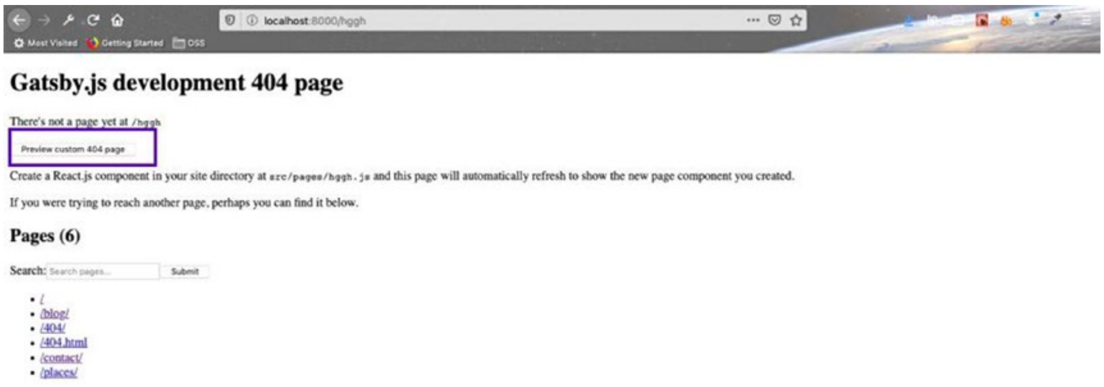
We will open our code in VSCode. Any page we create inside the `pages` folder will become an endpoint in the browser. We don't have to implement anything like `react-router` here. We will create four pages required for our project—Blog, Contact, Places, and 404.

We can create any type of React component, but we will make functional components as of now, for consistency. The `index.js` and `404.js` files are special pages and are displayed in home and error. We will create the `404.js` file with the content shown in Listing 3-3.

Listing 3-3. The 404.js File

```
import React from "react"

export default function error() {
return (
        <div>
            Error Page
        </div>
    )
}
```

If you move to a nonexistent page, you will see Figure 3-2.



localhost:8000/hggh

Gatsby.js development 404 page

There's not a page yet at `/hggh`

Preview custom 404 page

Create a React.js component in your site directory at `src/pages/hggh.js` and this page will automatically refresh to show the new page component you created.

If you were trying to reach another page, perhaps you can find it below.

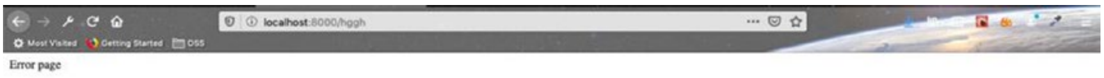
Pages (6)

Search: Search pages... Submit

- /
- /blog/
- /404/
- /404.html
- /contact/
- /places/

Figure 3-2. *The preview page*

Upon clicking the Preview Custom 404 page, we will get the error page shown in Figure 3-3.



Error page

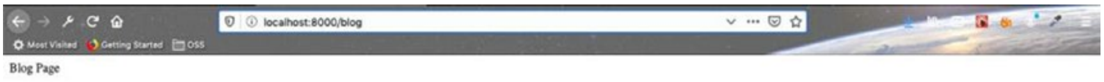
Figure 3-3. *The error page*

We will create the `blog.js` page next, with the contents in Listing 3-4.

Listing 3-4. The blog.js Page

```
import React from "react"

export default function blog() {
return (
        <div>
            Blog Page
        </div>
    )
}
```

Now, on moving to `http://localhost:8000/blog`,[2] we will see our blog page, as shown in Figure 3-4.



Blog Page

Figure 3-4. *The blog page*

We will create the `contact.js` and `places.js` pages in a similar manner. The contents are shown in Listing 3-5 and Listing 3-6, respectively.

[2]`http://localhost:8000/blog`

Listing 3-5. The contact.js Page

```
import React from "react"

export default function contact() {
return (
        <div>
            Contact Page
        </div>
    )
}
```

Listing 3-6. The places.js Page

```
import React from "react"

export default function places() {
return (
        <div>
            Places Page
        </div>
    )
}
```

Now, let's add `navbar` and `footer` components. We will make them inside the `components` folder, which will be inside the `src` folder. Place the contents of Listing 3-7 in the `Navbar.js` file.

Listing 3-7. The Navbar.js File

```
import React from "react"

export default function Navbar() {
return (
        <div>
            Navbar Component
        </div>
    )
}
```

Likewise, place the contents of Listing 3-8 in the Footer.js file.

Listing 3-8. The Footer.js File

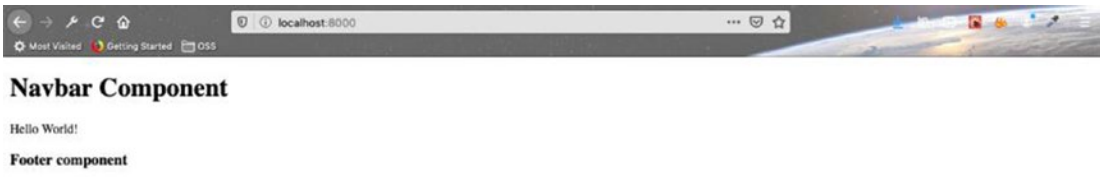
```
import React from "react"

export default function Footer() {
return (
        <div>
            Footer Component
        </div>
    )
}
```

The most common React way to show these two components on any page is to import them and show it. We will change the index.js file with the content in Listing 3-9.

Listing 3-9. The index.js File

```
import React from "react"
import Navbar from "../components/Navbar"
import Footer from "../components/Footer"

export default () => (

<Navbar />
    Hello World!
<Footer />

)
```

Figure 3-5 shows the component that will appear.



Navbar Component

Hello World!

Footer component

Figure 3-5. *Showing the component*

We could do this for every page, but Gatsby provides an easier solution. We will use a Layout component and include the `navbar` and the `footer` components there. We will also pass the children `props` to the Layout component. It will be obvious in a minute why we use this, after we use the Layout component in our pages.

So, create a `Layout.js` file inside the `components` folder and place Listing 3-10's contents in that file.

Listing 3-10. The Layout.js File

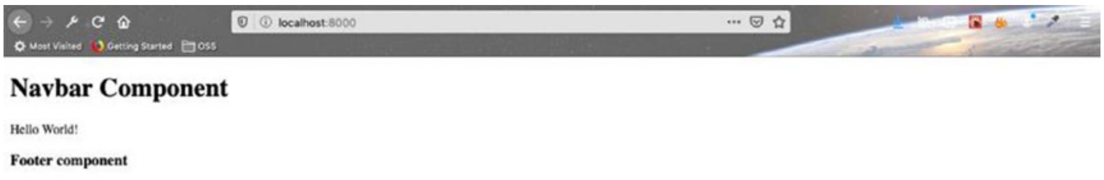
```
import React from 'react'
import Navbar from './Navbar'
import Footer from './Footer'

const Layout = ({children}) => {
    return (

          <Navbar />
                 {children}
           <Footer />

    )
}

export default Layout;
```

Next, let's use this in the index.js file. As you might have noticed, the Layout component wraps all the other things, which is only `Hello World!` now. These are the children, which is the `props` passed to the Layout component. Update the `index.js` file with the contents in Listing 3-11.

Listing 3-11. The index.js File

```
import React from "react"
import Layout from "../components/Layout"

export default () => (
<Layout>
      Hello World!
</Layout>
)
```

The home page is still the same, as shown in Figure 3-6.



Navbar Component

Hello World!

Footer component

Figure 3-6. The home page

We can use the reusable Layout component in all our other pages and they will show the `navbar` and `footer` components. Update `blog.js` with the contents in Listing 3-12.

Listing 3-12. The blog.js File

```
import React from "react"
import Layout from "../components/Layout"

export default function blog() {
return (
        <Layout>
            Blog Page
        </Layout>
    )
}
```

Update `places.js` with the contents in Listing 3-13.

Listing 3-13. The places.js File

```
import React from "react"
import Layout from "../components/Layout"

export default function places() {
return (
        <Layout>
            Places Page
        </Layout>
    )
}
```

Update `contact.js` with the contents in Listing 3-14.

Listing 3-14. The contact.js File

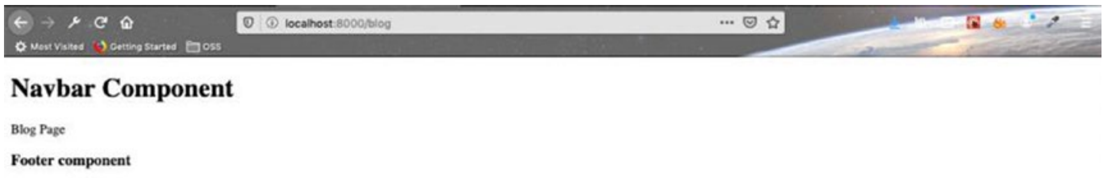
```
import React from "react"
import Layout from "../components/Layout"

export default function contact() {
return (
        <Layout>
            Contact Page
        </Layout>
    )
}
```

Update `404.js` with the contents in Listing 3-15.

Listing 3-15. The 404.js File

```
import React from "react"
import Layout from "../components/Layout"

export default function error() {
return (
        <Layout>
            Error Page
        </Layout>
    )
}
```

If you use any other path, you will see `navbar` and `footer` present there as well, as shown in Figure 3-7.



Navbar Component

Blog Page

Footer component

Figure 3-7. *The blog components*

I hope you learned something new. You can find the code at this[3] link. We don't go through the CSS in this series, as it's a Gatsby series.

The global CSS is in the `layout.css` file, in the `components` directory. You can get the contents from my GitHub[4].

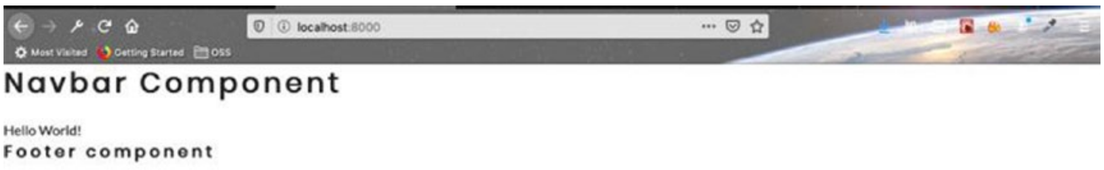
Place an `images` folder inside the `src` folder. The contents are on my GitHub as well.

We also need to import the `layout.css` file in `Layout.js`, which is marked in bold in Listing 3-16.

Listing 3-16. The Layout.js File

```
import React from 'react'
import Navbar from './Navbar'
import Footer from './Footer'
import './layout.css'

const Layout = ({children}) => {
    return (

        <Navbar />
            {children}
        <Footer />

    )
}

export default Layout;
```

This will show the home page at `http://localhost:8000/`[5] with the new fonts, as shown in Figure 3-8.



Navbar Component

Hello World!
Footer component

Figure 3-8. *The home page with new fonts*

[3]https://github.com/nabendu82/gatsbyTourism
[4]https://github.com/nabendu82/gatsbyTourism
[5]http://localhost:8000/

We will now install a package called `react-icons`, which will help us show nice icons in the project. Go ahead and exit `gatsby develop` and then `npm install` on the package, in your project directory, with the command shown in Listing 3-17.

Listing 3-17. The npm install Command

```
npm install --save react-icons
-
```

Make sure to run your `gatsby develop` again. Next, we will create a folder of constants inside `src` and create two files (`links.js` and `social-icons.js`) inside that folder.

The contents of `links.js` are shown in Listing 3-18. We will use this code at various places in this project to navigate to different links.

Listing 3-18. The links.js File

```
export default [
  {
    path: "/",
    text: "home",
  },
  {
    path: "/places",
    text: "places",
  },
  {
    path: "/blog",
    text: "blog",
  },
  {
    path: "/contact",
    text: "contact",
  }
]
```

The contents of `social-icons.js` are in Listing 3-19. It contains our icons from `react-icons`, which we will show at various places in this project.

Listing 3-19. The social-icons.js File

```
import React from "react"
import { FaFacebook, FaTwitterSquare, FaInstagram } from "react-icons/fa"

export default [
  {
    icon: <FaFacebook />,
    url: "https://facebook.com/nabendu.biswas.77",
  },
  {
    icon: <FaTwitterSquare />,
    url: "https://twitter.com/nabendu82",
  },
  {
    icon: <FaInstagram />,
    url: "https://www.instagram.com/nabendu82/",
  },
]
```

Next, we will create a `css` folder inside our `src` folder. Place the `navbar.module.css` file inside it. You can get the contents from Listing 3-20.

Listing 3-20. The navbar.module.css File

```
.nav-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 1rem 1.25rem;
}
.brand-logo {
  width: 170px;
  height: 40px;
}
```

```
.logo-btn {
  background: transparent;
  border: none;
  outline: none;
}
.logo-btn:hover {
  cursor: pointer;
}
.logo-icon {
  color: var(--primaryColor);
  font-size: 1.5rem;
}

.nav-links {
  list-style-type: none;
  transition: var(--mainTransition);
  height: 0;
  overflow: hidden;
}
.show-nav {
  height: 268px;
}
.nav-links a {
  display: block;
  padding: 1rem 1.25rem;
  text-decoration: none;
  text-transform: capitalize;
  color: var(--mainBlack);
  transition: var(--mainTransition);
  font-weight: bold;
  letter-spacing: var(--mainSpacing);
}
.nav-links a:hover {
  color: var(--primaryColor);
}
```

```
.nav-social-links {
  display: none;
}
@media screen and (min-width: 576px) {
  .navbar {
    padding: 0 2rem;
  }
}

@media screen and (min-width: 992px) {
  .logo-btn {
    display: none;
  }
  .nav-center {
    max-width: 1170px;
    margin: 0 auto;
    display: flex;
    justify-content: space-between;
    align-items: center;
  }
  .nav-links {
    height: auto;
    display: flex;
  }
  .nav-social-links {
    display: flex;
    line-height: 0;
  }
  .nav-social-links a {
    color: var(--primaryColor);
    margin: 0 0.5rem;
    font-size: 1.2rem;
    transition: var(--mainTransition);
  }
  .nav-social-links a:hover {
    color: var(--mainBlack);
    transform: translateY(-5px);
  }
}
```

Navbar and Footer

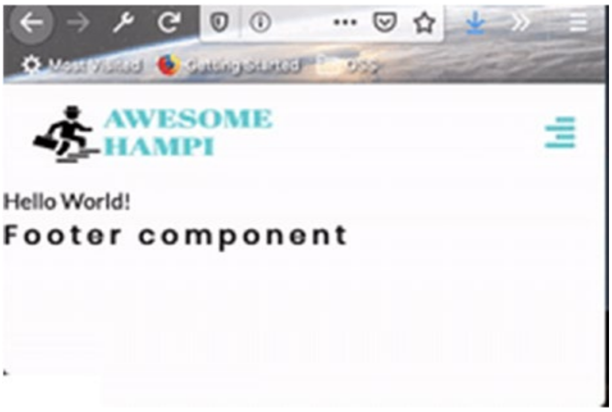
Let's start creating the navbar component. Go ahead and update your `Navbar.js` file with Listing 3-21.

Listing 3-21. The Updated Navbar.js File

```
import React, { useState } from "react"
import { Link } from "gatsby"
import styles from "../css/navbar.module.css"
import { FaAlignRight } from "react-icons/fa"
import links from "../constants/links"
import socialIcons from "../constants/social-icons"
import logo from "../images/logo.png"
const Navbar = () => {
    const [isOpen, setNav] = useState(false);
    const toggleNav = () => {
        setNav(isOpen => !isOpen)
    }

    return (
        <nav className={styles.navbar}>
            <div className={styles.navCenter}>
                <div className={styles.navHeader}>
                    <img src={logo} className={styles.brandLogo}
                    alt="backroads logo" />
                   <button type="button" className={styles.logoBtn}
                   onClick={toggleNav}>
                        <FaAlignRight className={styles.logoIcon} />
                    </button>
                </div>
            <ul className={isOpen ? `${styles.navLinks} ${styles.showNav}` :
            `${styles.navLinks}`}>
                    {links.map((item, index) => {
                        return (
                            <li key={index}>
                                <Link  to={item.path}>{item.text}</Link>
                            </li>
                        )
                    })}
                </ul>
                <div className={styles.navSocialLinks}>
                    {socialIcons.map((item, index) => {
                        return (
                            <a key={index} href={item.url} target="_blank"
                            rel="noopener noreferrer">
                                {item.icon}
                            </a>
                        )
                    })}
                </div>
            </div>
        </nav>
    )
}

export default Navbar
```

Here, we are using the `useState` hook to toggle our links on a smaller screen. You can learn more about hooks from this post: "Understanding React Hooks by Building a Simple App"[6]. First we show a `react-icon FaAlignRight` on a smaller screen. When the user clicks the button, we call the `toggleNav` function, which will set `isOpen` to `true`. When `isOpen` is `true` we will be loading different CSS classes, which will basically open a drawer to show the links on a smaller screen, as shown in Figure 3-9.

[6]https://dev.to/nabendu82/understanding-react-hooks-by-building-a-simple-app-4i6d

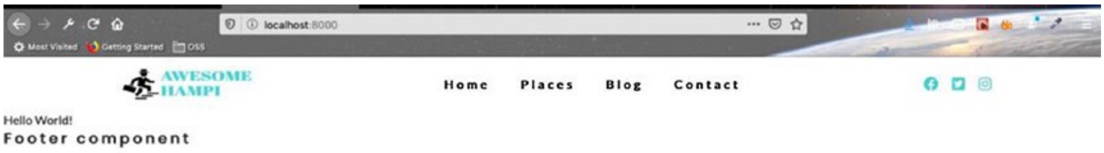


Hello World!

Footer component

Figure 3-9. *The smaller screen*

On a desktop, we get the whole menu and the `react-icon` will be hidden, as shown in Figure 3-10.



Hello World!

Footer component

Figure 3-10. *Navbar on a desktop*

We will create a footer first. Inside the `css` folder, add an `footer.module.css` file. You can get the contents from Listing 3-22.

Listing 3-22. The footer.module.css File

```
.footer {
  margin-top: auto;
  background: var(--mainBlack);
  padding: 2rem;
  text-align: center;
  color: var(--mainWhite);
}
.links a {
  display: inline-block;
  text-decoration: none;
  text-transform: uppercase;
  color: var(--mainWhite);
  margin: 0.5rem 1rem;
  letter-spacing: var(--mainSpacing);
  transition: var(--mainTransition);
  font-weight: bold;
}
.links a:hover {
  color: var(--primaryColor);
}
.icons a {
  display: inline-block;
  margin: 1rem;
  font-size: 1.3rem;
  color: var(--mainWhite);
  transition: var(--mainTransition);
}
.icons a:hover {
  color: var(--primaryColor);
}
.copyright {
  text-transform: capitalize;
  letter-spacing: var(--mainSpacing);
  line-height: 2;
}
```

Next, we will update our `Footer.js` component to use this CSS. We are just mapping through our links and social icons from the constant files and displaying them. We also have a copyright section. The contents are shown in Listing 3-23.

Listing 3-23. The Footer.js File

```
import React from "react"
import styles from "../css/footer.module.css"
import links from "../constants/links"
import socialIcons from "../constants/social-icons"
import { Link } from "gatsby"
```

```
const Footer = () => {
return (
    <footer className={styles.footer}>
        <div className={styles.links}>
        {links.map((item, index) => {
            return (
            <Link key={index} to={item.path}>
                {item.text}
            </Link>
            )
        })}
        </div>
        <div className={styles.icons}>
        {socialIcons.map((item, index) => {
            return (
     <a key={index} href={item.url} target="_blank" rel="noopener
     noreferrer">{item.icon}</a>
            )
        })}
        </div>
        <div className={styles.copyright}>
            copyright &copy; Amazing Hampi {new Date().getFullYear()} all
            rights reserved
        </div>
    </footer>
)
}

export default Footer;
```

It will show our footer in desktop view, as shown in Figure 3-11.

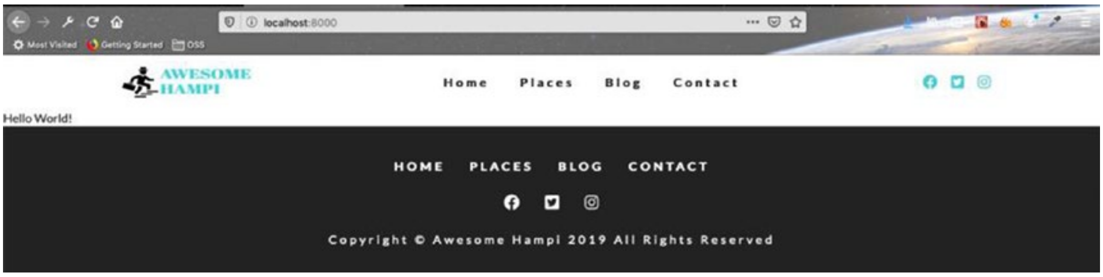


***Figure 3-11.** Footer on a desktop*

The footer is very responsive in the mobile view, as shown in Figure 3-12.

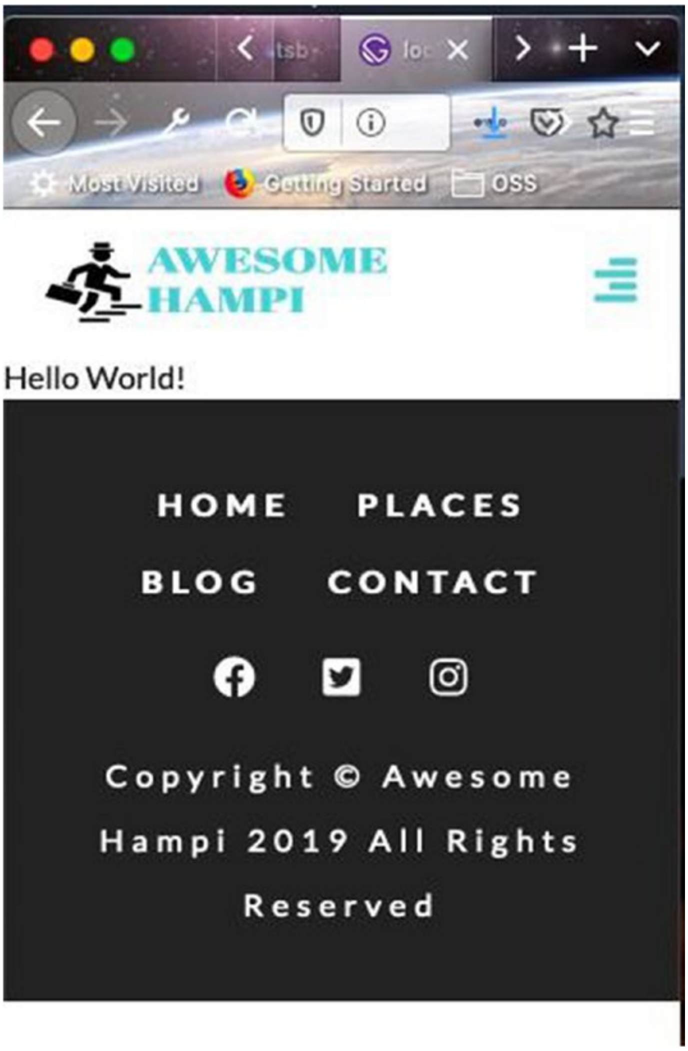


***Figure 3-12.** Footer in mobile view*

Creating the SimpleHero Component

In this section, we will create the `SimpleHero` component, which will show an image covering the whole home page. Create a new file called `SimpleHero.js` inside the `components` directory. The contents are shown in Listing 3-24.

Listing 3-24. The SimpleHero.js File

```
import React from "react"

const SimpleHero = ({ children }) => {
  return <header className="defaultHero">{children}</header>
}

export default SimpleHero
```

One of the main things to check in the `SimpleHero` component is the `defaultHero` class. It comes from the `layout.css` file. Here, we are using `min-height` to display it in the whole page, minus the header. We are also using `linear-gradient` here. Then we are using `display: flex`, which will display the center text. You need to add the Listing 3-25 contents to the `layout.css` file.

Listing 3-25. The layout.css File

```
.defaultHero {
  min-height: calc(100vh - 62px);
  background: linear-gradient(rgba(63, 208, 212, 0.7), rgba(0, 0, 0, 0.7)),
    url("../images/defaultBcg.jpeg") center/cover no-repeat;
  display: flex;
  justify-content: center;
  align-items: center;
}
```

We will then update the `index.js` page to show this component, using the code in Listing 3-26.

Listing 3-26. The Updated index.js File

```
import React from "react"
import Layout from "../components/Layout"
import SimpleHero from "../components/SimpleHero"

export default () => (
    <Layout>
        <SimpleHero />
    </Layout>
)
```

This file will show the huge image on the home page, as you can see in Figure 3-13.

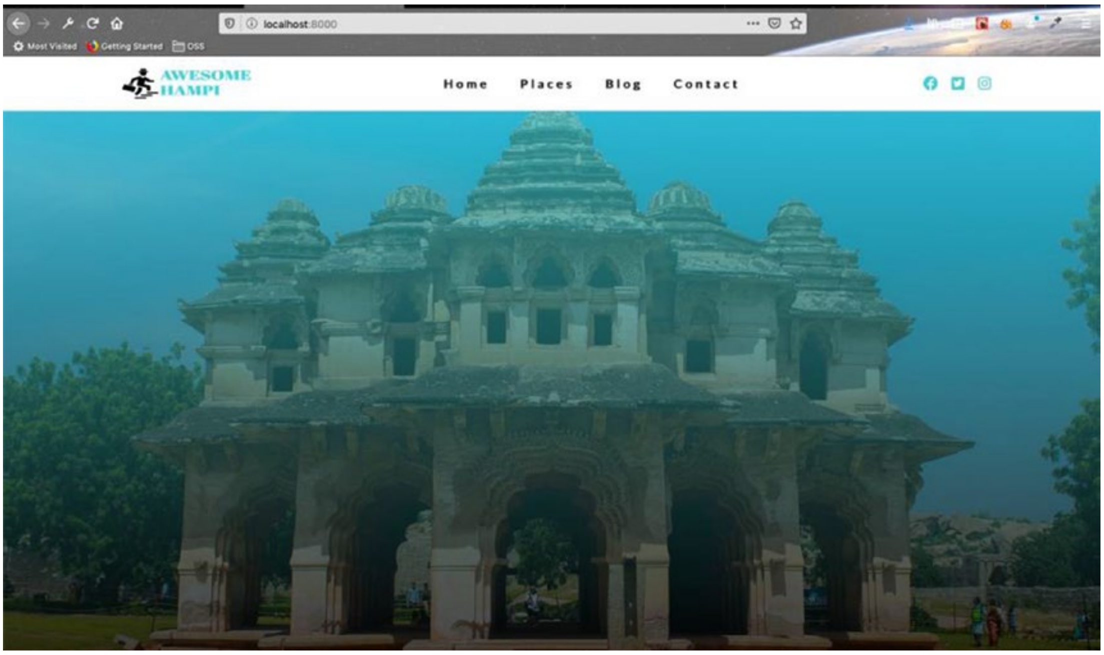


Figure 3-13. *Huge image on the home page*

After this, we will create a banner that will contain a large heading, then a paragraph and a large button to take us to the Places page. Go ahead and create `banner.module.css` inside the `css` folder and the code in Listing 3-27 to it.

Listing 3-27. The banner.module.css File

```
.banner {
  text-align: center;
  letter-spacing: var(--mainSpacing);
  color: var(--mainWhite);
}
.banner h1 {
  font-size: 3.3rem;
  text-transform: uppercase;
  margin-bottom: 2rem;
  padding: 0 1rem;
  letter-spacing: 6px;
}
.banner p {
  width: 85%;
  margin: 0 auto;
  margin-bottom: 2rem;
}
@media screen and (min-width: 768px) {
  .banner h1 {
    font-size: 4.5rem;
  }
  .banner p {
    width: 70%;
  }
}
```

Next, we will create `Banner.js` inside the `components` folder and it will contain the code in Listing 3-28. We can pass the `title`, `info`, and `children  props` to it.

Listing 3-28. The Banner.js File

```
import React from "react"
import styles from "../css/banner.module.css"
const Banner = ({ title, info, children }) => {
  return (
    <div className={styles.banner}>
      <h1>{title}</h1>
      <p>{info}</p>
      {children}
    </div>
  )
}

export default Banner
```

Now, we will add the banner component to the index.js file and pass the required props (title and info). Also, we are passing a Link as children. The updated code is marked in bold in Listing 3-29.

Listing 3-29. The index.js File

```
import React from "react"
import Layout from "../components/Layout"
import SimpleHero from "../components/SimpleHero"
import Banner from "../components/Banner"
import { Link } from "gatsby"

export default () => (
    <Layout>
        <SimpleHero>
            <Banner  title="Amazing Hampi"  info="Come and Explore Hampi,
            the city of ruins, which is a UNESCO World Heritage Site.">
                <Link to="/places" className="btn-white">explore places
                </Link>
            </Banner>
        </SimpleHero>
    </Layout>
)
```

This will show our Banner text inside the large image, as shown in Figure 3-14.

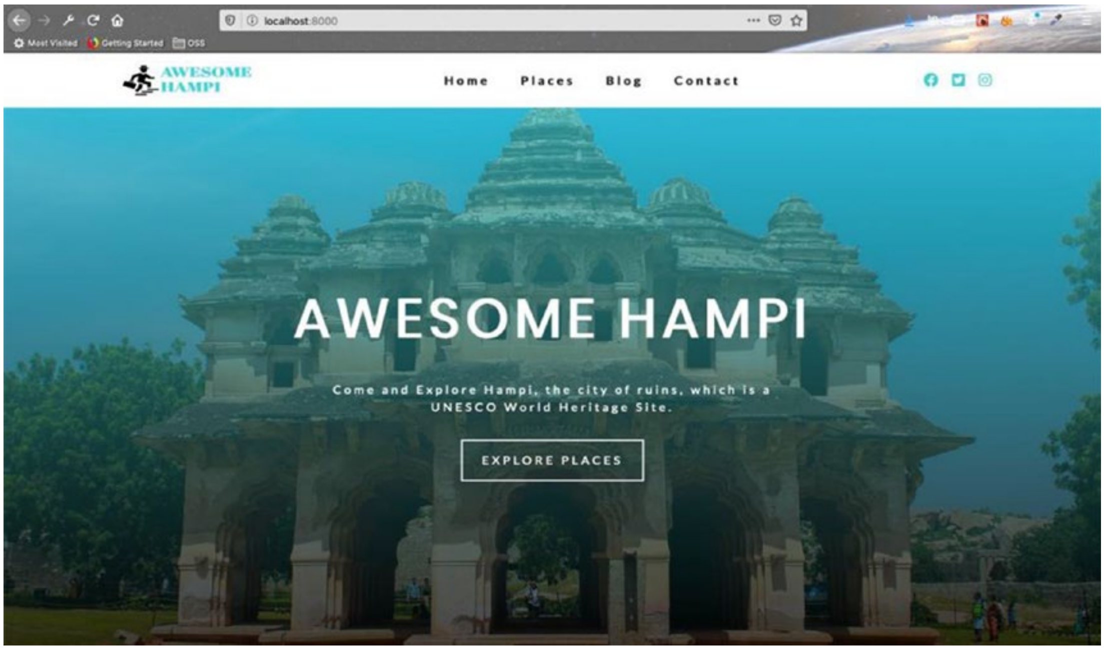

localhost:8000
AWESOME HAMPI
Home
Places
Blog
Contact
AWESOME HAMPI
Come and Explore Hampi, the city of ruins, which is a UNESCO World Heritage Site.
EXPLORE PLACES

Figure 3-14. *Banner text*

Next, we update our `Error` component. Go ahead and create `error.module.css` inside the css folder and add the code in Listing 3-30 to it.

Listing 3-30. The error.module.css File

```
.error {
  background: var(--primaryColor);
  min-height: calc(100vh - 62px);
  display: flex;
  justify-content: center;
  align-items: center;
}
```

Next, update the `404.js` page with the code in Listing 3-31. As you might have noticed, we are reusing the `Banner` component.

Listing 3-31. The 404.js File

```
import React from "react"
import Layout from "../components/Layout"
import styles from "../css/error.module.css"
import { Link } from "gatsby"
import Banner from "../components/Banner"

export default function error() {
return (
    <Layout>
        <header className={styles.error}>
        <Banner title="oops it's a dead end">
            <Link to="/" className="btn-white">
                back to home page
            </Link>
        </Banner>
        </header>
    </Layout>
     )
}
```

When we go to any nonexistent page, the text in Figure 3-15 will be displayed.

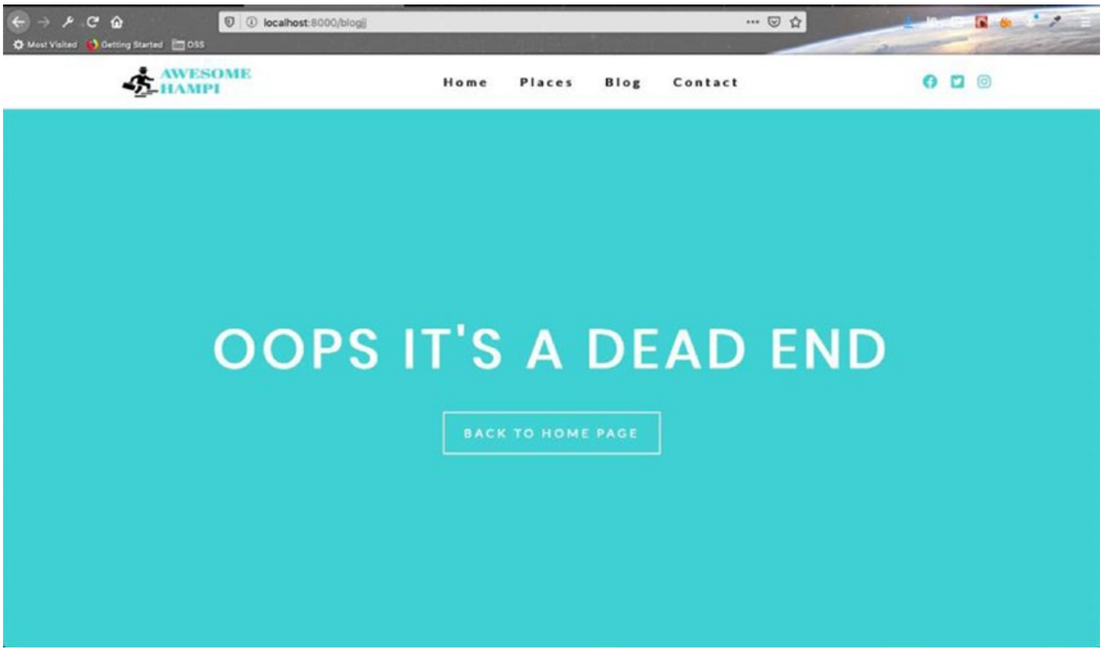


Figure 3-15. *The dead end page*

We will use `styled-components` in this post to style our project. Install the `gatsby-plugin-styled-components` plugin in the project.

As per the official document,[7] we need to install these packages first, with the command in Listing 3-32.

Listing 3-32. The npm install Command

```
npm install --save gatsby-plugin-styled-components styled-components babel-
plugin-styled-components
1
```

Head over to your project root directory and stop the `gatsby develop` and then run `npm install`.

Then head over to the `gatsby-config.js` file in the root directory and add the plugin in Listing 3-33.

Listing 3-33. The gatsby-config.js File

```
module.exports = {
       plugins: [`gatsby-plugin-styled-components`]
}
```

Next, head over to your terminal and run the `gatsby develop` again.

About Section

Next, we will create the About Hampi section in the home page. Add the `about.module.css` file to the `css` folder. You can get the content from Listing 3-34.

Listing 3-34. The about.module.css File

```
.about {
  padding: 4rem 0;
}
.about-center {
  width: 80vw;
```

[7] https://www.gatsbyjs.org/packages/gatsby-plugin-styled-components/

```
  margin: 0 auto;
}
.about-img {
  margin: 3rem 0;
}
.about-info {
  margin-top: 3rem;
}
.about-img {
  position: relative;
}

.about-img img {
  width: 100%;
  display: block;
  box-shadow: var(--lightShadow);
}
.about-img div {
  box-shadow: var(--lightShadow);
}
.about-info h4 {
  font-size: 1.9rem;
  text-transform: uppercase;
}

@media screen and (min-width: 768px) {
  .about-center {
    display: grid;
    grid-template-columns: 1fr 1fr;
    grid-column-gap: 3rem;
    align-items: center;
    margin-top: 3rem;
  }

  .about-img,
  .about-info {
    margin: 0;
  }
  .about-img img {
    max-height: 500px;
  }
  .img-container {
    max-height: 500px;
  }
  .about-info p {
    width: 80%;
  }
}
@media screen and (min-width: 992px) {
  .img-container::before {
    content: "";
    position: absolute;
    width: 100%;
    height: 100%;
    border: 3px solid var(--primaryColor);
    box-sizing: border-box;
    top: -16px;
    left: -16px;
    z-index: -1;
  }
}

@media screen and (min-width: 1200px) {
  .about-center {
    width: 95vw;
    max-width: 1170px;
  }
}
```

Next, we will create the About section inside the `components` folder. We will create another folder, called `Home`, inside it and the `About.js` file inside it. The contents are shown in Listing 3-35.

Listing 3-35. The About.js File

```
import React from "react"
import Title from "../Title"

const About = () => {
    return (
        <div>
            About Component
            <Title title="about" subtitle="hampi" />
        </div>
    )
}

export default About
```

We will have a general-purpose Title component inside About, in which we are using `styled-components`. We are styling the two words differently, which we had passed as props from the About component. Create a file called `Title.js` inside the `components` folder and put the code in Listing 3-36 into it.

Listing 3-36. The Title.js File

```
import React from "react"
import styled from "styled-components"
const Title = ({ title, subtitle }) => {
  return (
    <TitleWrapper>
      <h4>
        <span className="title">{title}</span>
        <span>{subtitle}</span>
      </h4>
    </TitleWrapper>
  )
}

const TitleWrapper = styled.div`
  text-transform: uppercase;
  font-size: 2.3rem;
  margin-bottom: 2rem;
  h4 {
    text-align: center;
    letter-spacing: 7px;
    color: var(--primaryColor);
  }
  .title {
    color: var(--mainBlack);
  }
  span {
    display: block;
  }
  @media (min-width: 576px) {
    span {
      display: inline-block;
      margin: 0 0.35rem;
    }
  }
`

export default Title
```

Now, let's show the About component in `index.js`. The updated code is marked in bold in Listing 3-37.

Listing 3-37. The index.js File

```
import React from "react"
import Layout from "../components/Layout"
import SimpleHero from "../components/SimpleHero"
import Banner from "../components/Banner"
import About from "../components/Home/About"
import { Link } from "gatsby"

export default () => (
    <Layout>
        <SimpleHero>
            <Banner
```

```
                title="Amazing Hampi"
                info="Come and Explore Hampi, the city of ruins, which is a
                UNESCO World Heritage Site."
            >
                <Link to="/places" className="btn-white">explore places
                </Link>
            </Banner>
        </SimpleHero>
        <About />
    </Layout>
)
```

It will show the About component, which contains the Title component in our home page, as shown in Figure 3-16.

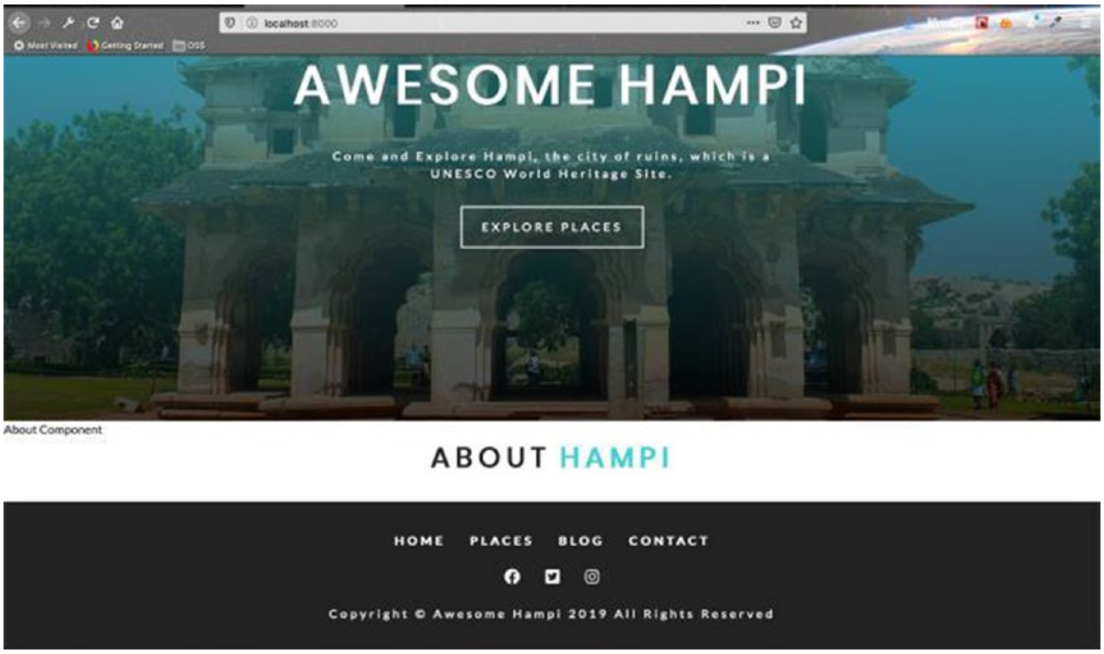

localhost:8000
AWESOME HAMPI
Come and Explore Hampi, the city of ruins, which is a UNESCO World Heritage Site.
EXPLORE PLACES
About Component
ABOUT HAMPI
HOME
PLACES
BLOG
CONTACT
Copyright © Awesome Hampi 2019 All Rights Reserved

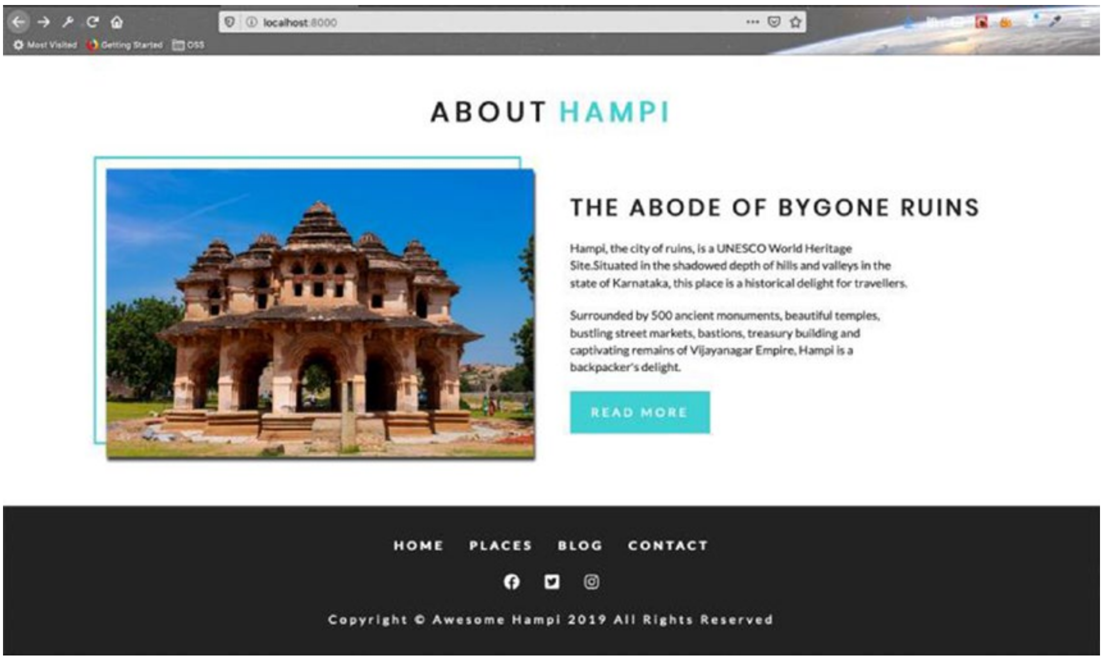
Figure 3-16. *Showing the title only*

Next, we will complete our About component, in which we will show an image, a subtitle, two paragraphs, and a button. Update the `About.js` file with Listing 3-38.

Listing 3-38. The About.js File

```
import React from "react"
import Title from "../Title"
import styles from "../../css/about.module.css"
import img from "../../images/defaultBcg.jpeg"

const About = () => {
  return (
    <section className={styles.about}>
      <Title title="about" subtitle="hampi" />
      <div className={styles.aboutCenter}>
        <article className={styles.aboutImg}>
          <div className={styles.imgContainer}>
            <img src={img} alt="about company" />
          </div>
        </article>
        <article className={styles.aboutInfo}>
          <h4>The abode of bygone ruins</h4>
          <p>
          Hampi, the city of ruins, is a UNESCO World Heritage Site.
          Situated in the shadowed depth of hills and valleys in the state
          of Karnataka, this place is a historical delight for travellers.
          </p>
          <p>
          Surrounded by 500 ancient monuments, beautiful temples, bustling
          street markets, bastions, treasury building and captivating
          remains of Vijayanagar Empire, Hampi is a backpacker's delight.
          </p>
          <a href="https://en.wikipedia.org/wiki/Hampi" className="btn-
          primary" target="_blank" rel="noopener noreferrer">
            read more
          </a>
        </article>
      </div>
    </section>
  )
}

export default About
```

It will show our beautiful About Hampi section, as shown in Figure 3-17.



Hampi, the city of ruins, is a UNESCO World Heritage Site.Situated in the shadowed depth of hills and valleys in the state of Karnataka, this place is a historical delight for travellers.

Surrounded by 500 ancient monuments, beautiful temples, bustling street markets, bastions, treasury building and captivating remains of Vijayanagar Empire, Hampi is a backpacker's delight.

Copyright © Awesome Hampi 2019 All Rights Reserved

Figure 3-17. *The About Hampi section*

Creating a Hot Tips Section

Next, we will create the Hot Tips section in the home page. Add the `tips.module.css` file to the `css` folder. The contents are shown in Listing 3-39.

Listing 3-39. The tips.module.css File

```
.tips {
  background: var(--mainGrey);
  padding: 4rem 0;
}
```

```
.center {
  width: 80vw;
  margin: 0 auto;
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
  grid-column-gap: 2rem;
}
.tip {
  margin: 2rem 0;
  text-align: center;
}
.tip span {
  background: var(--primaryColor);
  padding: 0.5rem;
  display: inline-block;
  font-size: 2rem;
  margin-bottom: 1.5rem;
}
.tip h4 {
  text-transform: uppercase;
}
```

Then add `tips.js` inside the `constants` folder and add the content from Listing 3-40.

Listing 3-40. The tips.js File

```
import React from "react"
import { FaWallet, FaCamera, FaSocks } from "react-icons/fa"

export default [
  {
    icon: <FaWallet />,
    title: "saving money",
    text: "Travel by train from Bangalore, instead of taxi or bus.",
  },
  {
    icon: <FaCamera />,
    title: "top attractions",
    text: "Top attractions are Vittala Temple, Virupaksha Temple,
    Achyutaraya Temple.",
  },
  {
    icon: <FaSocks />,
    title: "amazing comfort",
    text: "Stay in Oyo rooms, instead of Airbnb or some travel site.",
  },
]
```

Now, we will add a `Tips.js` file inside the `components->Home` folder. As you can see, we are reusing the Title component here. Also, we are mapping through the `tips` constant to display all its items. The contents are shown in Listing 3-41.

Listing 3-41. The Tips.js File

```
import React from "react"
import Title from "../Title"
import styles from "../../css/tips.module.css"
import tips from "../../constants/tips"

const Tips = () => {
  return (
    <section className={styles.tips}>
      <Title title="hot" subtitle="tips" />
      <div className={styles.center}>
        {tips.map((item, index) => {
          return (
            <article key={index} className={styles.tip}>
              <span>{item.icon}</span>
              <h4>{item.title}</h4>
              <p>{item.text}</p>
            </article>
          )
        })}
      </div>
    </section>
  )
}

export default Tips
```

Let's show the Tips component in the index.js file. The updated code is marked in bold in Listing 3-42.

Listing 3-42. The index.js File

```
import React from "react"
import Layout from "../components/Layout"
import SimpleHero from "../components/SimpleHero"
import Banner from "../components/Banner"
import About from "../components/Home/About"
import Tips from "../components/Home/Tips"
import { Link } from "gatsby"

export default () => (
    <Layout>
        <SimpleHero>
            <Banner
                title="Amazing Hampi"
                info="Come and Explore Hampi, the city of ruins, which is a
                UNESCO World Heritage Site."
            >
                <Link to="/places" className="btn-white">explore places
                </Link>
            </Banner>
        </SimpleHero>
        <About />
        <Tips />
    </Layout>
)
```

This will show the Hot Tips section on the home page, as shown in Figure 3-18.

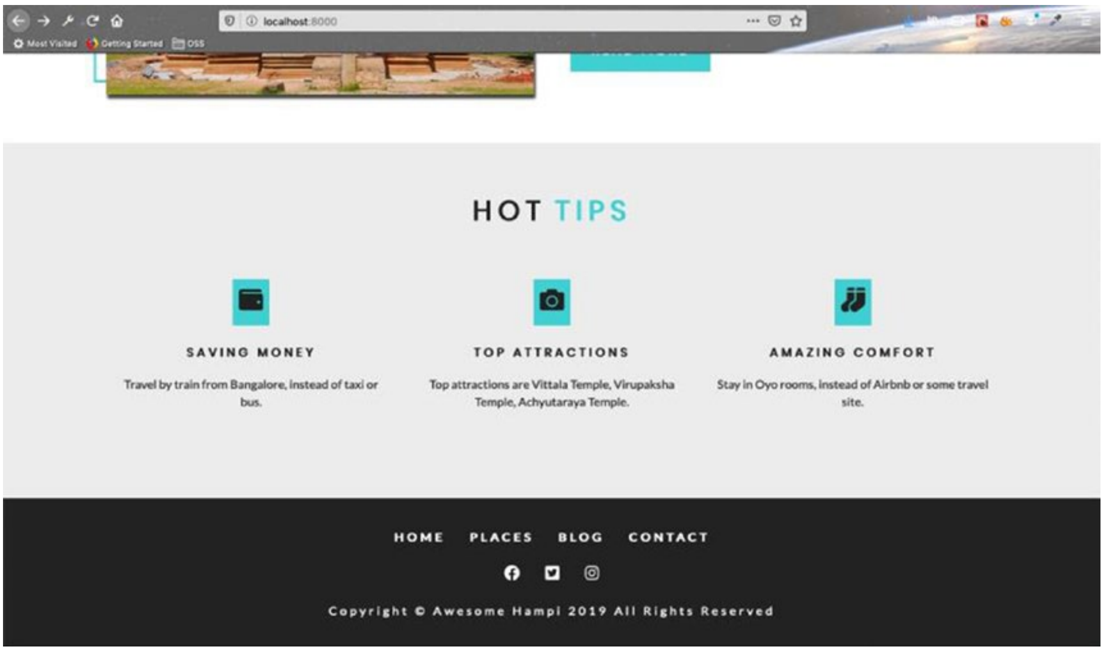

localhost:8000
HOT TIPS
SAVING MONEY
Travel by train from Bangalore, instead of taxi or bus.
TOP ATTRACTIONS
Top attractions are Vittala Temple, Virupaksha Temple, Achyutaraya Temple.
AMAZING COMFORT
Stay in Oyo rooms, instead of Airbnb or some travel site.
HOME
PLACES
BLOG
CONTACT
Copyright © Awesome Hampi 2019 All Rights Reserved

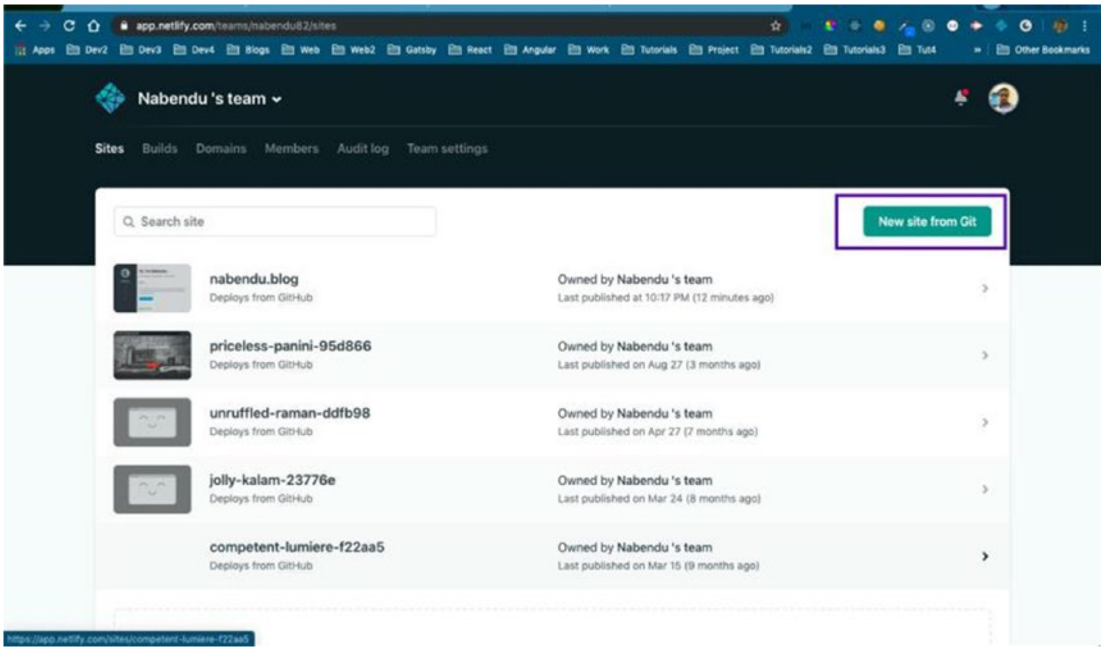
Figure 3-18. *The Hot Tips section*

Creating a Deployment Site

In this section, we will learn how to do continuous deployment using Netlify. It is a great service to host your Gatsby project. Since this project is already on GitHub,[8] I simply have to log in to my Netlify and link it to host the site.

Log in to your Netlify[9] account or create one. Since I already have a Netlify account and have many sites hosted on it, my login screen is shown in Figure 3-19. Click the New Site from Git button, shown in Figure 3-19.

[8]https://github.com/nabendu82/gatsbyTourism
[9]https://www.netlify.com/



nabendu.blog
Deploys from GitHub
Owned by Nabendu 's team
Last published at 10:17 PM (12 minutes ago)

priceless-panini-95d866
Deploys from GitHub
Owned by Nabendu 's team
Last published on Aug 27 (3 months ago)

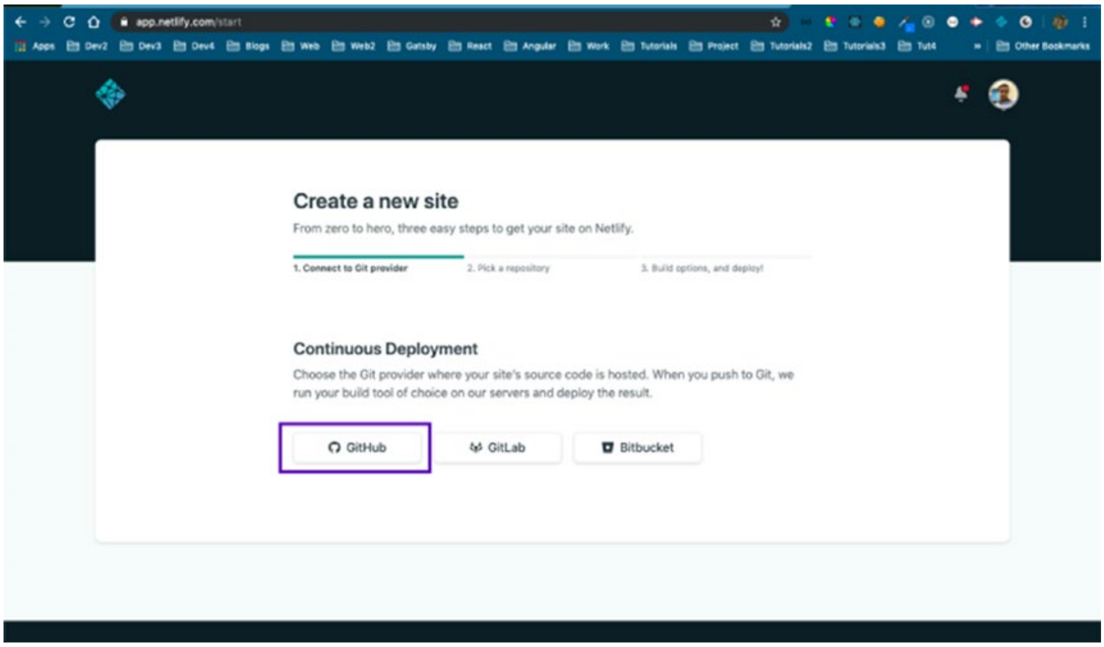
unruffled-raman-ddfb98
Deploys from GitHub
Owned by Nabendu 's team
Last published on Apr 27 (7 months ago)

jolly-kalam-23776e
Deploys from GitHub
Owned by Nabendu 's team
Last published on Mar 24 (8 months ago)

competent-lumiere-f22aa5
Deploys from GitHub
Owned by Nabendu 's team
Last published on Mar 15 (9 months ago)

Figure 3-19. *Netlify*

Since, my project is hosted on Github, I will click the same. It is shown in Figure 3-20.



Create a new site

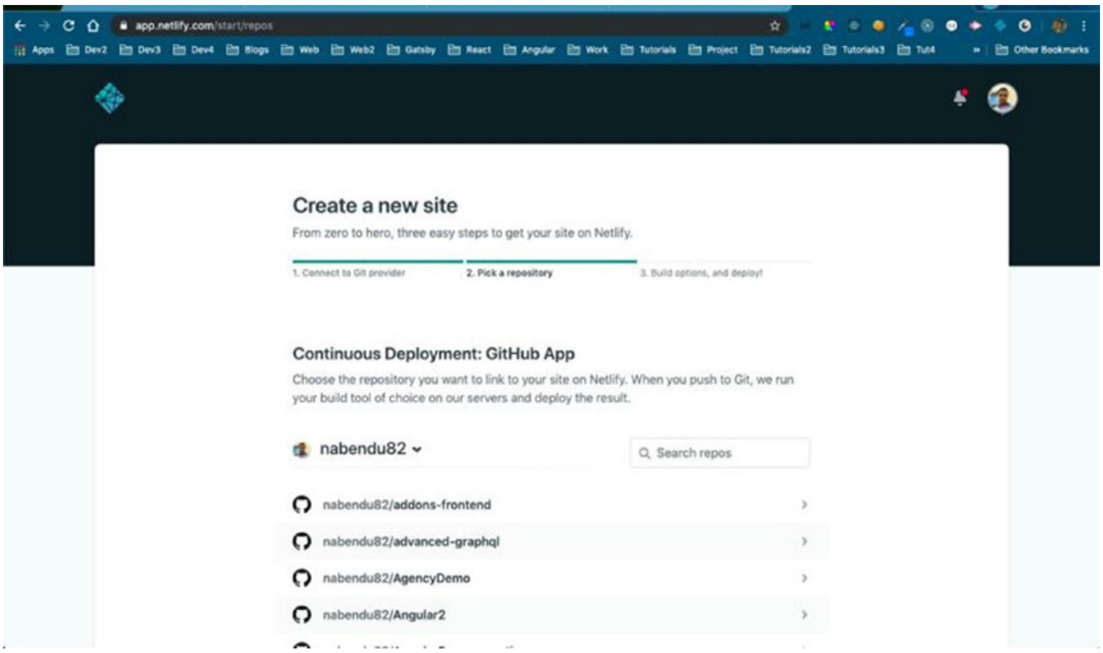
From zero to hero, three easy steps to get your site on Netlify.

Continuous Deployment

Choose the Git provider where your site's source code is hosted. When you push to Git, we run your build tool of choice on our servers and deploy the result.

Figure 3-20. *The GitHub site*

It will open a popup window and ask you to authorize with your GitHub credentials for the first time. Since I was already authorized, it took me directly to this screen, which shows all my GitHub repos, shown in Figure 3-21.



Create a new site

From zero to hero, three easy steps to get your site on Netlify.

1. Connect to Git provider
2. Pick a repository
3. Build options, and deploy!

Continuous Deployment: GitHub App

Choose the repository you want to link to your site on Netlify. When you push to Git, we run your build tool of choice on our servers and deploy the result.

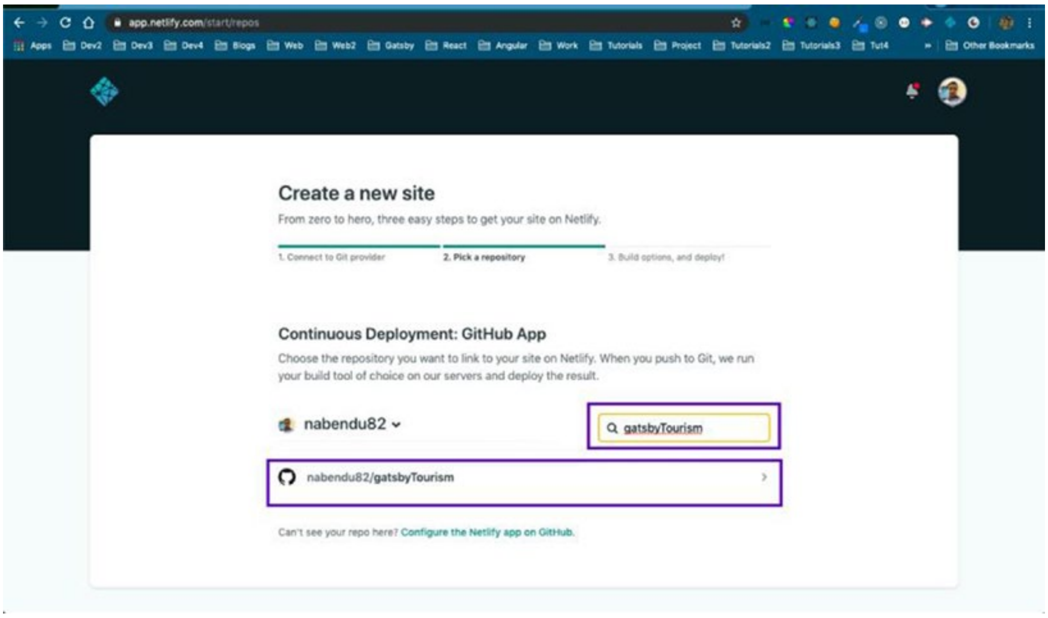
nabendu82

Search repos

nabendu82/addons-frontend
nabendu82/advanced-graphql
nabendu82/AgencyDemo
nabendu82/Angular2

Figure 3-21. *GitHub repos*

I have many repos so I have to search for the repo and then click it, as shown in Figure 3-22.


Create a new site

From zero to hero, three easy steps to get your site on Netlify.

1. Connect to Git provider
2. Pick a repository
3. Build options, and deploy!

Continuous Deployment: GitHub App

Choose the repository you want to link to your site on Netlify. When you push to Git, we run your build tool of choice on our servers and deploy the result.

nabendu82

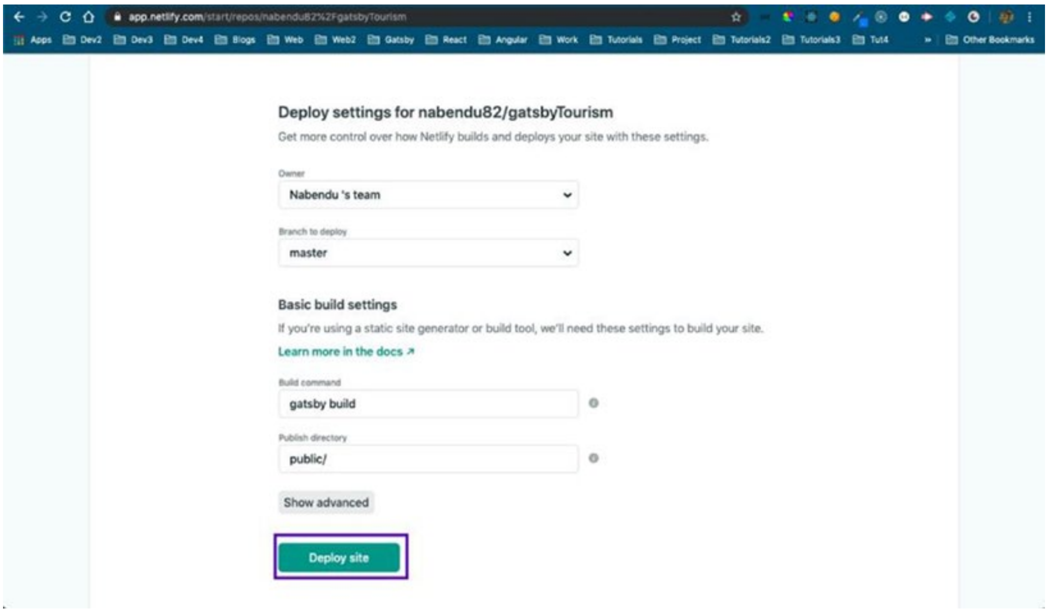
gatsbyTourism

nabendu82/gatsbyTourism

Can't see your repo here? Configure the Netlify app on GitHub.

Figure 3-22. *Tourism repo*

The next screen will show all the details of the repo. It will even run the `gatsby build` command for us, once we click the Deploy Site button. See Figure 3-23.


Deploy settings for nabendu82/gatsbyTourism

Get more control over how Netlify builds and deploys your site with these settings.

Owner: Nabendu 's team

Branch to deploy: master

Basic build settings

If you're using a static site generator or build tool, we'll need these settings to build your site.

Learn more in the docs

Build command: gatsby build

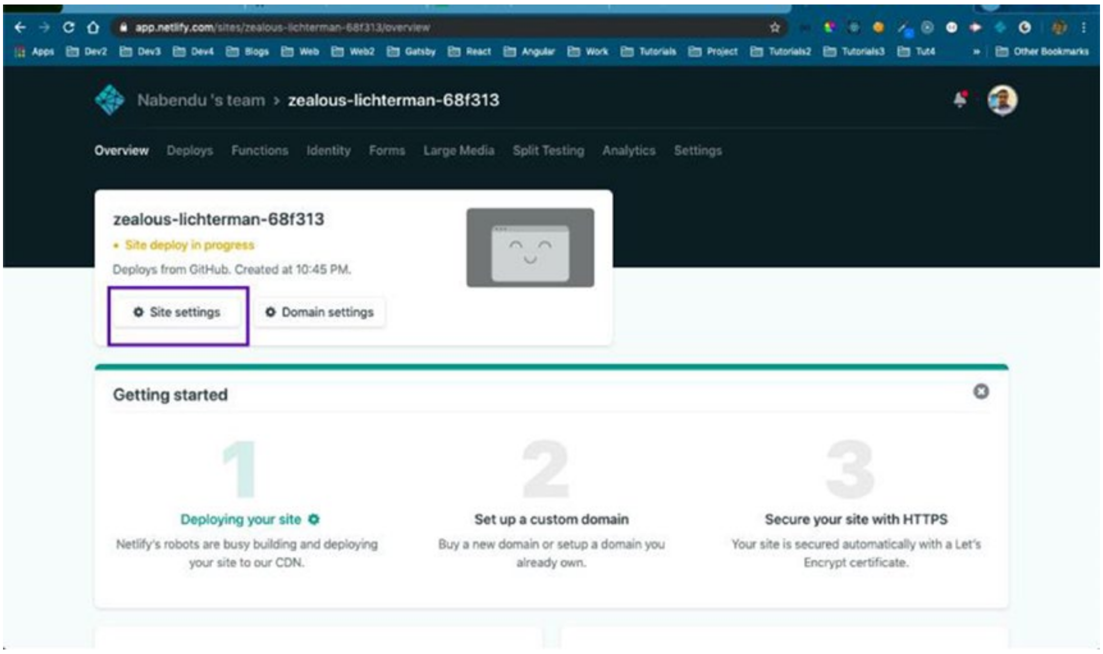
Publish directory: public/

Show advanced

Deploy site

Figure 3-23. *The gatsby build command*

The next page will show the random site name in which Netlify is deploying. We can click the Site Settings button to change the site name to a more suitable name, as shown in Figure 3-24.



zealous-lichterman-68f313
• Site deploy in progress
Deploys from GitHub. Created at 10:45 PM.

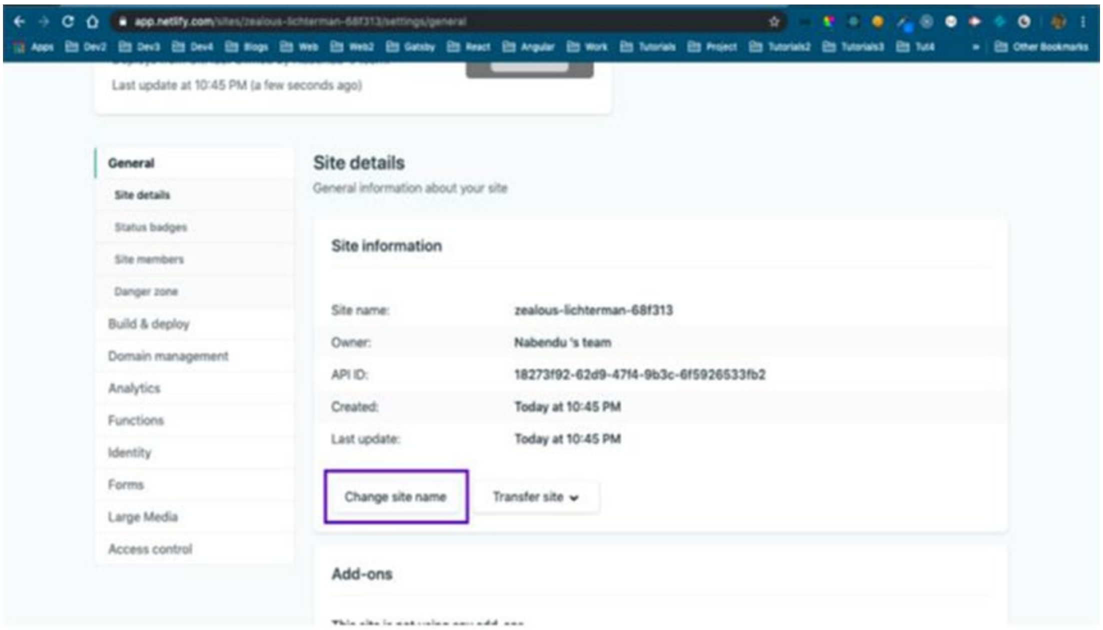
1 Deploying your site
Netlify's robots are busy building and deploying your site to our CDN.

2 Set up a custom domain
Buy a new domain or setup a domain you already own.

3 Secure your site with HTTPS
Your site is secured automatically with a Let's Encrypt certificate.

Figure 3-24. *Random site*

Click the Change Site Name button, as shown in Figure 3-25.



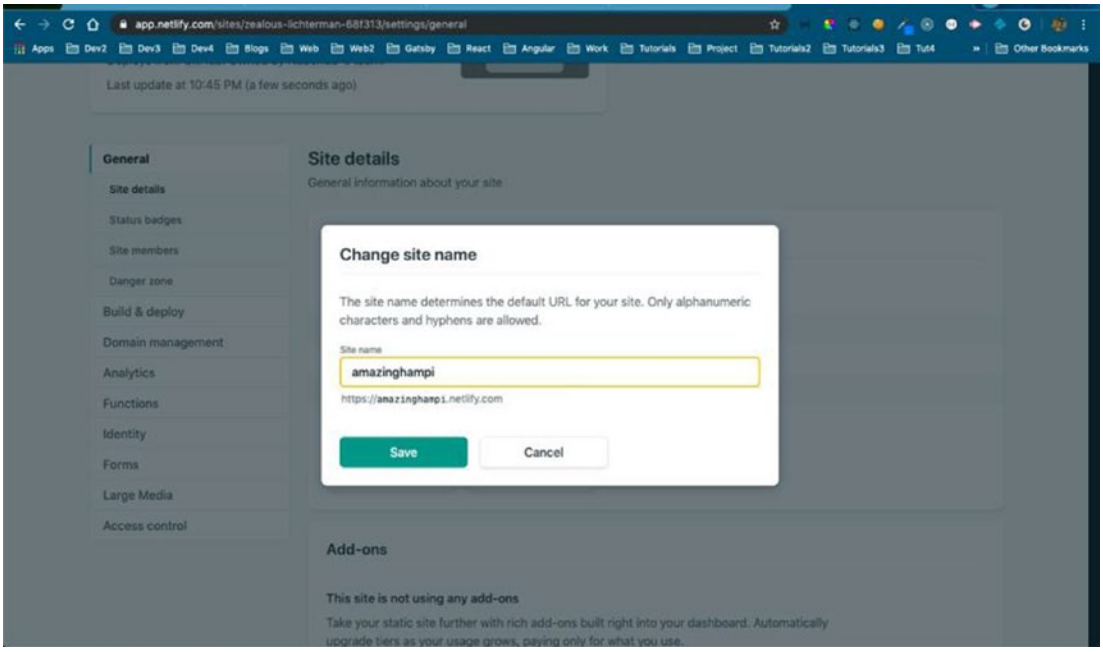
Site details
General information about your site

Site information

Site name:	zealous-lichterman-68f313
Owner:	Nabendu 's team
API ID:	18273f92-62d9-4714-9b3c-6f5926533fb2
Created:	Today at 10:45 PM
Last update:	Today at 10:45 PM

Figure 3-25. *Change the site name*

Provide any good name. I use `amazinghampi` here, but `netlify.com` will be added to it, as shown in Figure 3-26. We will remove this at the end of the series, by purchasing a domain name and changing it.



Change site name

The site name determines the default URL for your site. Only alphanumeric characters and hyphens are allowed.

Site name: amazinghampi

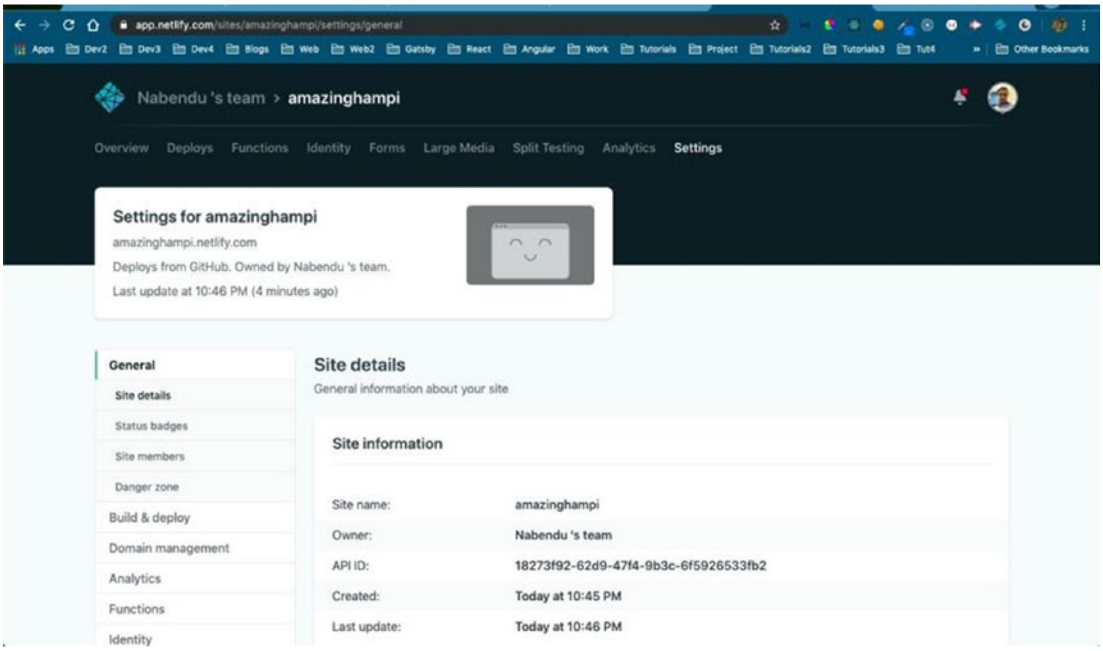
https://amazinghampi.netlify.com

This site is not using any add-ons

Take your static site further with rich add-ons built right into your dashboard. Automatically upgrade tiers as your usage grows, paying only for what you use.

Figure 3-26. ***Save the site with a new name***

Once you click the Save button, the site will be deployed, as shown in Figure 3-27.



Settings for amazinghampi

amazinghampi.netlify.com

Deploys from GitHub. Owned by Nabendu 's team.

Last update at 10:46 PM (4 minutes ago)

Site details

General information about your site

Site information

Site name:	amazinghampi
Owner:	Nabendu 's team
API ID:	18273f92-62d9-47f4-9b3c-6f5926533fb2
Created:	Today at 10:45 PM
Last update:	Today at 10:46 PM

Figure 3-27. ***The site has been deployed***

Click the deployed site. It is now on the Internet, as shown in Figure 3-28. Now, whenever you push some new code in GitHub, Netlify will directly update the site.

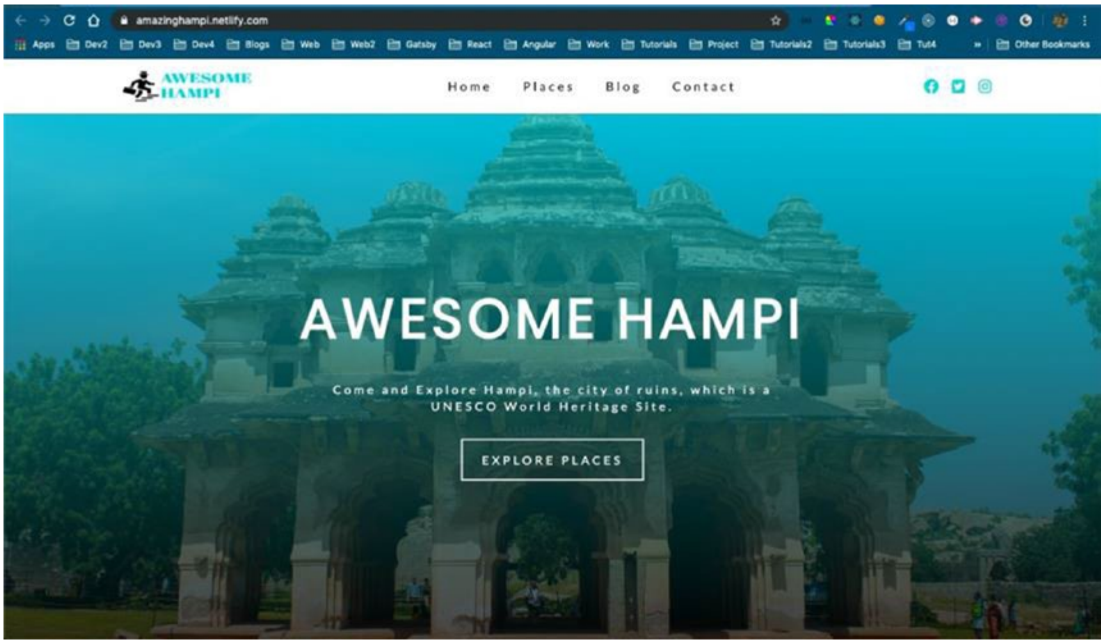


Figure 3-28. *Awesome Hampi*

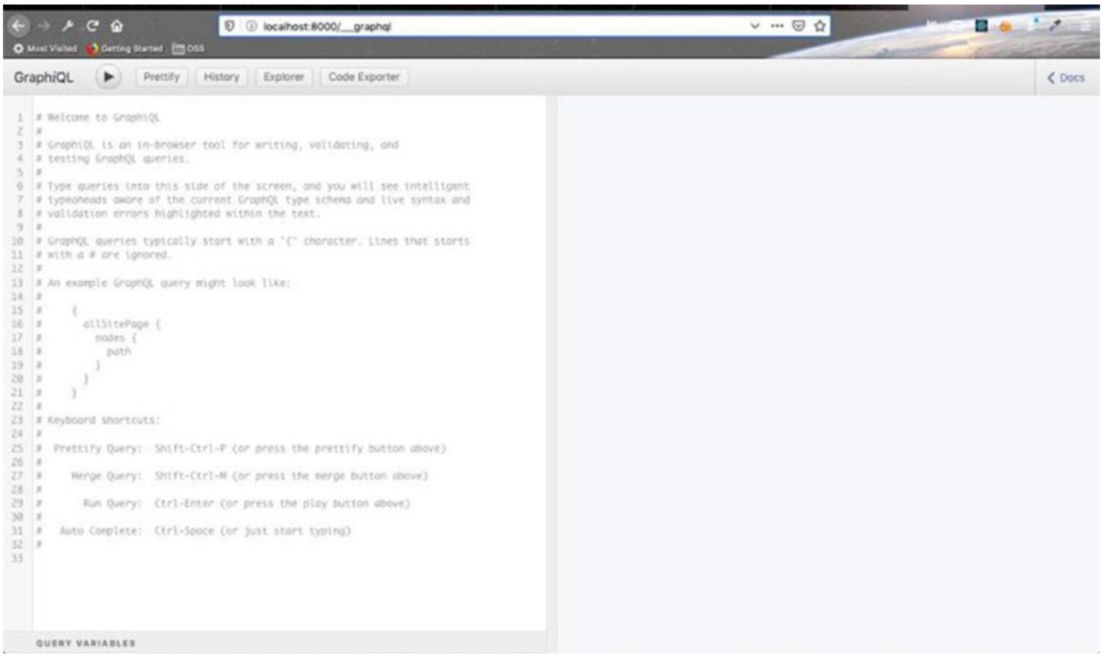
Image Optimization

Up to this moment, we used React to do things in Gatsby. But the real power of Gatsby comes from GraphQL and with using different plugins that use GraphQL.

When we run any Gatsby project with `gatsby develop` and, when it compiles successfully, we get the link of GraphiQL, which is `http://localhost:8000/___graphql`.

It is the playground in which we can test all our GraphQL queries before implementing them in our code. We can open it[10] in the web browser and it will be shown, as in Figure 3-29.

[10] `http://localhost:8000/graphql`



```
# Welcome to GraphiQL
#
# GraphiQL is an in-browser tool for writing, validating, and
# testing GraphQL queries.
#
# Type queries into this side of the screen, and you will see intelligent
# typeaheads aware of the current GraphQL type schema and live syntax and
# validation errors highlighted within the text.
#
# GraphQL queries typically start with a "{" character. Lines that starts
# with a # are ignored.
#
# An example GraphQL query might look like:
#
#     {
#       allSitePage {
#         nodes {
#           path
#         }
#       }
#     }
#
# Keyboard shortcuts:
#
#  Prettify Query:  Shift-Ctrl-P (or press the prettify button above)
#
#     Merge Query:  Shift-Ctrl-M (or press the merge button above)
#
#       Run Query:  Ctrl-Enter (or press the play button above)
#
#   Auto Complete:  Ctrl-Space (or just start typing)
#

```

Figure 3-29. *Open it in the browser*

Here, we will test our GraphQL queries before using them by `StaticQuery` or `PageQuery` in the Gatsby code. We will be using image optimization in our project through GraphQL, as that is one of the main reasons that sites load slowly.

But before we do that, we need to install some plugins and do some configurations.

We need to install `gatsby-source-filesystem` first. This plugin is for using data in your Gatsby application from a local filesystem. In our case, this will be images. The docs can be found here[11]. As per the docs, we need to `npm install` the plugin in our project directory.

Head over to the `project` directory and stop any `gatsby develop`, if any is running. Run the command in Listing 3-43 to install the plugin.

Listing 3-43. The npm install Command

```
npm install --save gatsby-source-filesystem
```

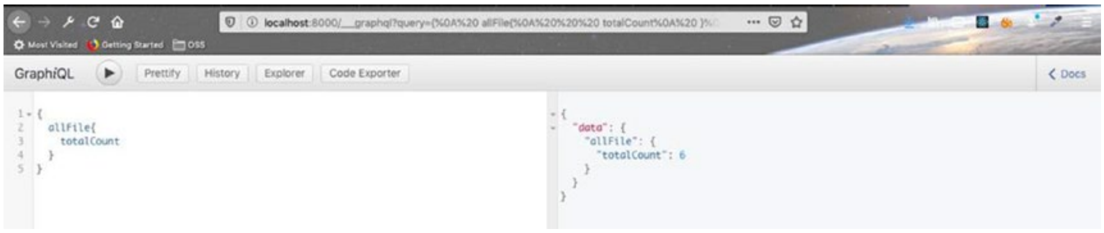
Next, we will update our `gatsby-config.js` file to use this plugin to get our `images` folder path. The updated code is marked in bold in Listing 3-44.

[11] https://www.gatsbyjs.org/packages/gatsby-source-filesystem/

Listing 3-44. The gatsby-config.js File

```
module.exports = {
            plugins: [
           {
      resolve: `gatsby-source-filesystem`,
      options: {
          name: `images`,
          path: `${__dirname}/src/images/`,
         },
    },
             `gatsby-plugin-styled-components`
               ]
}
```

Let's quickly verify whether the plugin is working properly. Head over to GraphiQL and refresh the browser. After that, run the following query. It will give us the number of files in the `images` folder. We have six files in the `images` folder, so it will give the output as `6`, as shown in Figure 3-30.

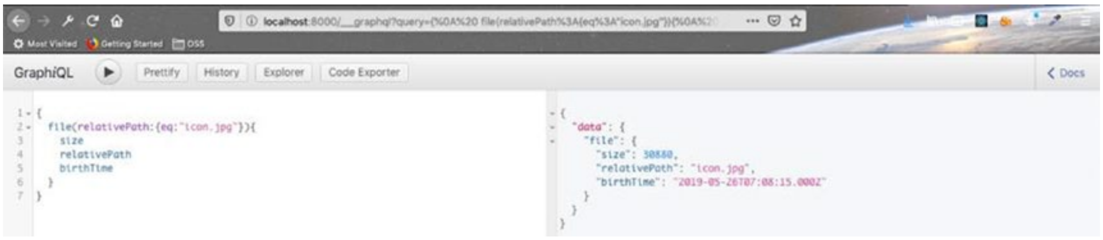


```
{
  allFile{
    totalCount
  }
}
```

```
{
  "data": {
    "allFile": {
      "totalCount": 6
    }
  }
}
```

Figure 3-30. *GraphiQL*

The `gatsby-source-filesystem` file also provides us with another query called `file`, which provides the details of any file present in the `images` folder, as shown in Figure 3-31.


localhost:8000/___graphql?query={%0A%20 file(relativePath%3A{eq%3A"icon.jpg"}){%0A%2
Most Visited
Getting Started
OSS
GraphiQL
Prettify
History
Explorer
Code Exporter
Docs

```
{
  file(relativePath:{eq:"icon.jpg"}){
    size
    relativePath
    birthTime
  }
}
```

```
{
  "data": {
    "file": {
      "size": 30880,
      "relativePath": "icon.jpg",
      "birthTime": "2019-05-26T07:08:15.000Z"
    }
  }
}
```

Figure 3-31. *The file query*

Next, we will install the `gatsby-image`[12] plugin. This is the plugin that we will use in our project to speed up image loading.

As per the docs[13], we have to do two `npm installs`. The command is shown in Listing 3-45.

Listing 3-45. The npm install Command

```
npm install --save gatsby-image
npm install --save gatsby-transformer-sharp gatsby-plugin-sharp
```

After the installs are complete, update the **gatsby-config.js** file. The updated code is marked in bold in Listing 3-46.

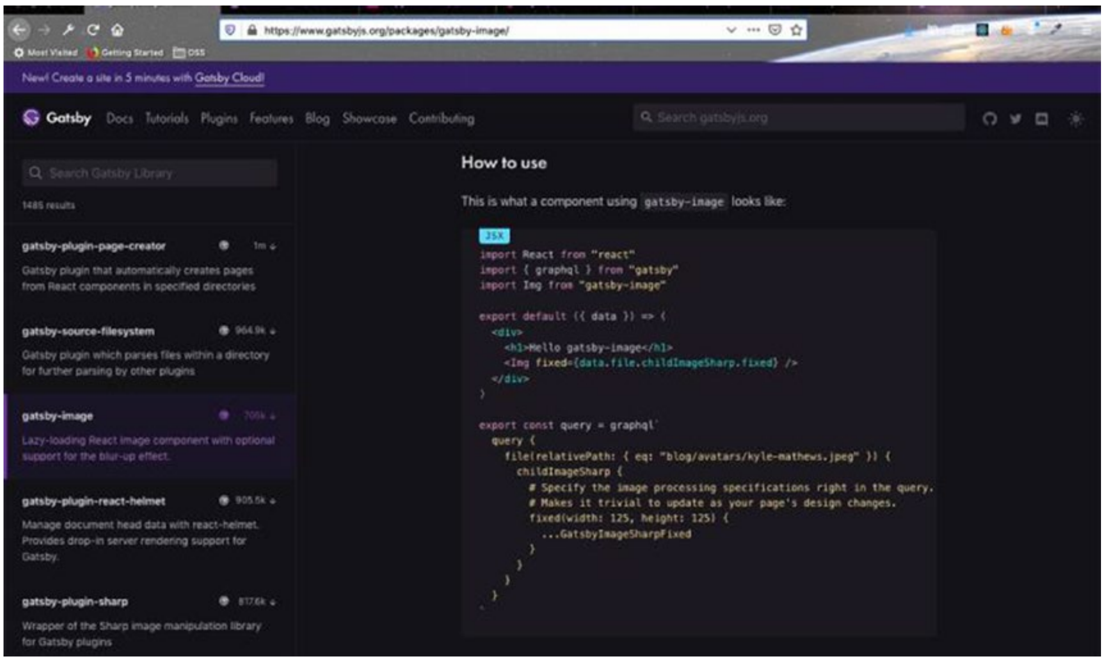
Listing 3-46. The gatsby-config.js File

```
module.exports = {
            plugins: [
            {
      resolve: `gatsby-source-filesystem`,
      options: {
        name: `images`,
        path: `${__dirname}/src/images/`,
      },
    },
```

[12] https://www.gatsbyjs.org/packages/gatsby-image/
[13] https://www.gatsbyjs.org/packages/gatsby-image/

```
                    `gatsby-plugin-styled-components`,
                    `gatsby-transformer-sharp`,
                    `gatsby-plugin-sharp`
            ]
}
```

Next, start the project by running `gatsby develop`. As per the documentation, we need to use the `gatsby-image` plugin shown in Figure 3-32. We can have two types of images—`fixed` or `fluid`.



gatsby-plugin-page-creator 1m
Gatsby plugin that automatically creates pages from React components in specified directories

gatsby-source-filesystem 964.9k
Gatsby plugin which parses files within a directory for further parsing by other plugins

gatsby-image 706k
Lazy-loading React image component with optional support for the blur-up effect.

gatsby-plugin-react-helmet 905.5k
Manage document head data with react-helmet. Provides drop-in server rendering support for Gatsby.

gatsby-plugin-sharp 817.6k
Wrapper of the Sharp image manipulation library for Gatsby plugins

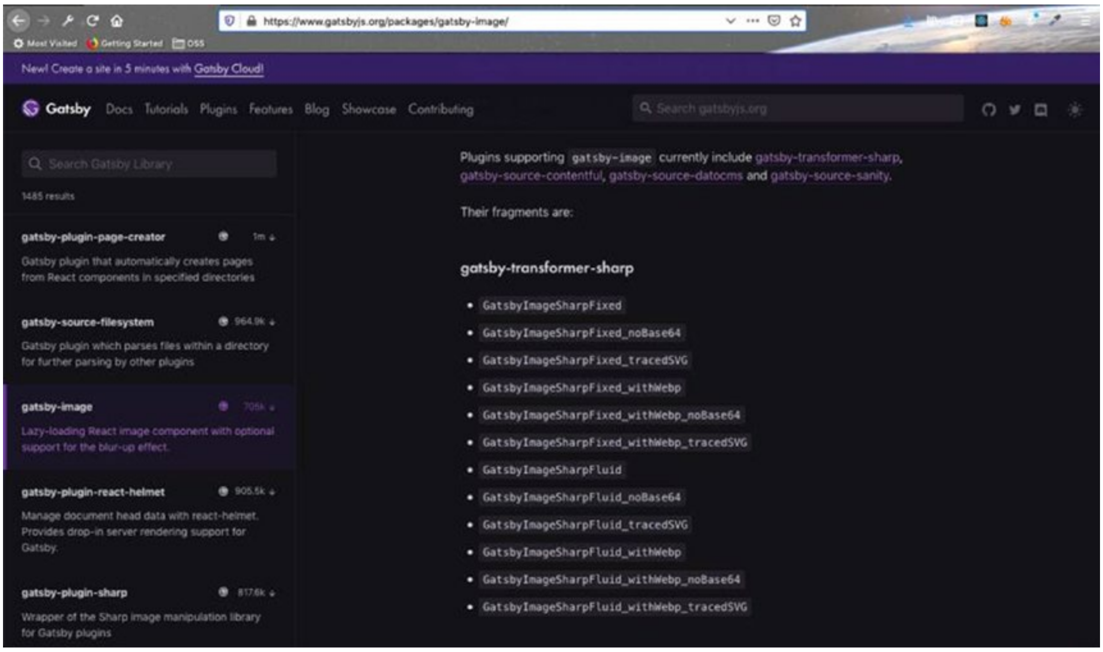
How to use

This is what a component using `gatsby-image` looks like:

```jsx
import React from "react"
import { graphql } from "gatsby"
import Img from "gatsby-image"

export default ({ data }) => (
  <div>
    <h1>Hello gatsby-image</h1>
    <Img fixed={data.file.childImageSharp.fixed} />
  </div>
)

export const query = graphql`
  query {
    file(relativePath: { eq: "blog/avatars/kyle-mathews.jpeg" }) {
      childImageSharp {
        # Specify the image processing specifications right in the query.
        # Makes it trivial to update as your page's design changes.
        fixed(width: 125, height: 125) {
          ...GatsbyImageSharpFixed
        }
      }
    }
  }
`
```

Figure 3-32. *How to use the component*

We can use the fragments, shown in Figure 3-33, in place of `GatsbyImageSharpFixed`.



Plugins supporting `gatsby-image` currently include gatsby-transformer-sharp, gatsby-source-contentful, gatsby-source-datocms and gatsby-source-sanity.

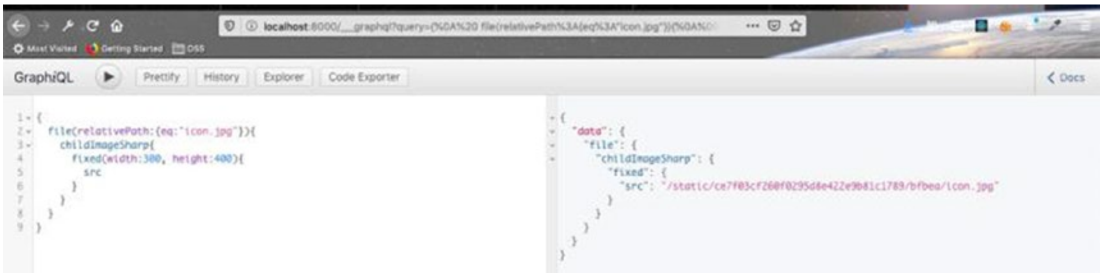
Their fragments are:

gatsby-transformer-sharp

- `GatsbyImageSharpFixed`
- `GatsbyImageSharpFixed_noBase64`
- `GatsbyImageSharpFixed_tracedSVG`
- `GatsbyImageSharpFixed_withWebp`
- `GatsbyImageSharpFixed_withWebp_noBase64`
- `GatsbyImageSharpFixed_withWebp_tracedSVG`
- `GatsbyImageSharpFluid`
- `GatsbyImageSharpFluid_noBase64`
- `GatsbyImageSharpFluid_tracedSVG`
- `GatsbyImageSharpFluid_withWebp`
- `GatsbyImageSharpFluid_withWebp_noBase64`
- `GatsbyImageSharpFluid_withWebp_tracedSVG`

Figure 3-33. *These fragments can be used*

We will use this later in our project. For now, we can check the query on the GraphiQL, as shown in Figure 3-34.

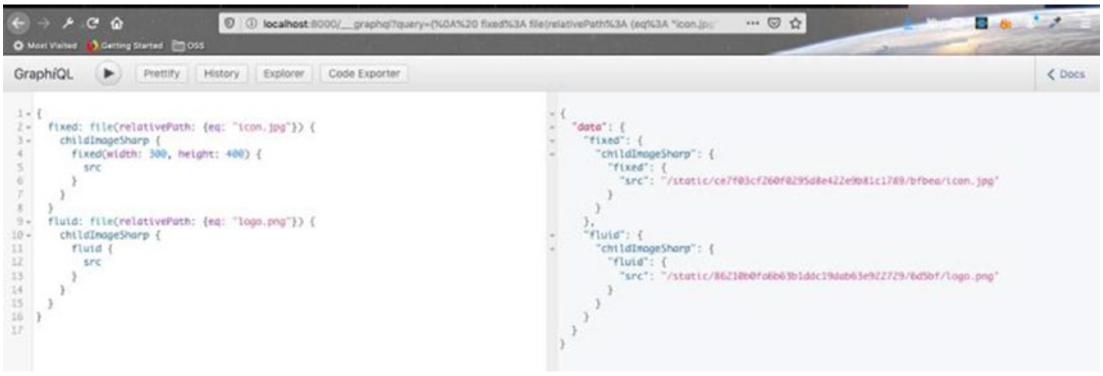


```
{
  file(relativePath:{eq:"icon.jpg"}){
    childImageSharp{
      fixed(width:300, height:400){
        src
      }
    }
  }
}
```

```
{
  "data": {
    "file": {
      "childImageSharp": {
        "fixed": {
          "src": "/static/ce7f03cf260f0295d6e422e9b81c1789/bfbea/icon.jpg"
        }
      }
    }
  }
}
```

Figure 3-34. *GraphiQL*

Let's also check for the fluid one. We can check this from the same `file` code, but we need to give aliases, as the name is the same. This is shown in Figure 3-35.


localhost:8000/___graphql?query=(%0A%20 fixed%3A file(relativePath%3A (eq%3A "icon.jpg
Most Visited
Getting Started
OSS
GraphiQL
Prettify
History
Explorer
Code Exporter
Docs

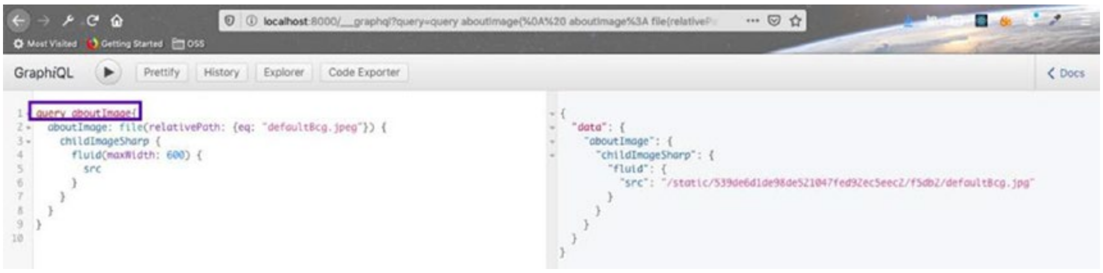
```
{
  fixed: file(relativePath: {eq: "icon.jpg"}) {
    childImageSharp {
      fixed(width: 300, height: 400) {
        src
      }
    }
  }
  fluid: file(relativePath: {eq: "logo.png"}) {
    childImageSharp {
      fluid {
        src
      }
    }
  }
}
```

```
{
  "data": {
    "fixed": {
      "childImageSharp": {
        "fixed": {
          "src": "/static/ce7f03cf260f0295d8e422e9b81c1789/bfbea/icon.jpg"
        }
      }
    },
    "fluid": {
      "childImageSharp": {
        "fluid": {
          "src": "/static/86210b0fa8b63b1ddc19dab63e922729/6d5bf/logo.png"
        }
      }
    }
  }
}
```

Figure 3-35. *fixed and fluid*

As you can see in Figure 3-35, you cannot use the fragments in GraphiQL and use `src`. This is a limitation as of now, but we can use them without any error in our code.

It's time to use `gatsby-image` in the About component, which is on the home page. We will again create our GraphQL query in GraphiQL. Notice that we have also named the query `aboutImage`. This is shown in Figure 3-36.


localhost:8000/___graphql?query=query aboutImage(%0A%20 aboutImage%3A file(relativeP
Most Visited
Getting Started
OSS
GraphiQL
Prettify
History
Explorer
Code Exporter
Docs

```
query aboutImage{
  aboutImage: file(relativePath: {eq: "defaultBcg.jpeg"}) {
    childImageSharp {
      fluid(maxWidth: 600) {
        src
      }
    }
  }
}
```

```
{
  "data": {
    "aboutImage": {
      "childImageSharp": {
        "fluid": {
          "src": "/static/539de6d1de98de521047fed92ec5eec2/f5db2/defaultBcg.jpg"
        }
      }
    }
  }
}
```

Figure 3-36. *The aboutImage query*

Next, we will move to About component and paste our query from GraphiQL, to a `getAbout` variable. Also, notice that we replaced `src` with a fluid fragment `GatsbyImageSharpFluid_-tracedSVG`.

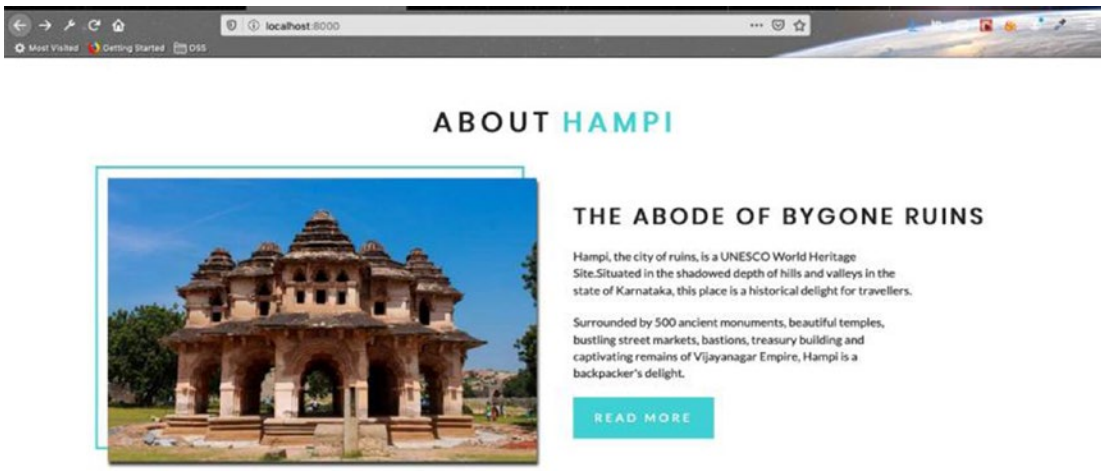
I did some required `imports` at the top of the file and commented out the earlier `import`, which was used to display the image. After that, we are destructuring `aboutImage` and using `useStaticQuery`, which is a hook provided by Gatsby.

Finally, we use `Img` from `gatsby-image` to display the image. The updates are marked in bold in Listing 3-47.

Listing 3-47. The About.js File

```
import React from "react"
import Title from "../Title"
import styles from "../../css/about.module.css"
//import img from "../../images/defaultBcg.jpeg"
import { useStaticQuery, graphql } from 'gatsby'
import Img from 'gatsby-image'

const getAbout = graphql`
query aboutImage{
  aboutImage: file(relativePath: {eq: "defaultBcg.jpeg"}) {
    childImageSharp {
      fluid(maxWidth: 600) {
        ...GatsbyImageSharpFluid_tracedSVG
      }
    }
  }
}
`;

const About = () => {
  const { aboutImage } = useStaticQuery(getAbout);
  return (
    <section className={styles.about}>
      <Title title="about" subtitle="hampi" />
      <div className={styles.aboutCenter}>
        <article className={styles.aboutImg}>
          <div className={styles.imgContainer}>
            {/* <img src={img} alt="about company" /> */}
            <Img fluid={aboutImage.childImageSharp.fluid} alt="landscape" />
          </div>
        </article>
        <article className={styles.aboutInfo}>
          ...
          ...
        </article>
      </div>
    </section>
  )
}

export default About
```

When you see the output on your browser, you can see the About component displaying the same image, as shown in Figure 3-37. But what you see is an optimized image, which loads very quickly on all screens with even with slow Internet speeds.



ABOUT HAMPI

THE ABODE OF BYGONE RUINS

Hampi, the city of ruins, is a UNESCO World Heritage Site.Situated in the shadowed depth of hills and valleys in the state of Karnataka, this place is a historical delight for travellers.

Surrounded by 500 ancient monuments, beautiful temples, bustling street markets, bastions, treasury building and captivating remains of Vijayanagar Empire, Hampi is a backpacker's delight.

READ MORE

Figure 3-37. *The new image*

Background Image Optimization

In this section, we will start optimizing the background images on the site. Right now we have only one big background image on the home page. For this, we will use the gatsby-background-image plugin. The doc is here[14].

Stop the gatsby develop and npm install the plugin by running npm install --save gatsby-background-image from the terminal. Next, start the development server using gatsby develop. After that, head over to your code editor and create a new file called StyledHero.js inside it.

[14]https://www.gatsbyjs.org/packages/gatsby-background-image/

Put the contents in Listing 3-48 in the StyledHero.js file. We are making StyledHero.js a general-purpose component, which can be used in all pages for the large image. All of the code for the component is taken from the docs[15] of the gatsby-background-image plugin.

Listing 3-48. The StyledHero.js File

```
import React from 'react'
import styled from 'styled-components'
import BackgroundImage from 'gatsby-background-image'

const StyledHero = ({ img, className, children, home }) => {
    return (
        <BackgroundImage className={className} fluid={img} home={home}>
            {children}
        </BackgroundImage>
    )
}

export default styled(StyledHero)`
    min-height:${props => props.home ? 'calc(100vh - 62px)': '50vh'};
    background:${props => props.home ? 'linear-gradient(rgba(63, 208,
    212, 0.7), rgba(0, 0, 0, 0.7))': 'none'};
    background-position: center;
    background-size: cover;
    opacity: 1 !important;
    display: flex;
    justify-content: center;
    align-items: center;
`
```

We are using one additional props in our component, which is home. Our home page will also contain a linear gradient and the size of the image will cover the entire page.

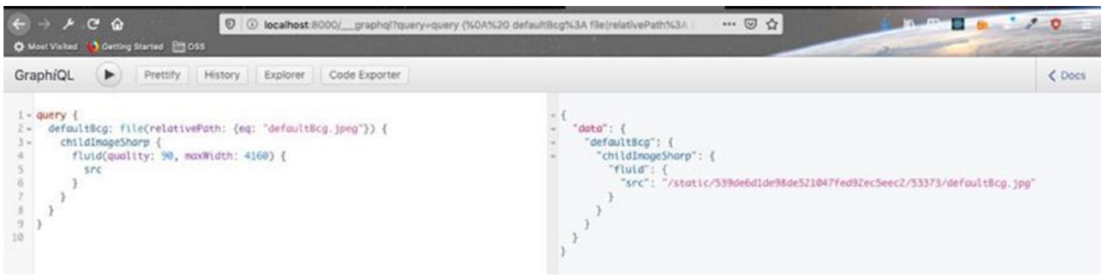
Now let's render StyledHero on the home page. Move over to index.js and replace SimpleHero with StyledHero, as marked in bold in Listing 3-49.

[15] https://www.gatsbyjs.org/packages/gatsby-background-image/

Listing 3-49. The index.js File

```
import React from "react"
import Layout from "../components/Layout"
import StyledHero from "../components/StyledHero"
import Banner from "../components/Banner"
import About from "../components/Home/About"
import Tips from "../components/Home/Tips"
import { Link } from "gatsby"

export default () => (
    <Layout>
        <StyledHero>
            <Banner
                title="Amazing Hampi"
                info="Come and Explore Hampi, the city of ruins, which is a
                UNESCO World Heritage Site."
            >
                <Link to="/places" className="btn-white">explore places
                </Link>
            </Banner>
        </StyledHero>
        <About />
        <Tips />
    </Layout>
)
```

Next, we will create a query to display the images in GraphiQL, as shown in Figure 3-38.


localhost:8000/___graphql?query=query (%0A%20 defaultBcg%3A file(relativePath%3A
Most Visited
Getting Started
OSS
GraphiQL
Prettify
History
Explorer
Code Exporter
Docs

```
query {
  defaultBcg: file(relativePath: {eq: "defaultBcg.jpeg"}) {
    childImageSharp {
      fluid(quality: 90, maxWidth: 4160) {
        src
      }
    }
  }
}
```

```
{
  "data": {
    "defaultBcg": {
      "childImageSharp": {
        "fluid": {
          "src": "/static/539de6d1de98de521047fed92ec5eec2/53373/defaultBcg.jpg"
        }
      }
    }
  }
}
```

Figure 3-38. *GraphiQL*

Earlier we used `StaticQuery`, but we will be using `PageQuery` because `index.js` is a page. Here, we are using the query we created in our GraphiQL and then passing it as props to our `StyledHero` component. The updated code is marked in bold in Listing 3-50.

Listing 3-50. The index.js File

```
import React from "react"
import Layout from "../components/Layout"
import StyledHero from "../components/StyledHero"
import Banner from "../components/Banner"
import About from "../components/Home/About"
import Tips from "../components/Home/Tips"
import { Link } from "gatsby"
import { graphql } from 'gatsby'

export const query = graphql`
query {
    defaultBcg: file(relativePath: {eq: "defaultBcg.jpeg"}) {
        childImageSharp {
            fluid(quality: 90, maxWidth: 4160) {
                ...GatsbyImageSharpFluid_withWebp
            }
        }
    }
}
`;

export default ({ data }) => (
    <Layout>
        <StyledHero home="true" img={data.defaultBcg.childImageSharp.fluid}>
            <Banner
                title="Amazing Hampi"
                info="Come and Explore Hampi, the city of ruins, which is a
                UNESCO World Heritage Site."
            >
```

```
                <Link to="/places" className="btn-white">explore places
                </Link>
            </Banner>
        </StyledHero>
        <About />
        <Tips />
    </Layout>
)
```

Now, when we check the browser, we can see the optimized background image, as shown in Figure 3-39.

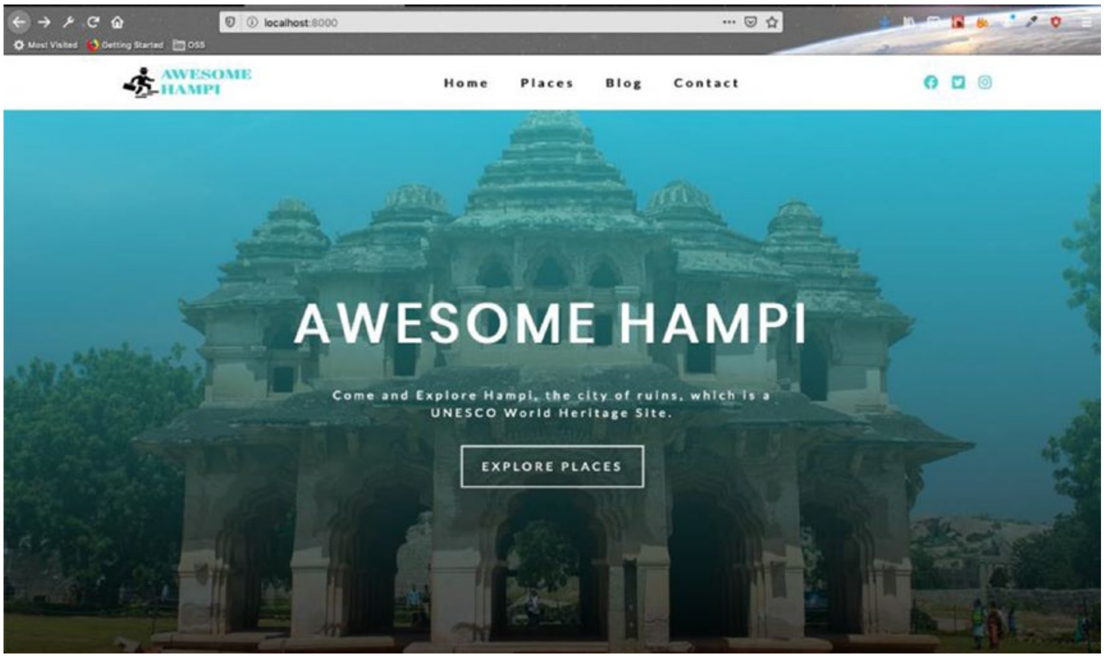

localhost:8000
AWESOME HAMPI
Home
Places
Blog
Contact
AWESOME HAMPI
Come and Explore Hampi, the city of ruins, which is a UNESCO World Heritage Site.
EXPLORE PLACES

Figure 3-39. *The optimized background*

Next, we will add background images to all the pages. First, we will add the `places.js` file. Everything will be similar to `index.js`, except that we are not passing the `home` prop in `StyledHero`. The updated code is marked in bold in Listing 3-51.

Listing 3-51. The places.js File

```
import React from 'react'
import Layout from "../components/Layout"
import StyledHero from "../components/StyledHero"
import { graphql } from 'gatsby'

export const query = graphql`
query {
    defaultBcg: file(relativePath: {eq: "defaultBcg.jpeg"}) {
        childImageSharp {
            fluid(quality: 90, maxWidth: 4160) {
                ...GatsbyImageSharpFluid_withWebp
            }
        }
    }
}
`;

export default function places({ data }) {
    return (
        <Layout>
               <StyledHero img={data.defaultBcg.childImageSharp.fluid}>
                            Places page
                     </StyledHero>
               </Layout>
    )
}
```

By not passing the home prop, we get a smaller image with no linear gradient, as shown in Figure 3-40.

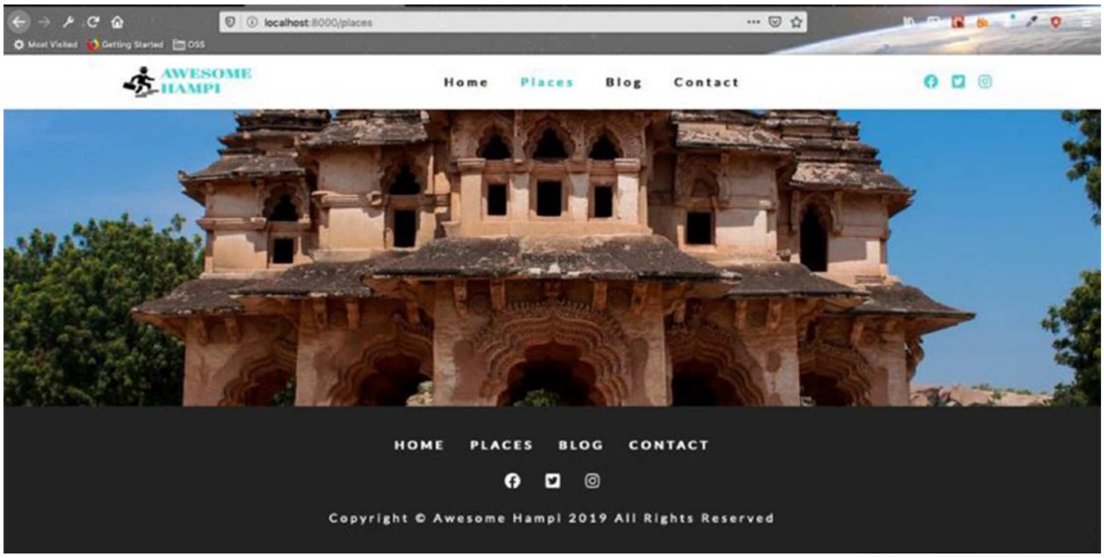


Figure 3-40. *The Places page*

Next, we will update the `contact.js` page. Here we are using a different image. The updated code is marked in bold in Listing 3-52.

Listing 3-52. The contact.js Page

```
import React from 'react'
import Layout from "../components/Layout"
import StyledHero from "../components/StyledHero"
import { graphql } from 'gatsby'

export const query = graphql`
query {
    connectBcg: file(relativePath: {eq: "connectBcg.jpeg"}) {
        childImageSharp {
            fluid(quality: 90, maxWidth: 4160) {
                ...GatsbyImageSharpFluid_withWebp
            }
        }
    }
}
`;
```

```
export default function contact({ data }) {
    return (
        <Layout>
            <StyledHero img={data.connectBcg.childImageSharp.fluid}>
                    Contact Page
            </StyledHero>
        </Layout>
    )
}
```

It will result in a change to the Contact page in the browser, as shown in Figure 3-41.

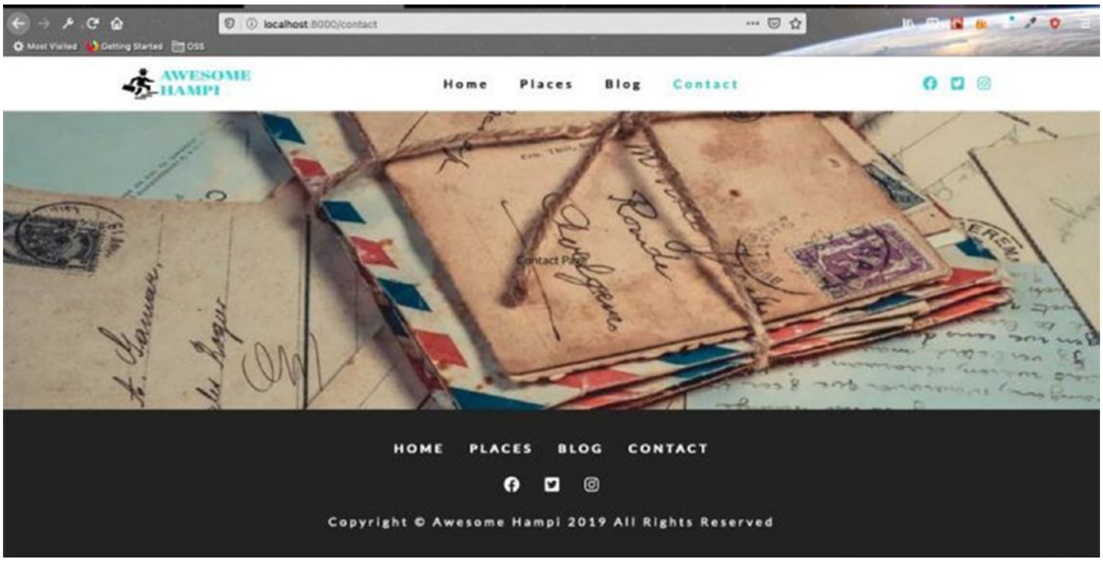


Figure 3-41. *The updated Contact page*

Next, we will update the `blog.js` file. We are using a different image here too. The updated code is marked in bold in Listing 3-53.

Listing 3-53. The blog.js File

```
import React from 'react'
import Layout from "../components/Layout"
import StyledHero from "../components/StyledHero"
import { graphql } from 'gatsby'
```

```
export const query = graphql`
query {
    blogBcg: file(relativePath: {eq: "blogBcg.jpeg"}) {
        childImageSharp {
            fluid(quality: 90, maxWidth: 4160) {
                ...GatsbyImageSharpFluid_withWebp
            }
        }
    }
}
`;

export default function blog({ data }) {
    return (
        <Layout>
            <StyledHero img={data.blogBcg.childImageSharp.fluid}>
                    Blog Page
              </StyledHero>
        </Layout>
    )
}
```

The blog page is shown in Figure 3-42.

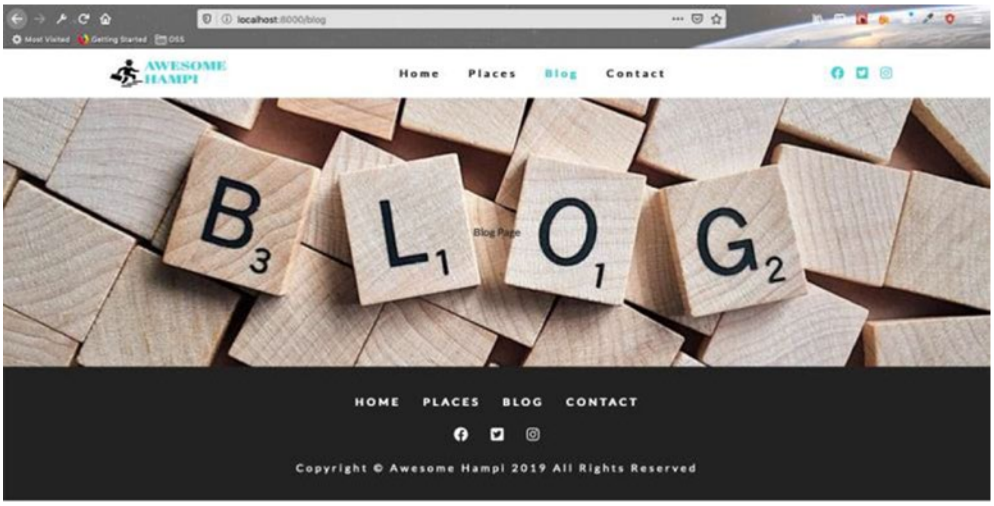


Figure 3-42. The blog page

You can find the code for this page at this[16] link. Since I pushed my code to GitHub, it also did an automatic deployment to Netlify. You can also find the updated site on `https://amazinghampi.netlify.com/`.[17]

Creating a Page Transition

We will start with our page transition, which will occur when we navigate to a different page. We will be again using an awesome Gatsby plugin, called `gatsby-plugin-transition-link`. The doc link can be found here[18].

Now, as per the docs, we need to `npm install` the plugin first. So, head over to your terminal and close `gatsby develop` if it is running. After that, install the plugin through the terminal by running `npm i gatsby-plugin-transition-link`.

Then open `gatsby-config.js` and add the plugin. It is marked in bold in Listing 3-54.

Listing 3-54. The gatsby-config.js File

```
module.exports = {
          plugins: [
          {
      resolve: `gatsby-source-filesystem`,
      options: {
        name: `images`,
        path: `${__dirname}/src/images/`,
      },
    },
            `gatsby-plugin-styled-components`,
            `gatsby-transformer-sharp`,
            `gatsby-plugin-sharp`,
            `gatsby-plugin-transition-link`
            ]
}
```

[16] https://github.com/nabendu82/gatsbyTourism
[17] https://amazinghampi.netlify.com/
[18] https://transitionlink.tylerbarnes.ca/docs/

We also need to install one more package to start using the transition link. This package will be used for `AniLink`, which is a part of the transition link. We need to run `npm i gsap` from the terminal.

Then start the development server by running `gatsby develop`. After that, we will start replacing our `Link` with `AniLink`. Head over to `Navbar.js` and update it. Notice that we are using a `prop paintDrip` and a hex code. These are used to customize the transition. We can use other props as well, as per their docs[19]. The updated code is marked in bold in Listing 3-55.

Listing 3-55. The Navbar.js File

```
import React, { useState } from "react"
//import { Link } from "gatsby"
import AniLink from "gatsby-plugin-transition-link/AniLink"
import styles from "../css/navbar.module.css"
import { FaAlignRight } from "react-icons/fa"
...
...

    return (
        <nav className={styles.navbar}>
            <div className={styles.navCenter}>
                <div className={styles.navHeader}>
                                ...
                                ...
                </div>
                <ul className={isOpen ? `${styles.navLinks} ${styles.
                showNav}` : `${styles.navLinks}`}>
                    {links.map((item, index) => {
                        return (
                            <li key={index}>
                                <AniLink paintDrip hex="#AEECEE" to={item.
                                path}>{item.text}</AniLink>
                            </li>
                        )
```

[19] https://transitionlink.tylerbarnes.ca/docs/anilink/

```
                    })}
                </ul>
                <div className={styles.navSocialLinks}>
                                ...
                                ...
                </div>
            </div>
        </nav>
    )
}

export default Navbar
```

Next, head over to `Footer.js` and update all `Links` to `AniLink`. The updated code is marked in bold in Listing 3-56.

Listing 3-56. The Footer.js File

```
import React from "react"
import styles from "../css/footer.module.css"
import links from "../constants/links"
import socialIcons from "../constants/social-icons"
//import { Link } from "gatsby"
import AniLink from "gatsby-plugin-transition-link/AniLink"

const Footer = () => {
return (
    <footer className={styles.footer}>
        <div className={styles.links}>
        {links.map((item, index) => {
            return (
            <AniLink paintDrip hex="#AEECEE" key={index} to={item.path}>
                {item.text}
            </AniLink>
            )
        })}
        </div>
        <div className={styles.icons}>
                       ...
                      ...
        </div>
        <div className={styles.copyright}>
            copyright &copy; Amazing Hampi {new Date().getFullYear()} all
            rights reserved
        </div>
    </footer>
)
}

export default Footer;
```

There is also a link in `index.js`. Let's update it. The updated code is marked in bold in Listing 3-57.

Listing 3-57. The index.js File

```
...
...
import Tips from "../components/Home/Tips"
//import { Link } from "gatsby"
import AniLink from "gatsby-plugin-transition-link/AniLink"
import { graphql } from 'gatsby'

...
...

export default ({ data }) => (
    <Layout>
        <StyledHero home="true" img={data.defaultBcg.childImageSharp.fluid}>
            <Banner
                title="Amazing Hampi"
                info="Come and Explore Hampi, the city of ruins, which is a
                UNESCO World Heritage Site."
            >
```

```
                <AniLink paintDrip hex="#AEECEE" to="/places"
                className="btn-white">explore places</AniLink>
            </Banner>
        </StyledHero>
        <About />
        <Tips />
    </Layout>
)
```

Let's also update the link in the `404.js` file. The updated code is marked in bold in Listing 3-58.

Listing 3-58. The 404.js File

```
import React from "react"
import Layout from "../components/Layout"
import styles from "../css/error.module.css"
//import { Link } from "gatsby"
import AniLink from "gatsby-plugin-transition-link/AniLink"
import Banner from "../components/Banner"

export default function error() {
return (
    <Layout>
        <header className={styles.error}>
        <Banner title="oops it's a dead end">
            <AniLink paintDrip hex="#AEECEE" to="/" className="btn-white">
                back to home page
            </AniLink>
        </Banner>
        </header>
    </Layout>
)
}
```

If you move to a new page by clicking the link, you will get the transition animation.

Adding a Contact Form

In this section, we add a contact form to our project. Head over to your code editor and add the `contact.module.css` file inside the `css` folder. The contents are shown in Listing 3-59.

Listing 3-59. The contact.module.css File

```css
.contact {
  padding: 4rem 0;
}
.center {
  width: 80vw;
  margin: 0 auto;
}
@media screen and (min-width: 992px) {
  .center {
    width: 50vw;
    margin: 0 auto;
  }
}
.contact label {
  text-transform: capitalize;
  display: block;
  margin-bottom: 0.5rem;
}
.formControl,
.submit {
  width: 100%;
  font-size: 1rem;
  margin-bottom: 1rem;
  padding: 0.375rem 0.75rem;
  border: 1px solid var(--darkGrey);
  border-radius: 0.25rem;
}
```

```
.submit {
  background-color: var(--primaryColor);
  border-color: var(--primaryColor);
  text-transform: uppercase;
  color: var(--mainBlack);
  transition: var(--mainTransition);
  cursor: pointer;
}
.submit:hover {
  background: var(--darkGrey);
  color: var(--mainWhite);
  border-color: var(--darkGrey);
}
```

Next, create a folder called `contact` inside the `components` folder and add a `Contact.js` file inside it. This is a simple form with three fields for the name, email, and message. The contents of the `Contact.js` file are shown in Listing 3-60.

Listing 3-60. Contact.js

```
import React from "react"
import Title from "../Title"
import styles from "../../css/contact.module.css"
const Contact = () => {
  return (
    <section className={styles.contact}>
      <Title title="contact" subtitle="us" />
      <div className={styles.center}>
        <form className={styles.form}>
          <div>
            <label htmlFor="name">name</label>
            <input
              type="text"
              name="name"
              id="name"
              className={styles.formControl}
              placeholder="john smith"
            />
          </div>
          <div>
            <label htmlFor="email">email</label>
            <input
              type="email"
              name="email"
              id="email"
              className={styles.formControl}
              placeholder="email@email.com"
            />
          </div>
          <div>
            <label htmlFor="message">message</label>
            <textarea
              name="message"
              id="message"
              rows="10"
              className={styles.formControl}
              placeholder="hello there"
            />
          </div>
          <div>
            <input
              type="submit"
              value="submit here"
              className={styles.submit}
            />
          </div>
```

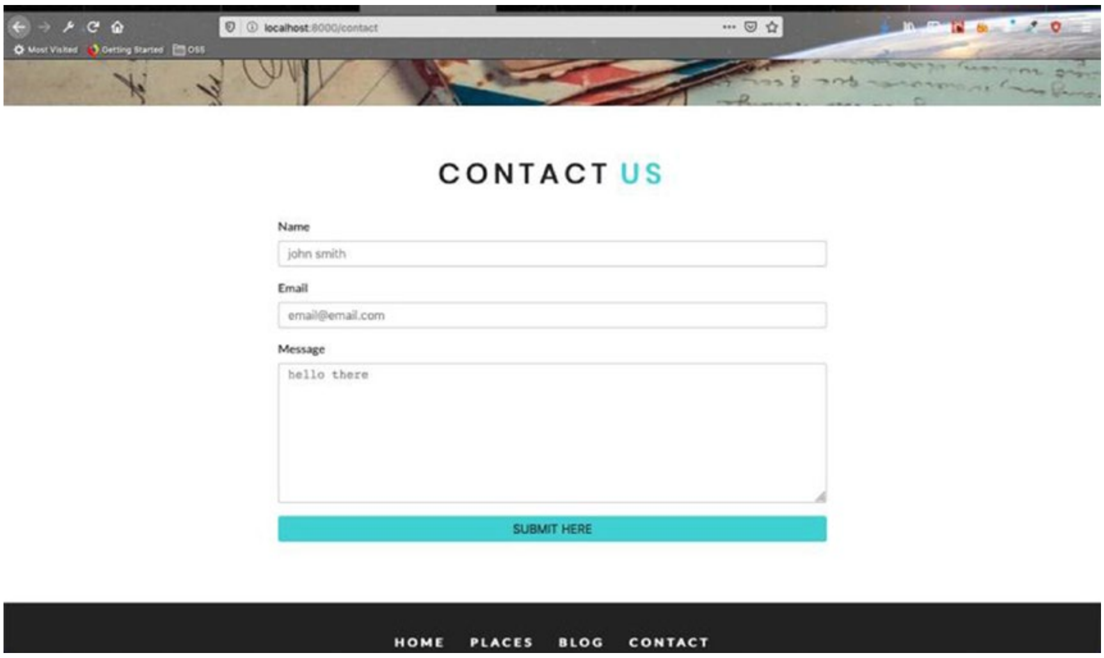
```
        </form>
      </div>
    </section>
  )
}

export default Contact

 1
```

Next, add this `Contact` component to the `contact.js` page. The updated code is marked in bold in Listing 3-61.

Listing 3-61. The Updated Contact.js File

```
import React from 'react'
import Layout from "../components/Layout"
import StyledHero from "../components/StyledHero"
import { graphql } from 'gatsby'
import Contact from '../components/Contact/Contact'

...
...

export default function contact({ data }) {
    return (
        <Layout>
            <StyledHero img={data.connectBcg.childImageSharp.fluid}>
            </StyledHero>
            <Contact />
        </Layout>
    )
}
```

The form appears on the Contact page, as shown in Figure 3-43.

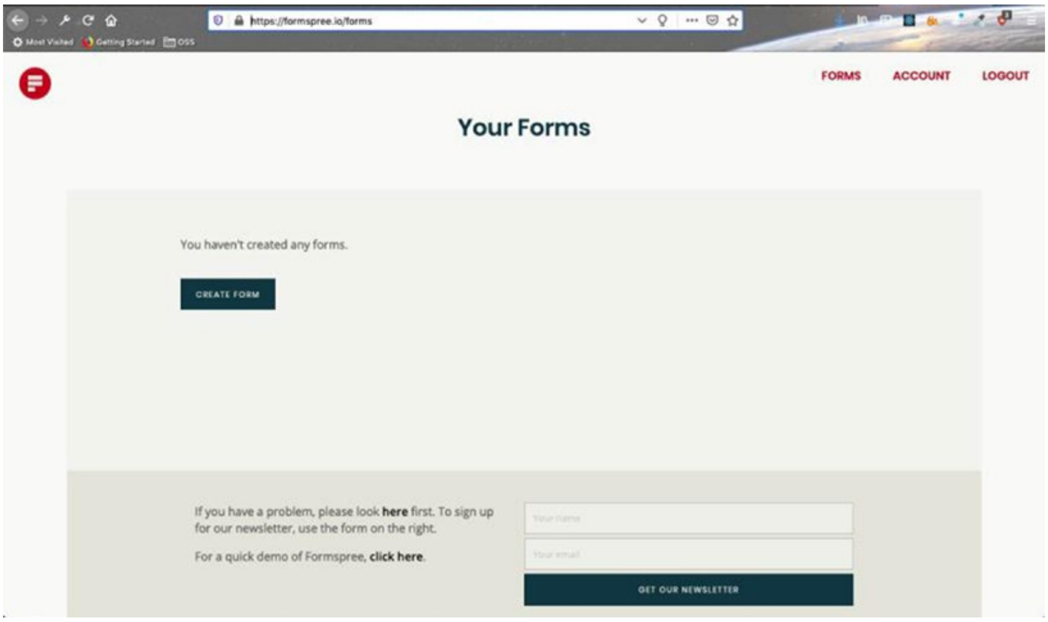


Name: john smith
Email: email@email.com
Message: hello there

Figure 3-43. *The Contact Us form*

We will be using an service called Formspree[20] to get user data from this form. Once you register and verify your email, you will see the screen in Figure 3-44.

[20] https://formspree.io



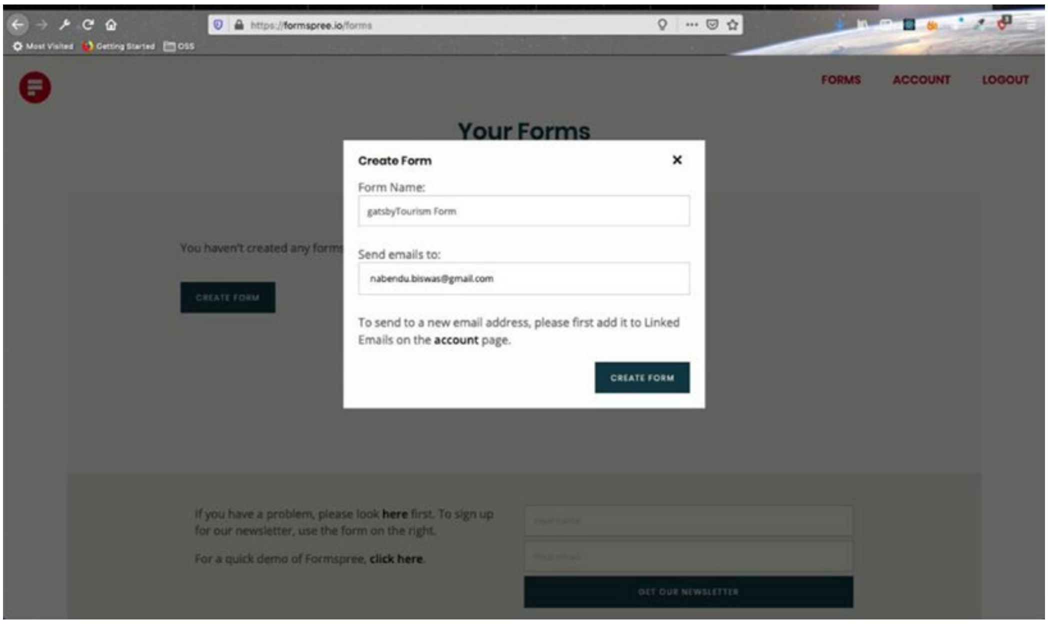
You haven't created any forms.

If you have a problem, please look **here** first. To sign up for our newsletter, use the form on the right.

For a quick demo of Formspree, **click here**.

Figure 3-44. *Create a form using FormspreeFormspree*

Once you click the Create Form button, a popup will ask you to provide a name for your form, as shown in Figure 3-45. Add the name and click Create Form again.



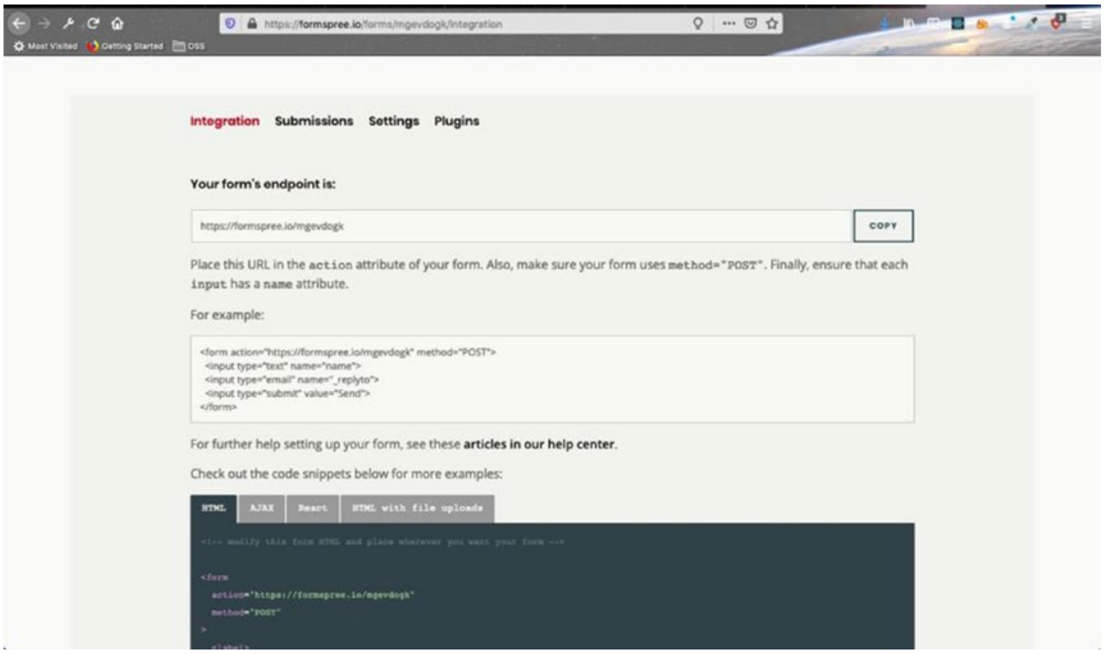
Form Name: gatsbyTourism Form

Send emails to: nabendu.biswas@gmail.com

To send to a new email address, please first add it to Linked Emails on the **account** page.

Figure 3-45. *The Create Form popup*

The next page will show the setup instructions; see Figure 3-46.



Your form's endpoint is:

https://formspree.io/mgevdogk

Place this URL in the `action` attribute of your form. Also, make sure your form uses `method="POST"`. Finally, ensure that each `input` has a `name` attribute.

For example:

```
<form action="https://formspree.io/mgevdogk" method="POST">
  <input type="text" name="name">
  <input type="email" name="_replyto">
  <input type="submit" value="Send">
</form>
```

For further help setting up your form, see these **articles in our help center**.

Check out the code snippets below for more examples:

```
<!-- modify this form HTML and place wherever you want your form -->

<form
  action="https://formspree.io/mgevdogk"
  method="POST"
>
  <label>
```

Figure 3-46. *The setup instructions*

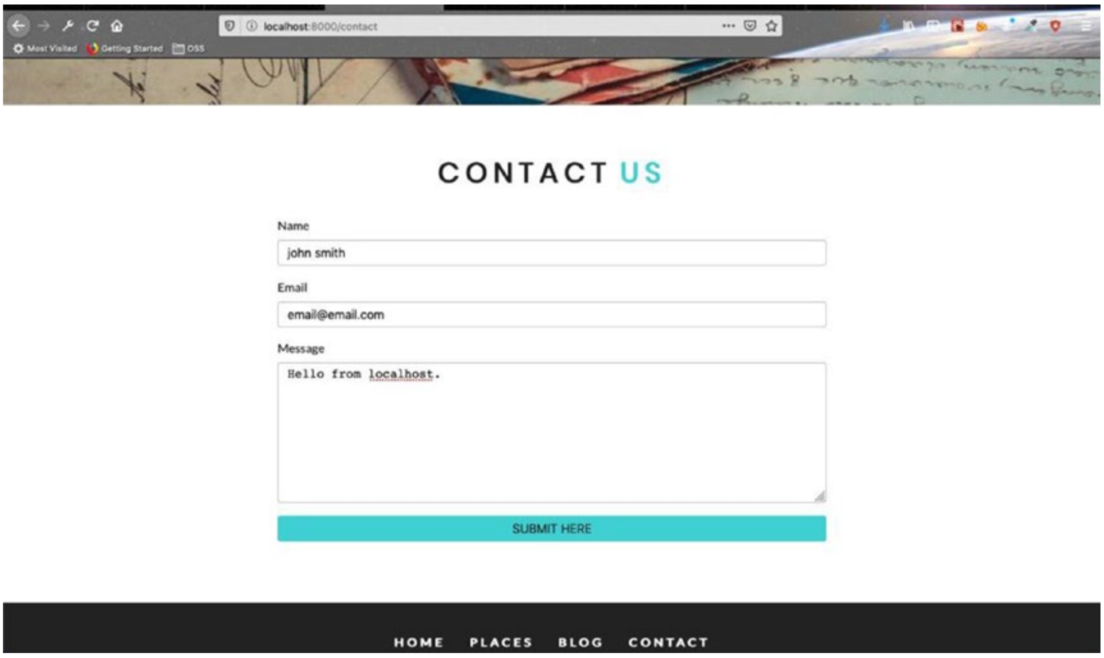
As shown in Figure 3-46, we need to add an action and a method to our form. Head over to `Contact.js` and add these. The change is marked in bold in Listing 3-62.

Listing 3-62. The Updated Contact.js File

```
import React from "react"
import Title from "../Title"
import styles from "../../css/contact.module.css"
const Contact = () => {
  return (
    <section className={styles.contact}>
      <Title title="contact" subtitle="us" />
      <div className={styles.center}>
        <form action="https://formspree.io/mgevdogk" method="POST"
        className={styles.form}>
                       ...
                       ...
```

```
          </form>
        </div>
      </section>
    )
  }

export default Contact
```

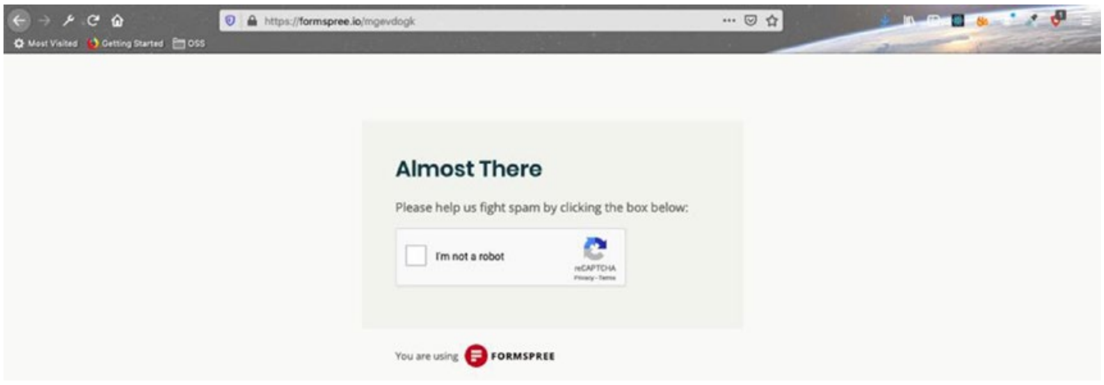
It's time to test our form. We can do that on a localhost also. Head over to the browser and add some values to all the fields. Click Submit Here, as shown in Figure 3-47.



Name
john smith
Email
email@email.com
Message
Hello from localhost.

***Figure 3-47.** Click the Submit Here button*

Next, you will be taken to the infamous I'm Not a Robot check box, as shown in Figure 3-48.



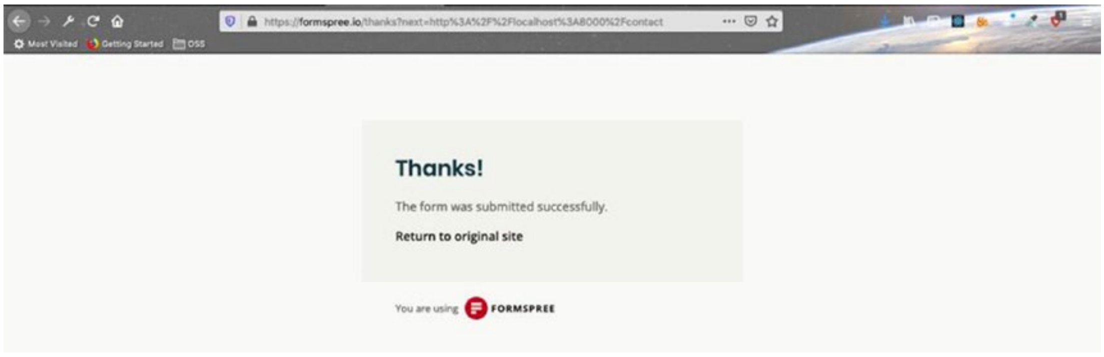
Almost There

Please help us fight spam by clicking the box below:

I'm not a robot

Figure 3-48. *If you are not a robot, check the box*

After verification, it will open the page shown in Figure 3-49. From there, you can move back to the site.


https://formspree.io/thanks?next=http%3A%2F%2Flocalhost%3A8000%2Fcontact
Most Visited
Getting Started
OSS

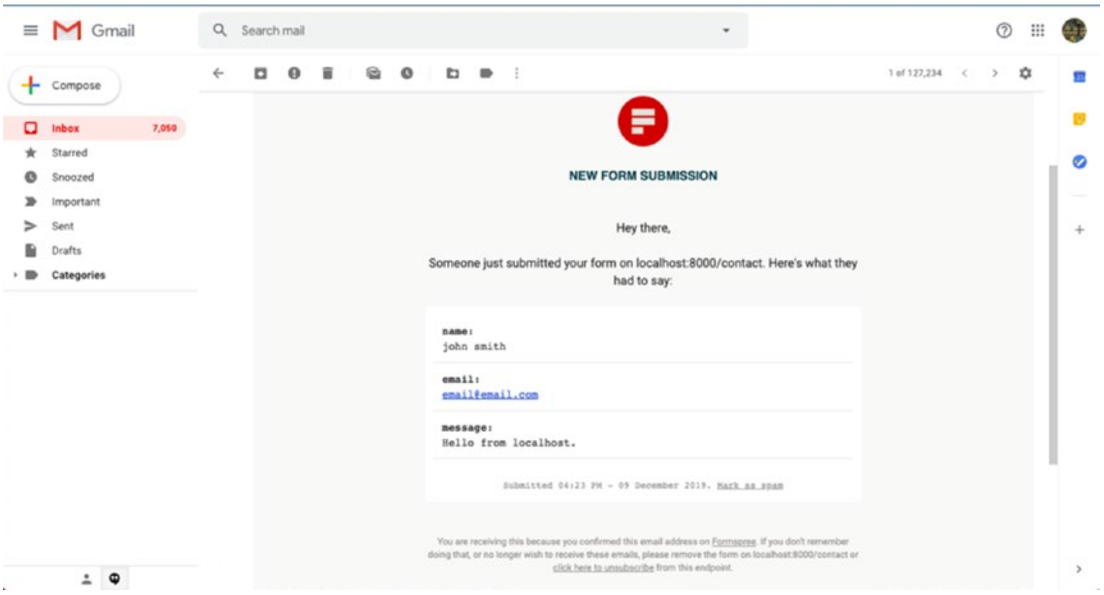
Thanks!

The form was submitted successfully.

Return to original site

Figure 3-49. *The form was submitted successfully*

Once I log in to my Gmail account, I can see that the form was submitted (see Figure 3-50).



NEW FORM SUBMISSION

Hey there,

Someone just submitted your form on localhost:8000/contact. Here's what they had to say:

name:
john smith

email:
email@email.com

message:
Hello from localhost.

Submitted 04:23 PM - 09 December 2019. Mark as spam

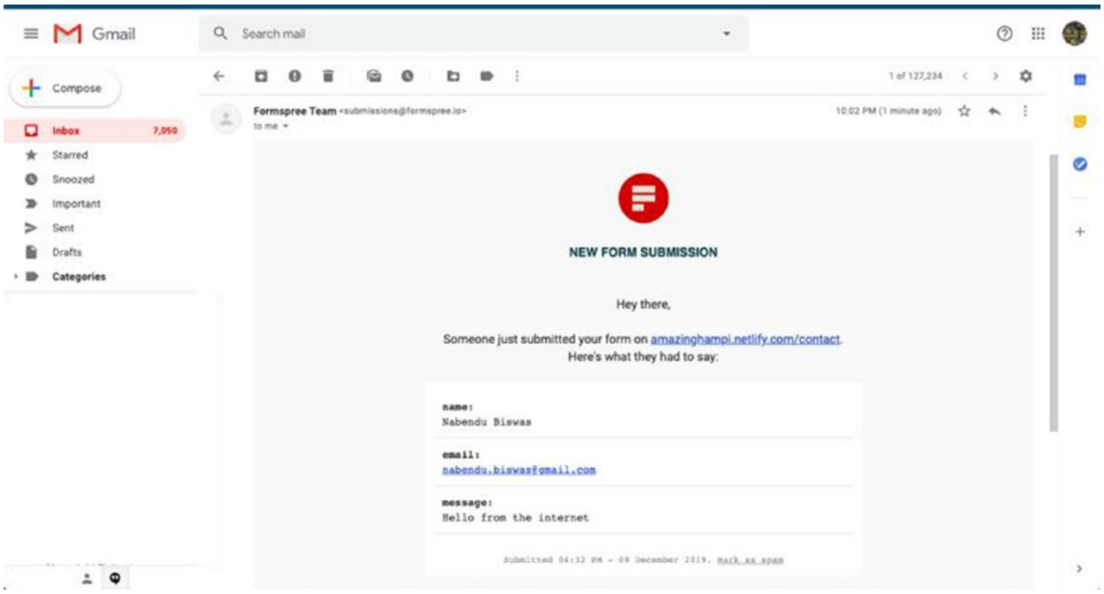
You are receiving this because you confirmed this email address on Formspree. If you don't remember doing that, or no longer wish to receive these emails, please remove the form on localhost:8000/contact or click here to unsubscribe from this endpoint.

Figure 3-50. *The form was submitted*

I pushed the code to GitHub[21]. It will also start my automatic deployment to Netlify. It's time to check the form capabilities on the Internet as well. Once this is deployed, you can head over to `https://amazinghampi.netlify.com/contact`[22] and submit a form.

It will take you through the same process and I get a message in my Gmail (see Figure 3-51).

[21]`https://github.com/nabendu82/gatsbyTourism`
[22]`https://amazinghampi.netlify.com/contact`



Formspree Team <submissions@formspree.io>
to me
10:02 PM (1 minute ago)

NEW FORM SUBMISSION

Hey there,

Someone just submitted your form on amazinghampi.netlify.com/contact.
Here's what they had to say:

```
name:
Nabendu Biswas

email:
nabendu.biswas@gmail.com

message:
Hello from the internet
```

Submitted 04:32 PM - 09 December 2019. Mark as spam

Figure 3-51. *Verification from the Internet*

Summary

This completes Chapter 3 and the first part of the tourism site with Contentful. We covered the following topics in this chapter:

- Creating the basic setup to create the Gatsby site
- Completing the Navbar and Footer components
- Completing the `SimpleHero` component, which shows the image in the site
- Adding two small sections—About Section and Hot Tips
- Deploying the site on the Internet using Netlify
- Optimizing the images using Gatsby plugins
- Doing page translation using the `AniLink` plugin
- Creating a fully working contact form with Formspree

In the next chapter, we will continue using Contentful to build the tourism site. We will set up Contentful and create the Places component in that chapter.

CHAPTER 4

Creating a Tourism Site with Contentful: Part Two

After completing much of the site in the previous chapter, it's time to set it up so it can accept data from the backend. We start this chapter by learning how to set up the Contentful CMS. Next, we will create the Places component. We are also going to use the data stored in the Contentful CMS in the Places component.

Setting Up Contentful

It's time to display data in our project. We can do this with internal data, but we will use a headless CMS (Content Management System) to store our data and then consume it. We will use the Contentful CMS[1] for this project.

CMS Setup

Go ahead and sign up. When you go to the dashboard the first time, you will get the screen shown in Figure 4-1.

[1] https://www.contentful.com/

N. Biswas, *Foundation Gatsby Projects*, https://doi.org/10.1007/978-1-4842-6558-1_4

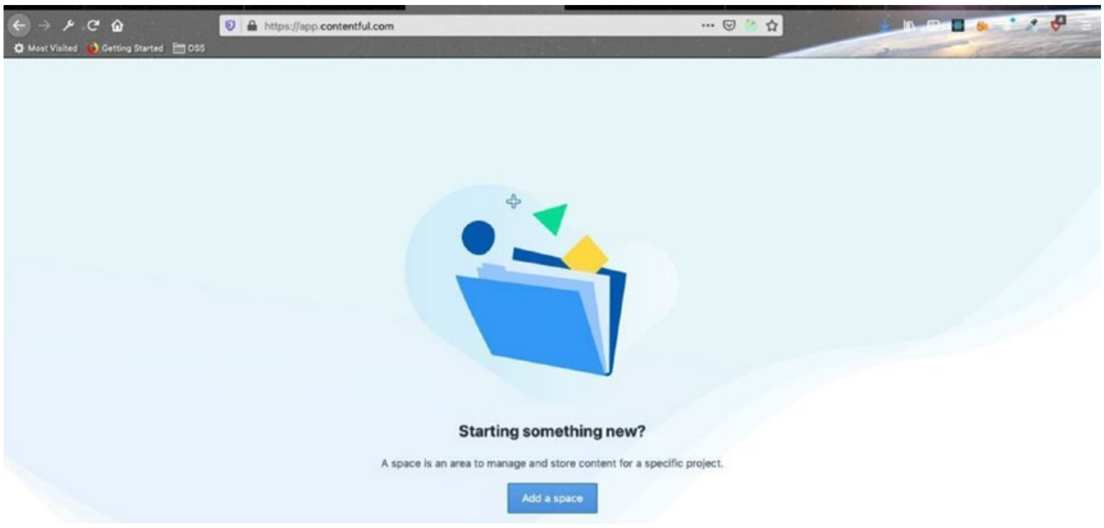
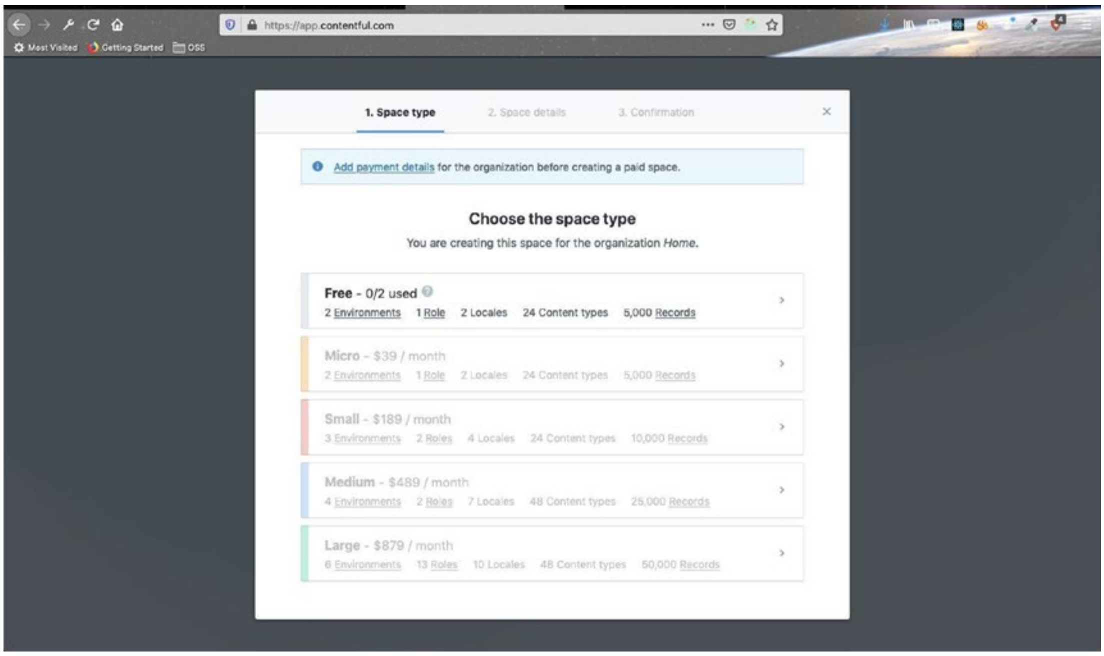

https://app.contentful.com
Most Visited
Getting Started
OSS
Starting something new?
A space is an area to manage and store content for a specific project.
Add a space

Figure 4-1. *Contentful space*

When you click Add a Space, you will see the popup in Figure 4-2. It shows that we have two free spaces and can have up to 5,000 records in the free account.



Add payment details for the organization before creating a paid space.

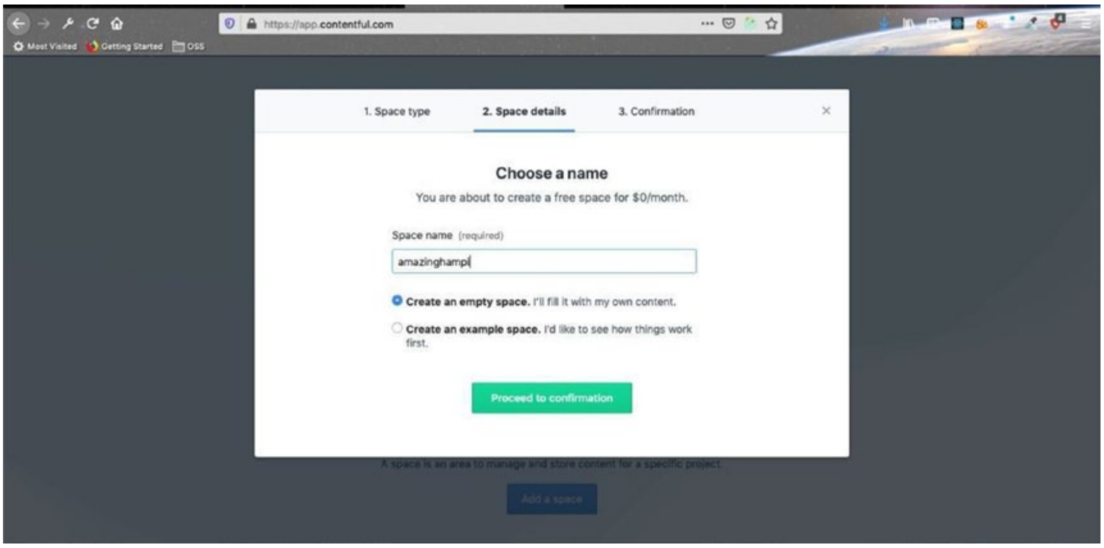
Choose the space type

You are creating this space for the organization *Home*.

Free - 0/2 used	2 Environments	1 Role	2 Locales	24 Content types	5,000 Records
Micro - $39 / month	2 Environments	1 Role	2 Locales	24 Content types	5,000 Records
Small - $189 / month	3 Environments	2 Roles	4 Locales	24 Content types	10,000 Records
Medium - $489 / month	4 Environments	2 Roles	7 Locales	48 Content types	25,000 Records
Large - $879 / month	6 Environments	13 Roles	10 Locales	48 Content types	50,000 Records

Figure 4-2. *Add a space*

When you click the Free tab, it will open the popup shown in Figure 4-3. Here, you have to enter a Space Name.



Choose a name

You are about to create a free space for $0/month.

Space name (required)

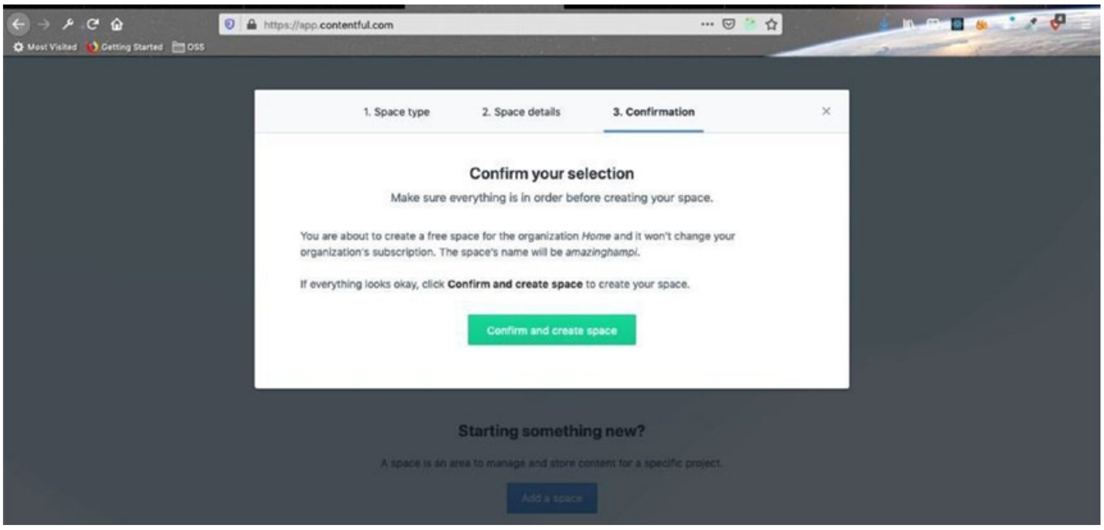
amazinghampi

● Create an empty space. I'll fill it with my own content.

○ Create an example space. I'd like to see how things work first.

Figure 4-3. *Name the space amazinghampi*

After you click the Proceed to Confirmation button, you will see the Confirmation screen shown in Figure 4-4.



Confirm your selection

Make sure everything is in order before creating your space.

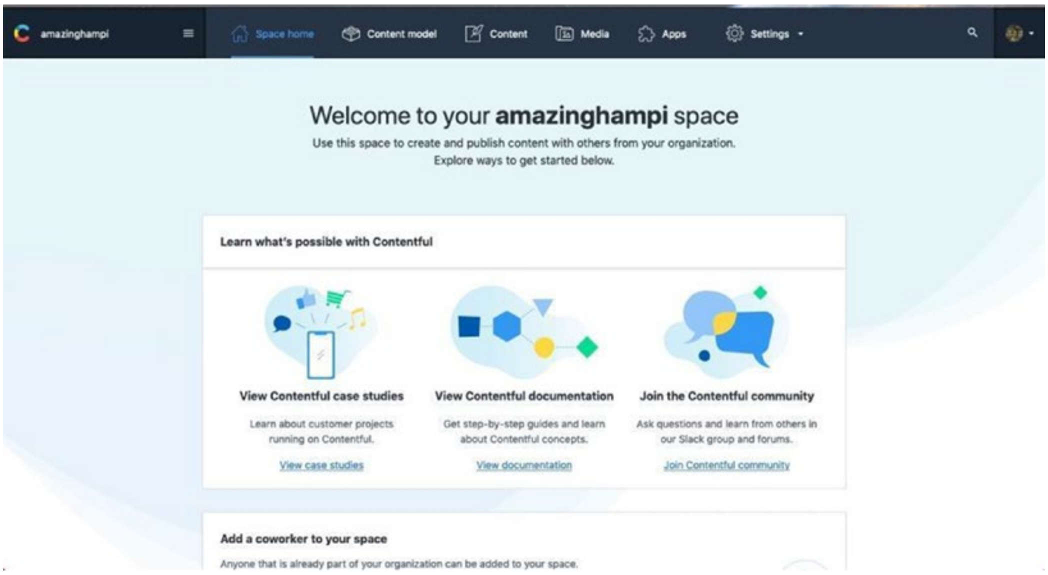
You are about to create a free space for the organization *Home* and it won't change your organization's subscription. The space's name will be *amazinghampi*.

If everything looks okay, click **Confirm and create space** to create your space.

Confirm and create space
Starting something new?
A space is an area to manage and store content for a specific project.
Add a space

Figure 4-4. *Confirm your selection*

Once you click Confirm and Create Space, it will take you to the screen in Figure 4-5. Here, Content Model and Content are the important tabs.



Welcome to your **amazinghampi** space

Use this space to create and publish content with others from your organization. Explore ways to get started below.

Learn what's possible with Contentful

View Contentful case studies

Learn about customer projects running on Contentful.

View case studies

View Contentful documentation

Get step-by-step guides and learn about Contentful concepts.

View documentation

Join the Contentful community

Ask questions and learn from others in our Slack group and forums.

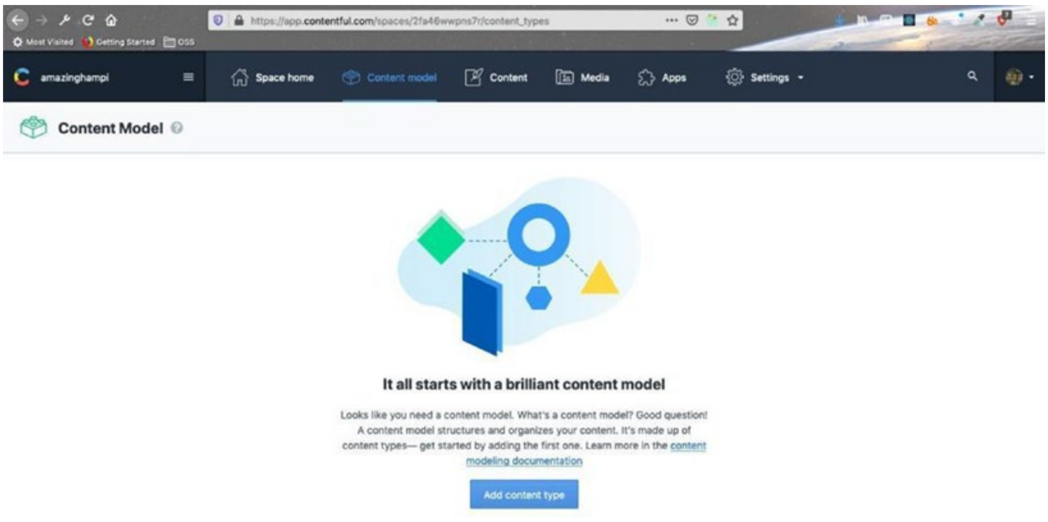
Join Contentful community

Add a coworker to your space

Anyone that is already part of your organization can be added to your space.

***Figure 4-5.** Home page*

The Content Model tab describes the fields of data and the Content tab describes the data. Head over to the Content Model tab and click Add Content Type, as shown in Figure 4-6.

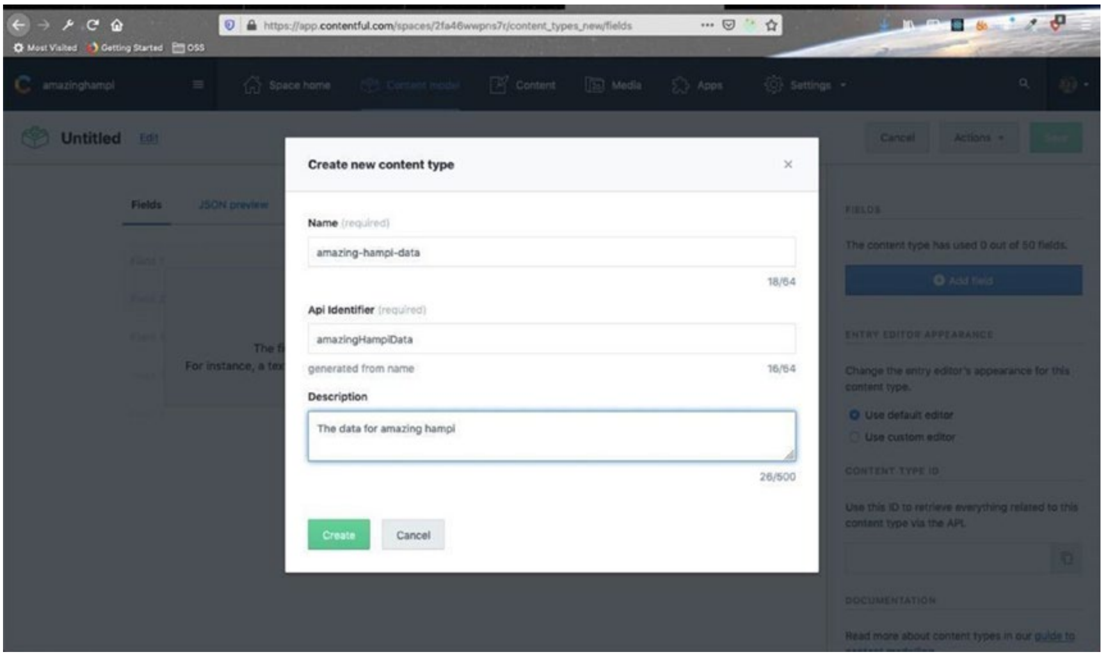


It all starts with a brilliant content model

Looks like you need a content model. What's a content model? Good question! A content model structures and organizes your content. It's made up of content types— get started by adding the first one. Learn more in the content modeling documentation

***Figure 4-6.** Add content type*

Next, you have to provide a name and description and click Create, as shown in Figure 4-7.


Create new content type

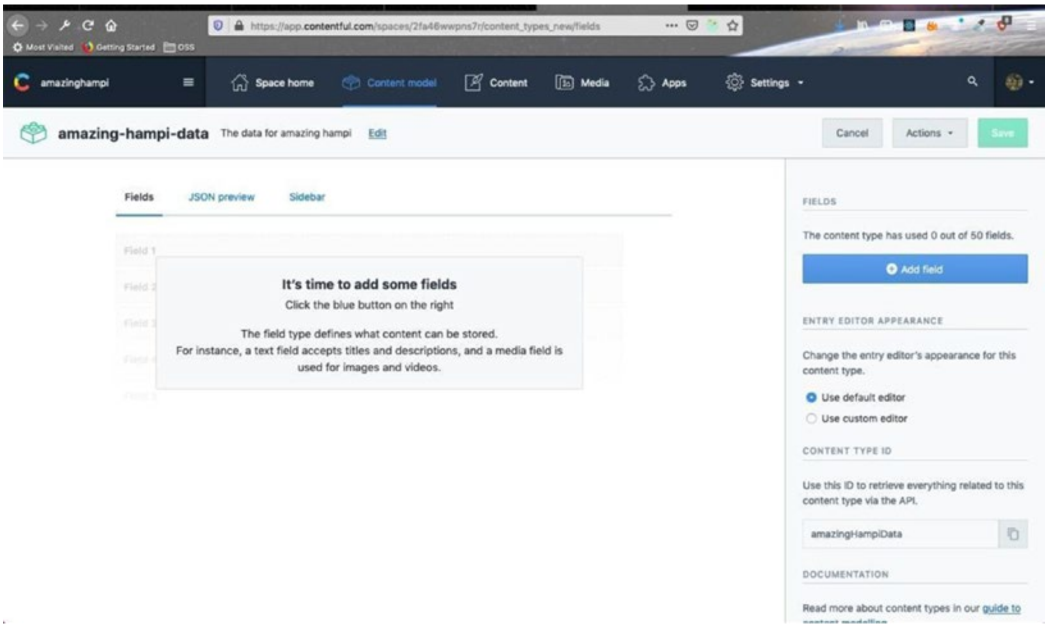
Name (required)
amazing-hampi-data
18/64

Api Identifier (required)
amazingHampiData
generated from name 16/64

Description
The data for amazing hampi
26/500

Figure 4-7. *Provide a name and description*

We are basically creating data about different places to visit in Hampi. The next screen will ask us to add some fields, as shown in Figure 4-8.



It's time to add some fields
Click the blue button on the right
The field type defines what content can be stored.
For instance, a text field accepts titles and descriptions, and a media field is used for images and videos.

The content type has used 0 out of 50 fields.

Change the entry editor's appearance for this content type.

Use this ID to retrieve everything related to this content type via the API.

Read more about content types in our guide to content modelling

Figure 4-8. *Adding fields*

Let's add some fields by clicking the Add Field button. The next screen lists the types of fields, as shown in Figure 4-9. Select Text by clicking it.

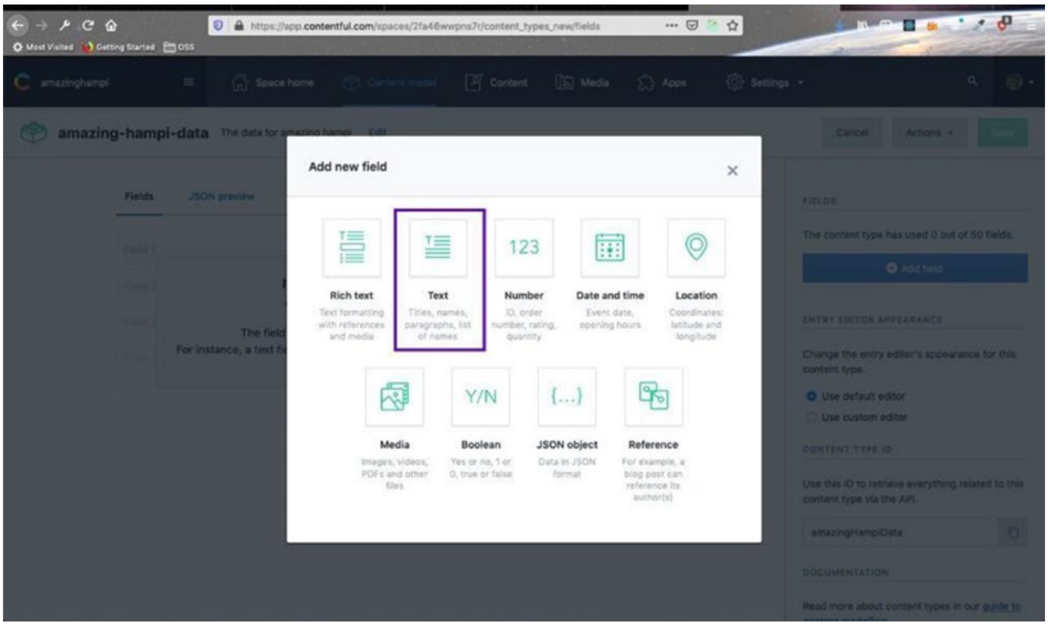
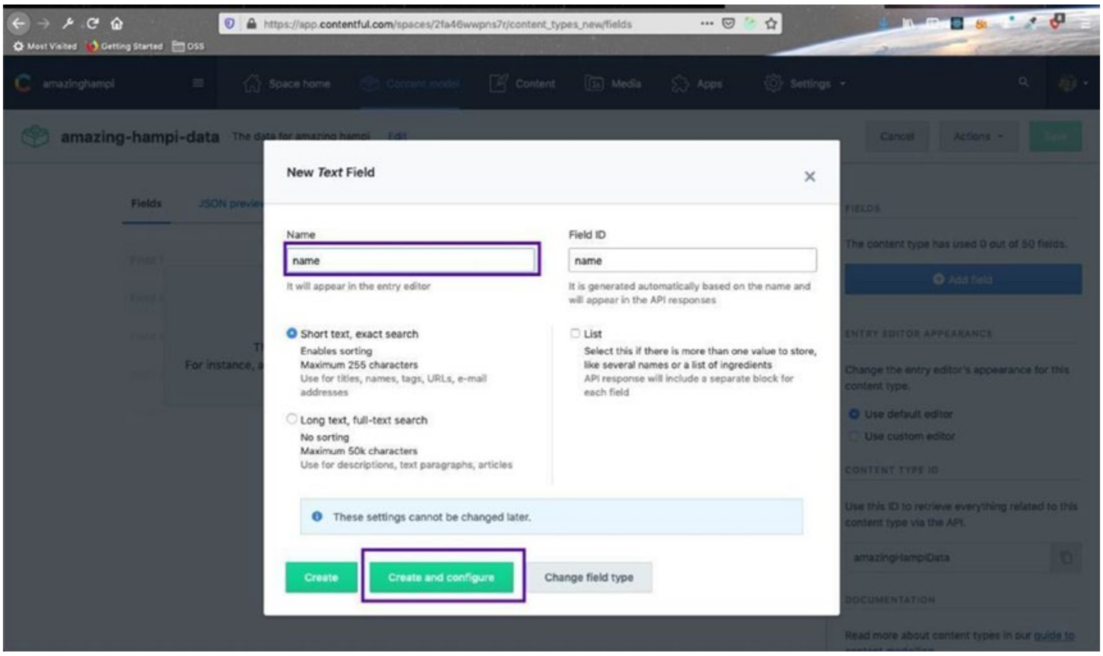

Add new field
Rich text
Text formatting with references and media
Text
Titles, names, paragraphs, list of names
123
Number
ID, order number, rating, quantity
Date and time
Event date, opening hours
Location
Coordinates: latitude and longitude
Media
Images, videos, PDFs and other files
Y/N
Boolean
Yes or no, 1 or 0, true or false
{...}
JSON object
Data in JSON format
Reference
For example, a blog post can reference its author(s)

Figure 4-9. *Adding a new field*

In Figure 4-10, provide a name for this text field. In this case, it is simply called `name`. Click Create and Configure.



New *Text* Field

Name: name
It will appear in the entry editor

Field ID: name
It is generated automatically based on the name and will appear in the API responses

(●) Short text, exact search
Enables sorting
Maximum 255 characters
Use for titles, names, tags, URLs, e-mail addresses

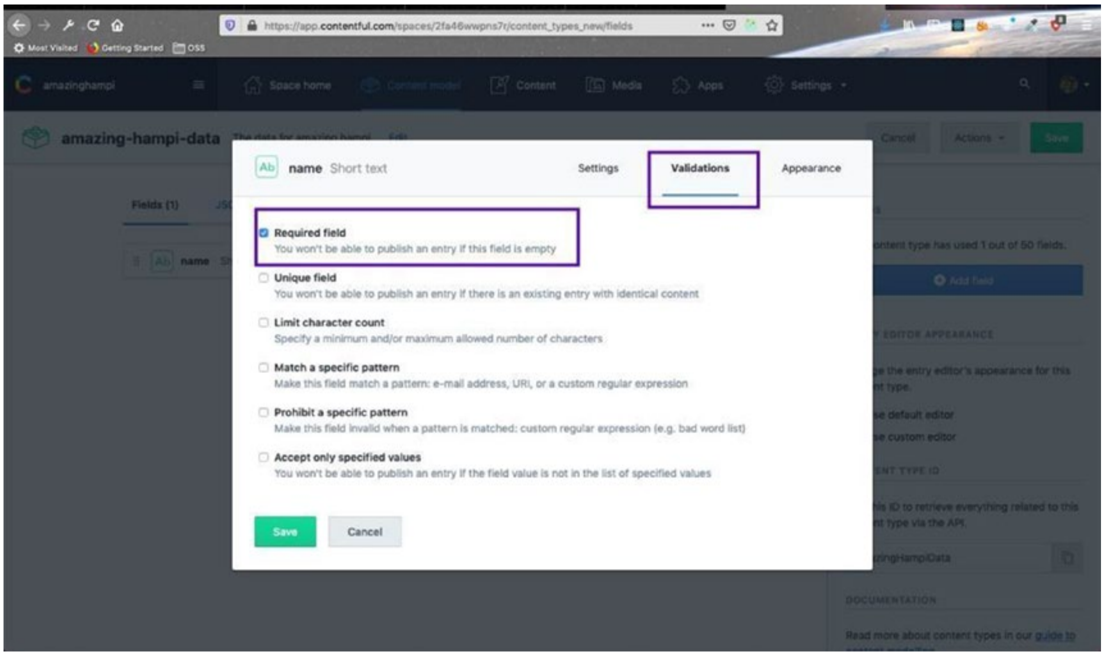
( ) Long text, full-text search
No sorting
Maximum 50k characters
Use for descriptions, text paragraphs, articles

☐ List
Select this if there is more than one value to store, like several names or a list of ingredients
API response will include a separate block for each field

These settings cannot be changed later.

Figure 4-10. *The New Text field*

Then in the next screen, shown in Figure 4-11, click the Validations tab and then check the Required Field option. We are doing this so that users cannot leave this field empty. After that, click the Save button.



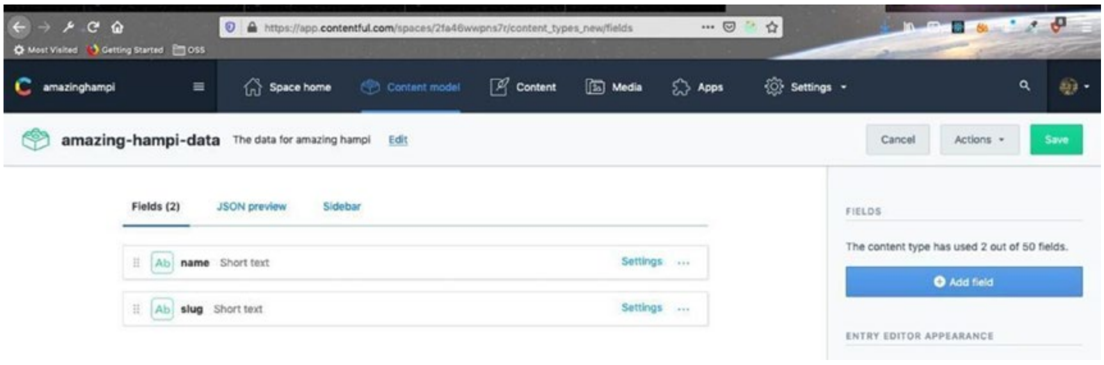
Ab **name** Short text

- [✓] **Required field** You won't be able to publish an entry if this field is empty
- [ ] **Unique field** You won't be able to publish an entry if there is an existing entry with identical content
- [ ] **Limit character count** Specify a minimum and/or maximum allowed number of characters
- [ ] **Match a specific pattern** Make this field match a pattern: e-mail address, URI, or a custom regular expression
- [ ] **Prohibit a specific pattern** Make this field invalid when a pattern is matched: custom regular expression (e.g. bad word list)
- [ ] **Accept only specified values** You won't be able to publish an entry if the field value is not in the list of specified values

Save Cancel

Figure 4-11. *Making validations a required field*

The next field will be `slug` and it will be a text field and will be required. This field is for the text at the end of the URL, which represents each place. Follow the same process as for `name` and create the field. It is shown in Figure 4-12.



Ab **name** Short text Settings

Ab **slug** Short text Settings

The content type has used 2 out of 50 fields.

Figure 4-12. *The slug field*

Next, we will create a Time Required field, as shown in Figure 4-14. This field tells the tourist the time required to visit a place. This will be an integer field, but not a required field. Once you click Add Field, choose Number from the popup, as shown in Figure 4-13.

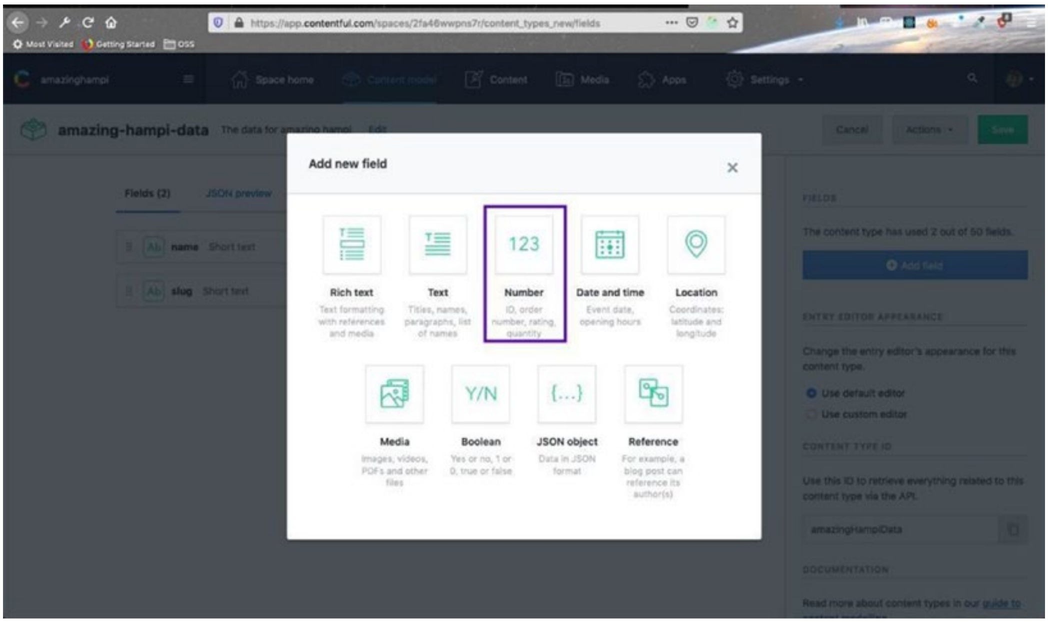
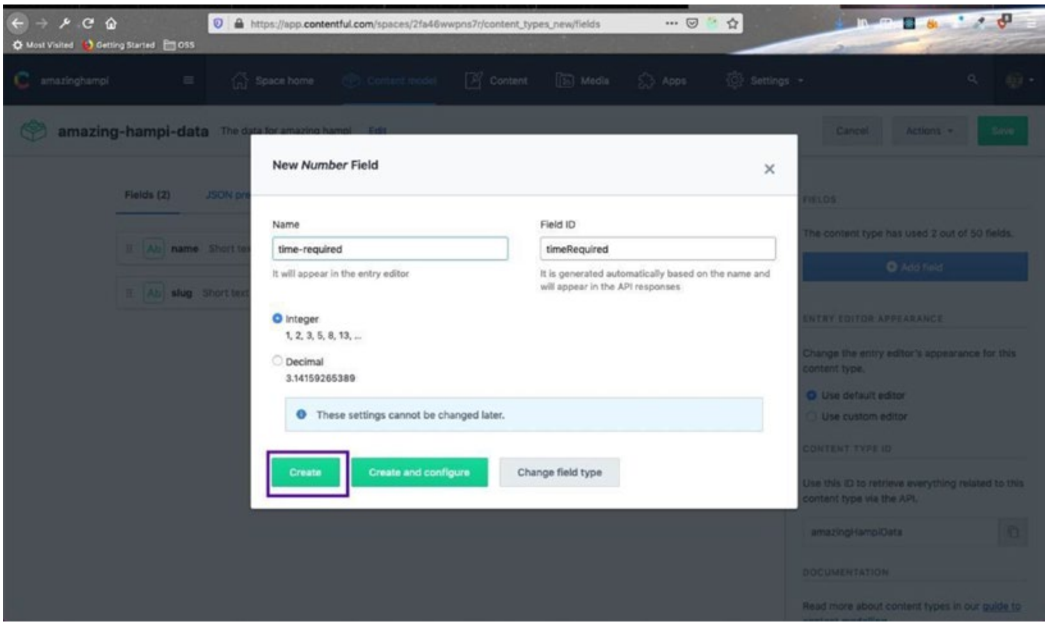

Add new field
Rich text
Text formatting with references and media
Text
Titles, names, paragraphs, list of names
Number
ID, order number, rating, quantity
Date and time
Event date, opening hours
Location
Coordinates: latitude and longitude
Media
Images, videos, PDFs and other files
Boolean
Yes or no, 1 or 0, true or false
JSON object
Data in JSON format
Reference
For example, a blog post can reference its author(s)

Figure 4-13. *The Number field*

Give it a name and then click Create, since it is not a required field. It is shown in Figure 4-14.


New Number Field

Name: time-required
It will appear in the entry editor

Field ID: timeRequired
It is generated automatically based on the name and will appear in the API responses

[✓ Integer] 1, 2, 3, 5, 8, 13, ...
Decimal 3.14159265389

These settings cannot be changed later.

Figure 4-14. *Creating the time-required field*

Next, create the Timings and Entry Fees fields. Both of them will consist of short text and will not be required. This is shown in Figure 4-15.

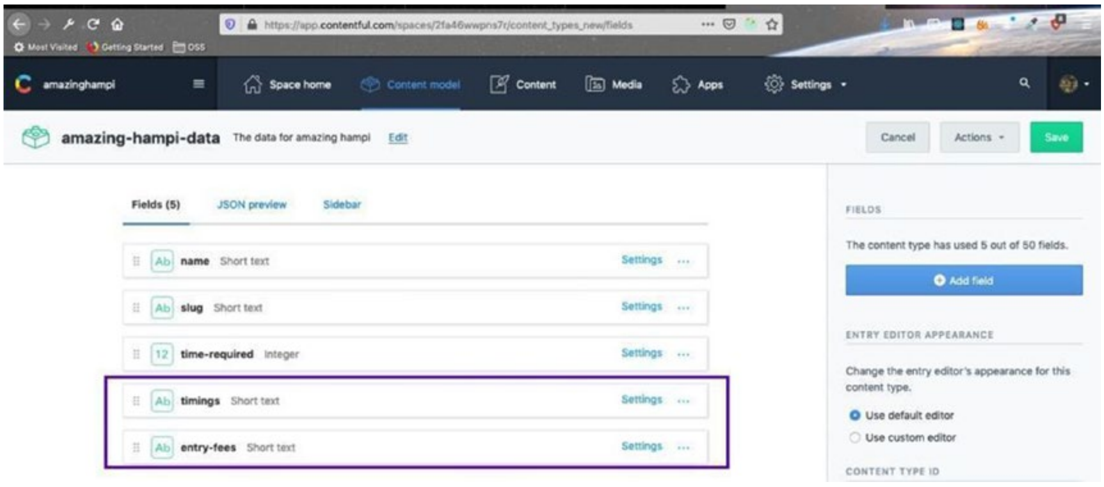
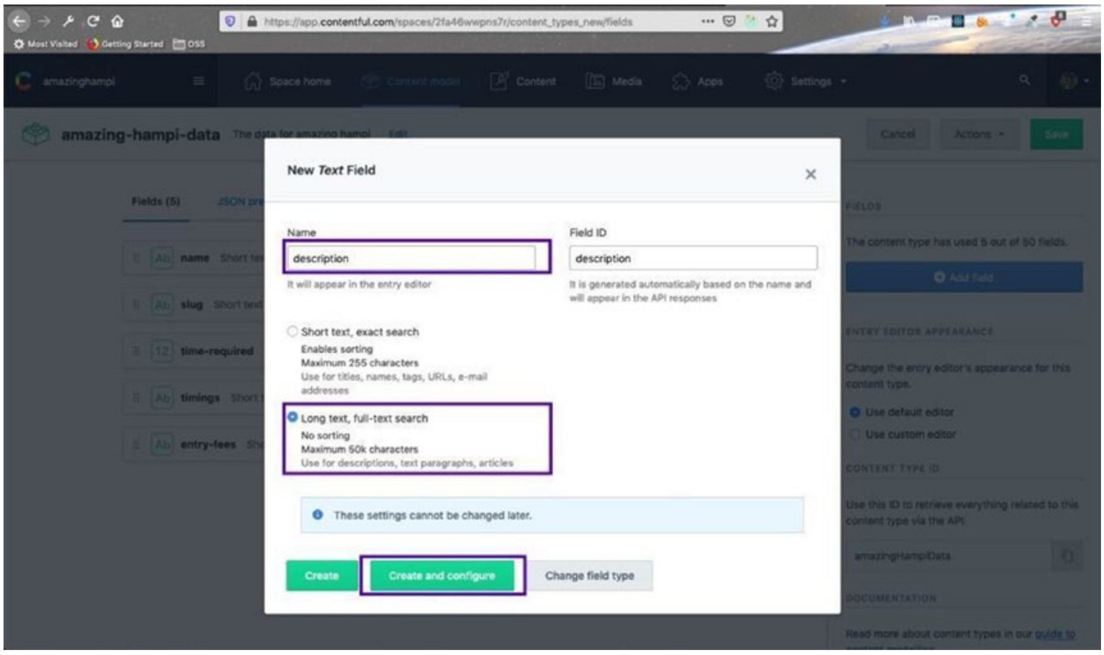


Figure 4-15. *The timings and entry-fees fields*

Next, create the Description field, as shown in Figure 4-16. It will be long text and will be required.


New *Text* Field

Name: description
It will appear in the entry editor

Field ID: description
It is generated automatically based on the name and will appear in the API responses

○ Short text, exact search
Enables sorting
Maximum 255 characters
Use for titles, names, tags, URLs, e-mail addresses

● Long text, full-text search
No sorting
Maximum 50k characters
Use for descriptions, text paragraphs, articles

These settings cannot be changed later.

Figure 4-16. *The Description field*

On the home page, we will have Featured Places, which will contain the top three places to visit in Hampi. Let's create a Boolean field for this. Click Add Field and choose Boolean, as shown in Figure 4-17.

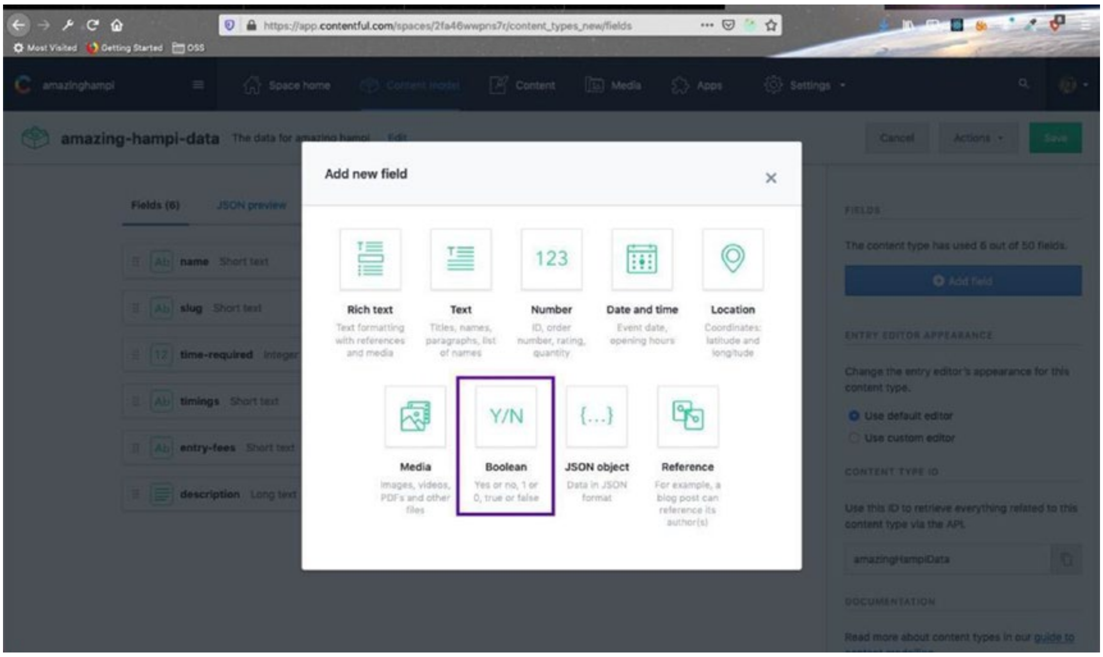


Figure 4-17. *Adding a Boolean field*

Give it the name `featured` and click Create and Configure to make it a required field, as shown in Figure 4-18.

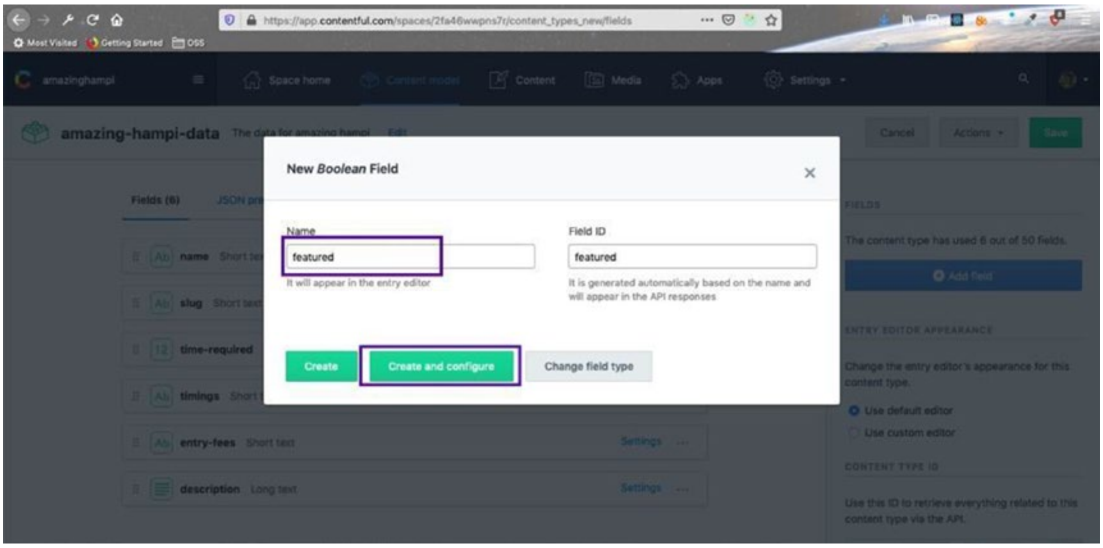

New Boolean Field
Name
featured
It will appear in the entry editor
Field ID
featured
It is generated automatically based on the name and will appear in the API responses
Create
Create and configure
Change field type

Figure 4-18. *The featured field is a Boolean*

Next, we will create the field for the images. It will hold one or more images, required for a place. Click Add Field and then choose Media, as shown in Figure 4-19.

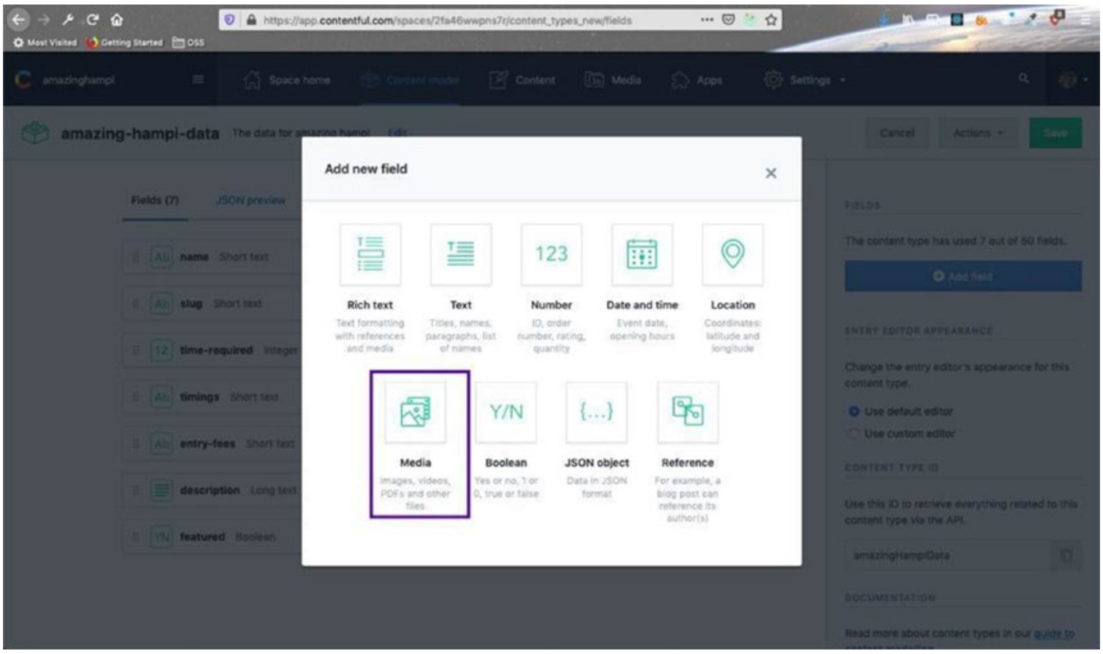
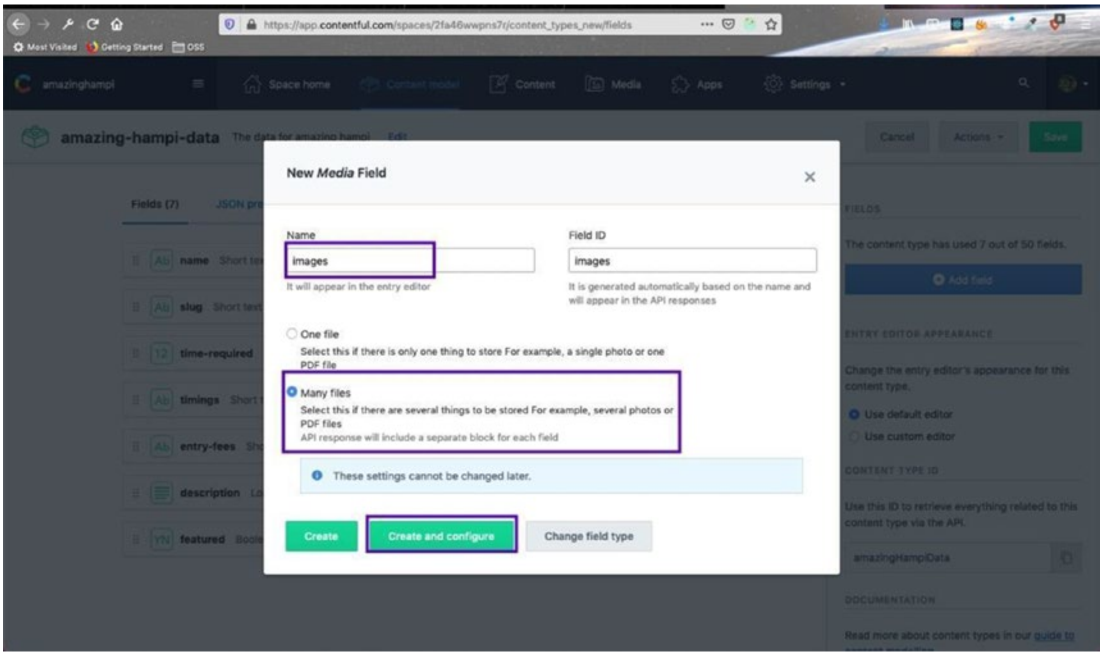

Add new field
Rich text
Text formatting with references and media
Text
Titles, names, paragraphs, list of names
Number
ID, order number, rating, quantity
Date and time
Event date, opening hours
Location
Coordinates: latitude and longitude
Media
Images, videos, PDFs and other files
Boolean
Yes or no, 1 or 0, true or false
JSON object
Data in JSON format
Reference
For example, a blog post can reference its author(s)

Figure 4-19. *Media is selected*

Call this field `images`, then select the radio button for Many Files. Then click Create and Configure to make it a required field, before saving. This process is shown in Figure 4-20.


New Media Field

Name: images
It will appear in the entry editor

Field ID: images
It is generated automatically based on the name and will appear in the API responses

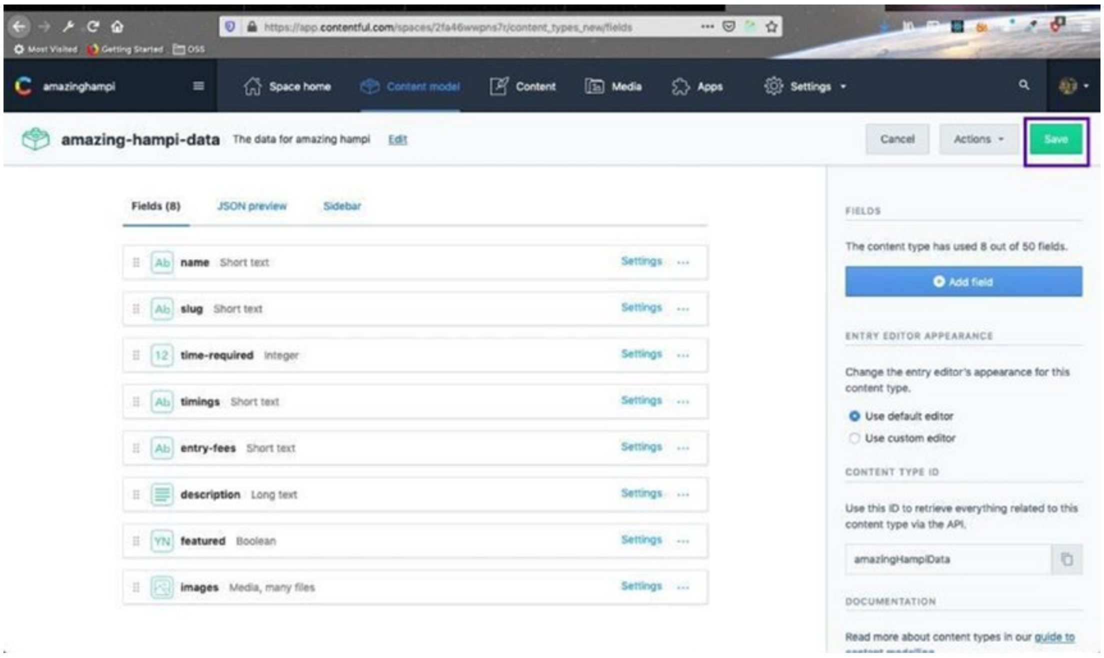
One file
Select this if there is only one thing to store For example, a single photo or one PDF file

[✓ Many files]
Select this if there are several things to be stored For example, several photos or PDF files
API response will include a separate block for each field

These settings cannot be changed later.

Figure 4-20. *Creating a Media field called images*

We are done with all the models. Click the Save button on the top-right corner, as shown in Figure 4-21.



amazing-hampi-data The data for amazing hampi Edit

name	Short text	Settings
slug	Short text	Settings
time-required	Integer	Settings
timings	Short text	Settings
entry-fees	Short text	Settings
description	Long text	Settings
featured	Boolean	Settings
images	Media, many files	Settings

FIELDS

The content type has used 8 out of 50 fields.

ENTRY EDITOR APPEARANCE

Change the entry editor's appearance for this content type.

[✓ Use default editor]

Use custom editor

CONTENT TYPE ID

Use this ID to retrieve everything related to this content type via the API.

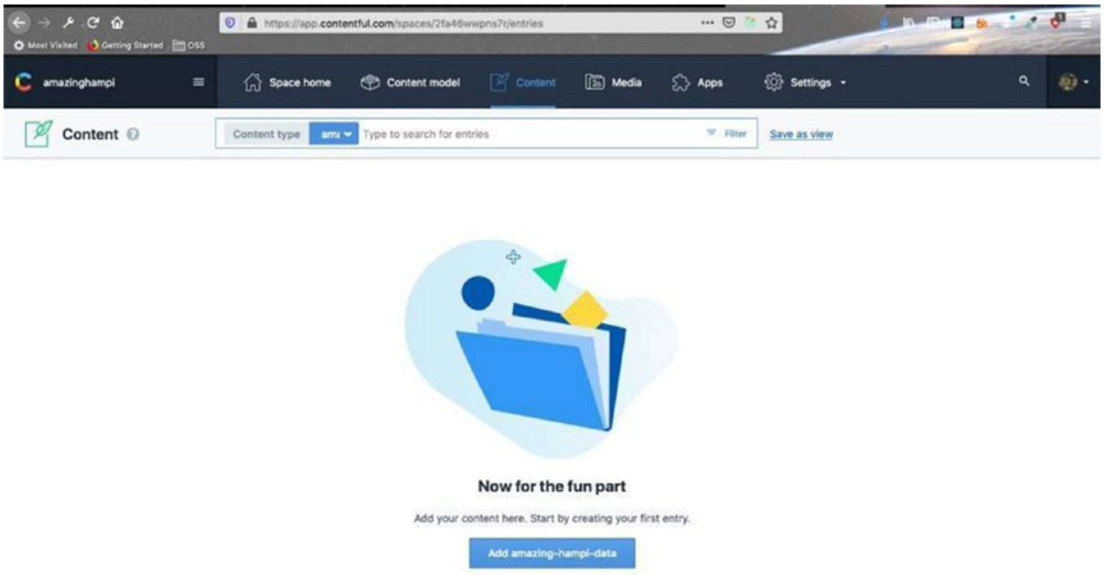
amazingHampiData

DOCUMENTATION

Read more about content types in our guide to

Figure 4-21. *Save your efforts*

It's time to add some content. Head over to the Content tab. You will see the screen shown in Figure 4-22. Click the Add amazing-hampi-data button.

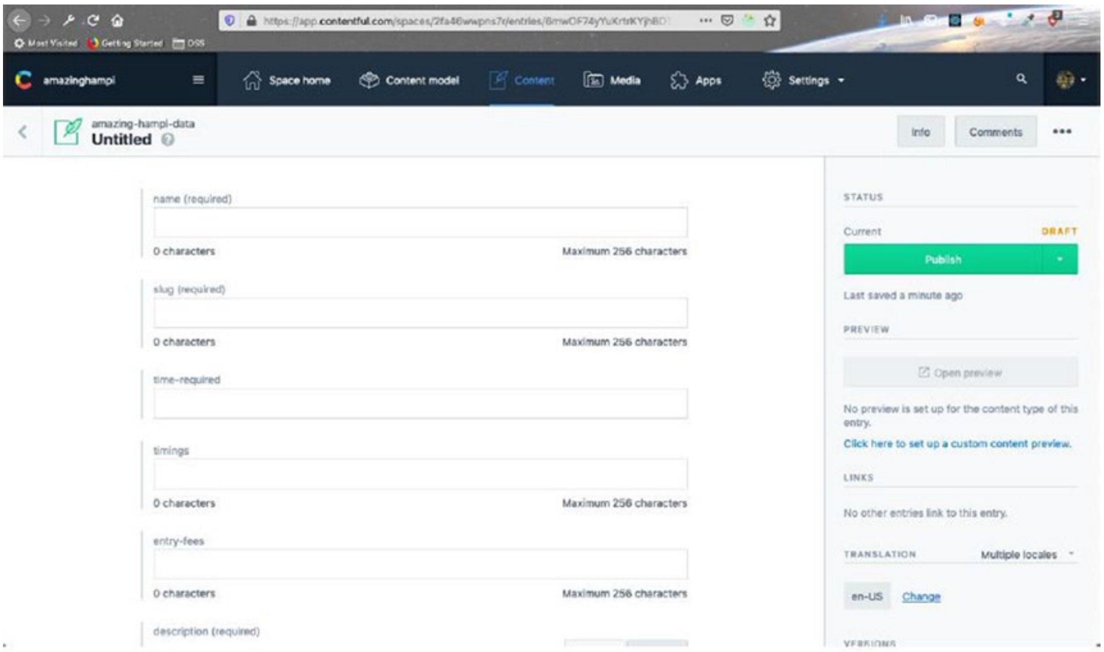


Now for the fun part

Add your content here. Start by creating your first entry.

Figure 4-22. *Adding data*

We will add data from the next screen, as shown in Figure 4-23.



https://app.contentful.com/spaces/2fa46wwpns7r/entries/6mwOF74yYuKrtrKYjhBD

Last saved a minute ago

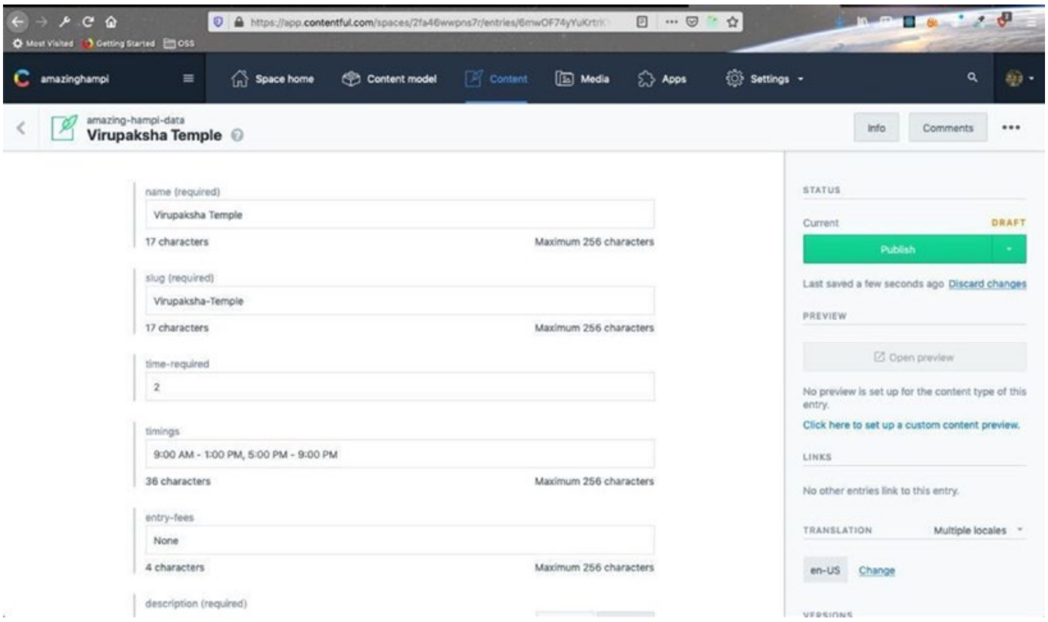
No preview is set up for the content type of this entry.

Click here to set up a custom content preview.

No other entries link to this entry.

Figure 4-23. *Adding data*

Let’s add some content about our first place in Hampi, as shown in Figure 4-24.



name (required)
Virupaksha Temple
17 characters Maximum 256 characters

slug (required)
Virupaksha-Temple
17 characters Maximum 256 characters

time-required
2

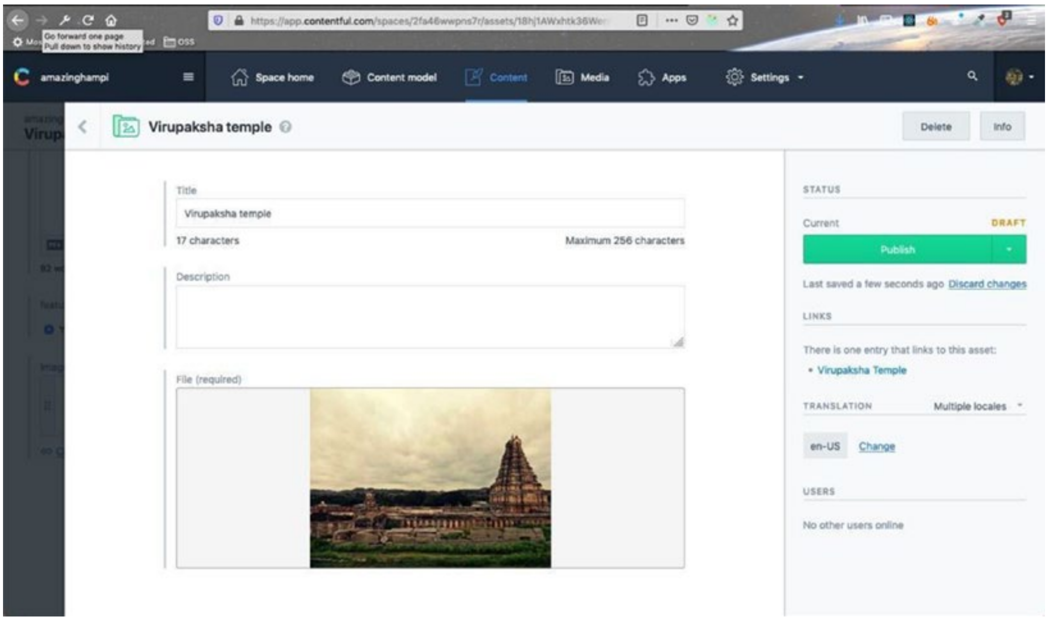
timings
9:00 AM - 1:00 PM, 5:00 PM - 9:00 PM
36 characters Maximum 256 characters

entry-fees
None
4 characters Maximum 256 characters

description (required)

Figure 4-24. *Adding content*

To add images, we select the image and then click the Publish button, as shown in Figure 4-25.



Title
Virupaksha temple
17 characters Maximum 256 characters

Description

File (required)

Figure 4-25. *Publishing images*

I added two images, as shown in Figure 4-26.

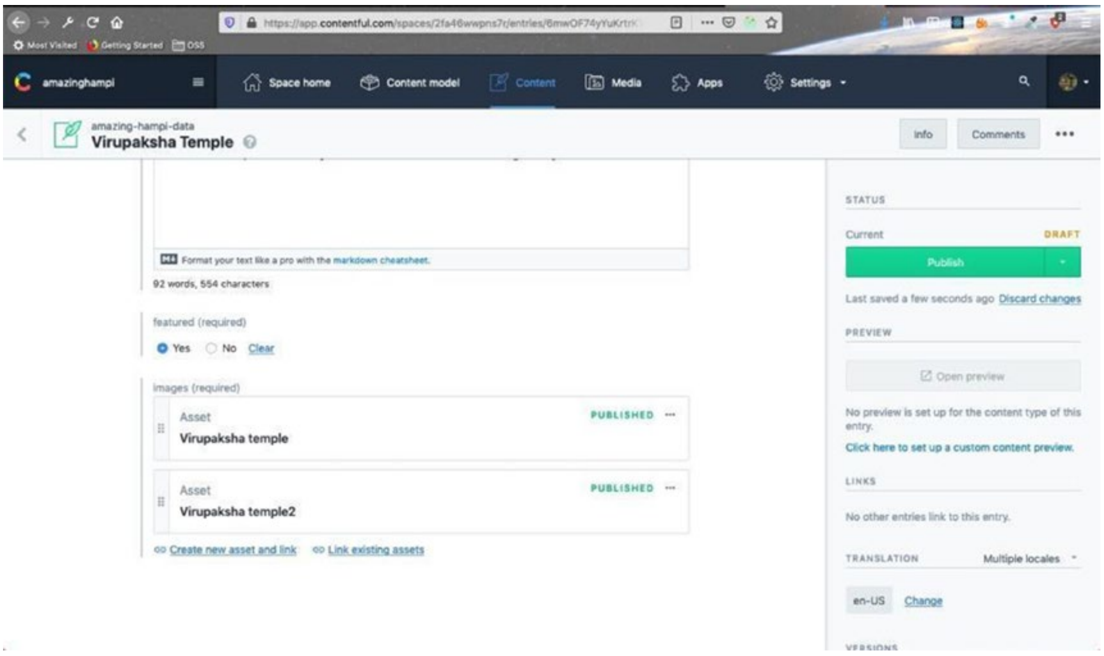
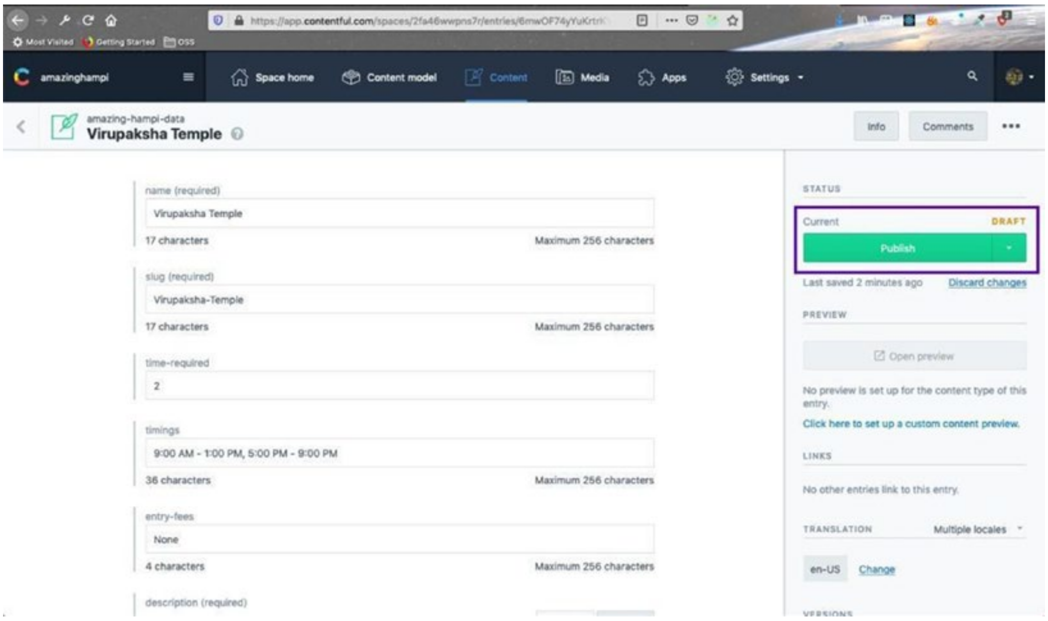


Figure 4-26. *Two images have been added*

Publish this on the main screen by clicking the Publish button, as shown in Figure 4-27.



https://app.contentful.com/spaces/2fa46wwpns7r/entries/6mwOF74yYuKrtrK

amazing-hampi-data
Virupaksha Temple

name (required)
Virupaksha Temple
17 characters   Maximum 256 characters

slug (required)
Virupaksha-Temple
17 characters   Maximum 256 characters

time-required
2

timings
9:00 AM - 1:00 PM, 5:00 PM - 9:00 PM
36 characters   Maximum 256 characters

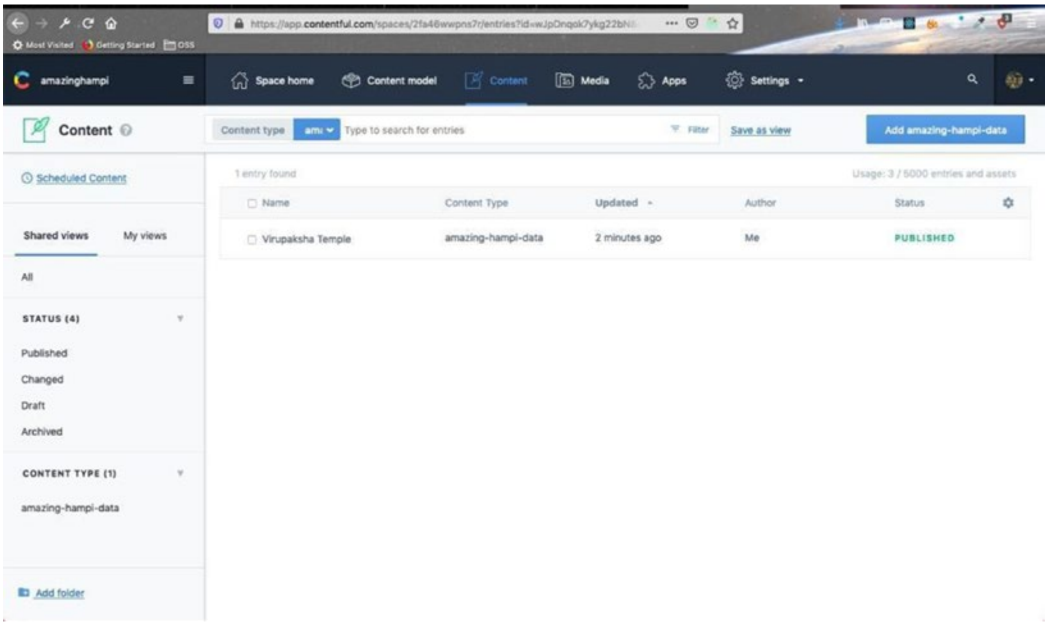
entry-fees
None
4 characters   Maximum 256 characters

description (required)

STATUS
Current   DRAFT
Last saved 2 minutes ago
PREVIEW
No preview is set up for the content type of this entry.
LINKS
No other entries link to this entry.
TRANSLATION   Multiple locales
en-US

Figure 4-27. *Published*

Contentful will show these published items in the Content screen, as shown in Figure 4-28.



https://app.contentful.com/spaces/2fa46wwpns7r/entries?id=wJpOnqok7ykg22bN

Content

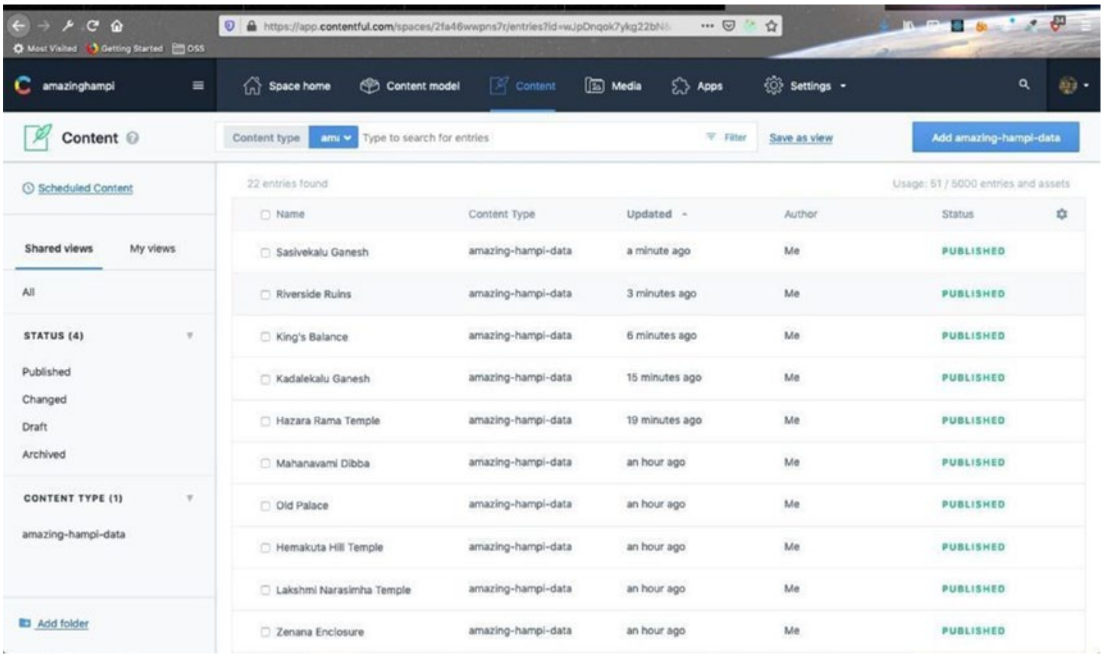
All
STATUS (4)
Published
Changed
Draft
Archived
CONTENT TYPE (1)
amazing-hampi-data

1 entry found   Usage: 3 / 5000 entries and assets

Name	Content Type	Updated	Author	Status
Virupaksha Temple	amazing-hampi-data	2 minutes ago	Me	PUBLISHED

Figure 4-28. *Published content*

Next, I will add more places to visit in Hampi by following the same steps. In fact, I added entries about 22 places in Contentful, as shown in Figure 4-29. You can add your own data.



22 entries found

Usage: 51 / 5000 entries and assets

Name	Content Type	Updated	Author	Status
Sasivekalu Ganesh	amazing-hampi-data	a minute ago	Me	PUBLISHED
Riverside Ruins	amazing-hampi-data	3 minutes ago	Me	PUBLISHED
King's Balance	amazing-hampi-data	6 minutes ago	Me	PUBLISHED
Kadalekalu Ganesh	amazing-hampi-data	15 minutes ago	Me	PUBLISHED
Hazara Rama Temple	amazing-hampi-data	19 minutes ago	Me	PUBLISHED
Mahanavami Dibba	amazing-hampi-data	an hour ago	Me	PUBLISHED
Old Palace	amazing-hampi-data	an hour ago	Me	PUBLISHED
Hemakuta Hill Temple	amazing-hampi-data	an hour ago	Me	PUBLISHED
Lakshmi Narasimha Temple	amazing-hampi-data	an hour ago	Me	PUBLISHED
Zenana Enclosure	amazing-hampi-data	an hour ago	Me	PUBLISHED

Figure 4-29. *Places to visit in Hampi added as content*

Install the Gatsby Plugins

Next, we will import this data into our site. To do this, we will install the `gatsby-source-contentful` plugin. The docs can be found here[2].

As usual, we need to first `npm install` in our project directory, with the following command.

```
npm install --save gatsby-source-contentful
```

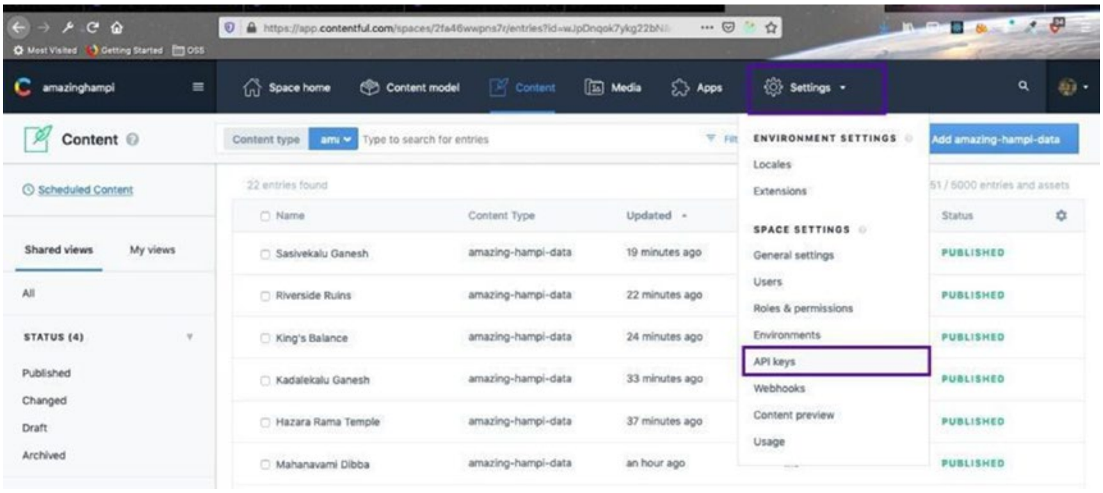
Next, in `gatsby-config.js` we had to add the plugin, which is highlighted in bold in Listing 4-1. As per the docs, let's add the object.

[2]https://www.gatsbyjs.org/packages/gatsby-source-contentful/

Listing 4-1. gatsby-config.js

```
module.exports = {
            plugins: [
            {
      resolve: `gatsby-source-filesystem`,
      options: {
        name: `images`,
        path: `${__dirname}/src/images/`,
      },
    },
          {
      resolve: `gatsby-source-contentful`,
      options: {
        spaceId: `your_space_id`,
 accessToken: process.env.CONTENTFUL_ACCESS_TOKEN,
      },
    },
            `gatsby-plugin-styled-components`,
            `gatsby-transformer-sharp`,
            `gatsby-plugin-sharp`,
            `gatsby-plugin-transition-link`
             ]
}
```

Let’s get our API keys from Contentful. From the Contentful API, choose Settings ➤ API Keys, as shown in Figure 4-30.



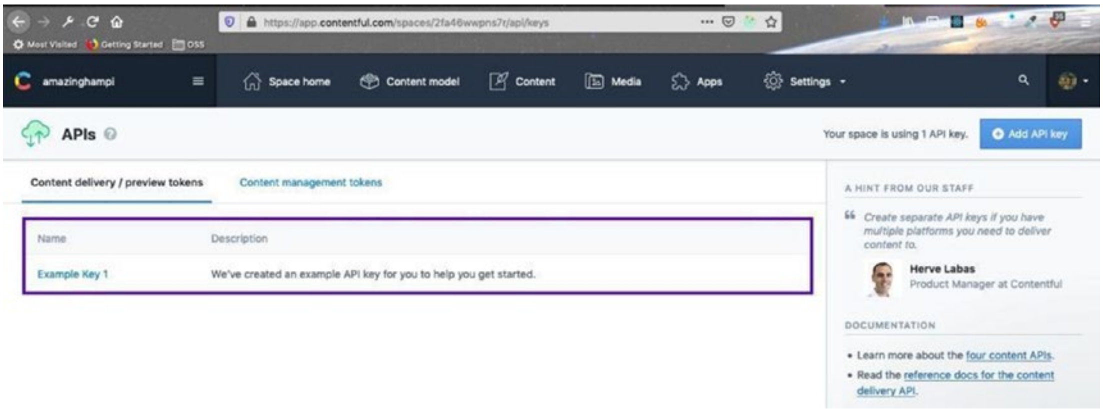
22 entries found

51 / 5000 entries and assets

Name	Content Type	Updated	Status
Sasivekalu Ganesh	amazing-hampi-data	19 minutes ago	PUBLISHED
Riverside Ruins	amazing-hampi-data	22 minutes ago	PUBLISHED
King's Balance	amazing-hampi-data	24 minutes ago	PUBLISHED
Kadalekalu Ganesh	amazing-hampi-data	33 minutes ago	PUBLISHED
Hazara Rama Temple	amazing-hampi-data	37 minutes ago	PUBLISHED
Mahanavami Dibba	amazing-hampi-data	an hour ago	PUBLISHED

Figure 4-30. *Getting the API keys*

It looks like a key has been created for us, as shown in Figure 4-31.



Your space is using 1 API key.

Name	Description
Example Key 1	We've created an example API key for you to help you get started.

A HINT FROM OUR STAFF

"Create separate API keys if you have multiple platforms you need to deliver content to.

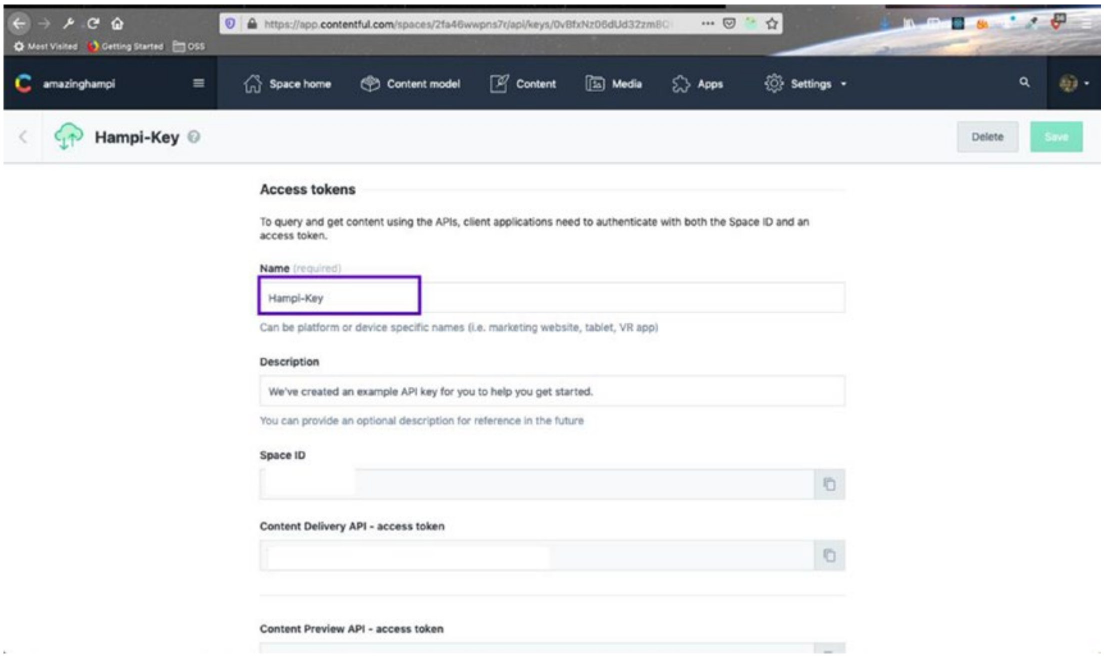
Herve Labas
Product Manager at Contentful

DOCUMENTATION

- Learn more about the four content APIs.
- Read the reference docs for the content delivery API.

Figure 4-31. *Example key 1*

Click Example Key 1 shown in Figure 4-31 to edit the name of the key. We need to note our Space ID and Content Delivery API - Access Token from here (see Figure 4-32).



Access tokens

To query and get content using the APIs, client applications need to authenticate with both the Space ID and an access token.

Name (required)

Hampi-Key

Can be platform or device specific names (i.e. marketing website, tablet, VR app)

Description

We've created an example API key for you to help you get started.

You can provide an optional description for reference in the future

Space ID

[redacted]

Content Delivery API - access token

[redacted]

Content Preview API - access token

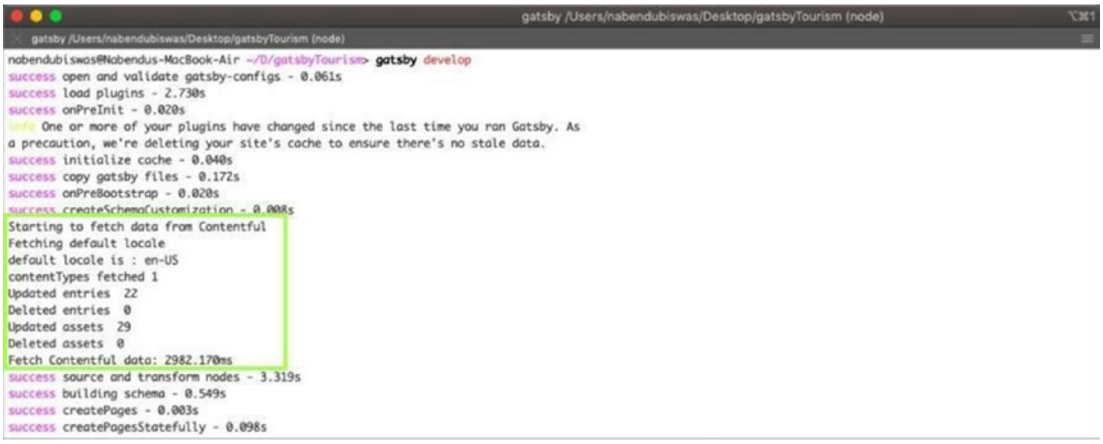
Figure 4-32. *Make note of the Space ID and Content Delivery API - Access Token*

Head over to your code and add these two numbers, as shown in bold in Listing 4-2. We are not going to keep these here and will soon move them to an environment variable, as we don't need to push these keys to GitHub and have everyone see them.

Listing 4-2. The gatsby-config.js File

```
module.exports = {
            plugins: [
            {
      resolve: `gatsby-source-filesystem`,
      options: {
        name: `images`,
        path: `${__dirname}/src/images/`,
      },
    },
        {
      resolve: `gatsby-source-contentful`,
      options: {
        spaceId: `2XXXXXXXXX2`,
        accessToken: `RXXXXXXXXXXXXXXXXXXXXE`,
      },
    },
            `gatsby-plugin-styled-components`,
            `gatsby-transformer-sharp`,
            `gatsby-plugin-sharp`,
            `gatsby-plugin-transition-link`
            ]
}
```

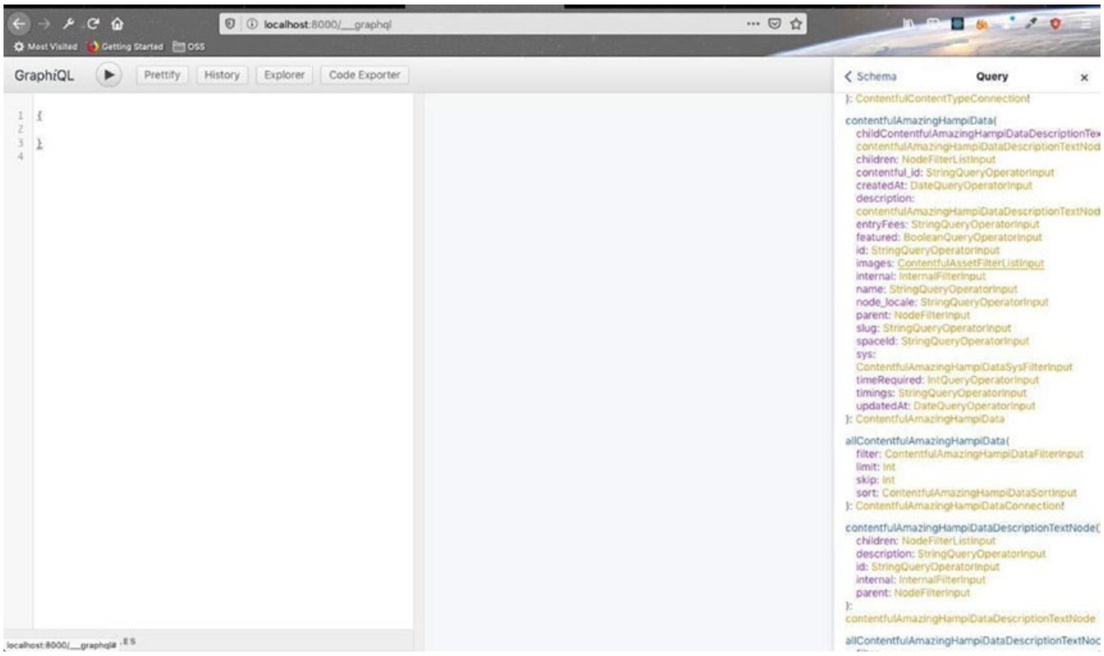
Let's head over to our terminal to check whether the setup was right, by running `gatsby develop`. The command runs without error and the connection also fetches the correct data from Contentful, as shown in Figure 4-33.



```
nabendubiswas@Nabendus-MacBook-Air ~/D/gatsbyTourism> gatsby develop
success open and validate gatsby-configs - 0.061s
success load plugins - 2.730s
success onPreInit - 0.020s
info One or more of your plugins have changed since the last time you ran Gatsby. As
a precaution, we're deleting your site's cache to ensure there's no stale data.
success initialize cache - 0.040s
success copy gatsby files - 0.172s
success onPreBootstrap - 0.020s
success createSchemaCustomization - 0.008s
Starting to fetch data from Contentful
Fetching default locale
default locale is : en-US
contentTypes fetched 1
Updated entries  22
Deleted entries  0
Updated assets  29
Deleted assets  0
Fetch Contentful data: 2982.170ms
success source and transform nodes - 3.319s
success building schema - 0.549s
success createPages - 0.003s
success createPagesStatefully - 0.098s
```

Figure 4-33. *Fetching data from Contentful*

Let's go to GraphQL and refresh the browser. Click docs to see Contentful's queries, as shown in Figure 4-34.



```
{
}
```

```
): ContentfulContentTypeConnection!

contentfulAmazingHampiData(
   childContentfulAmazingHampiDataDescriptionTex
   contentfulAmazingHampiDataDescriptionTextNod
   children: NodeFilterListInput
   contentful_id: StringQueryOperatorInput
   createdAt: DateQueryOperatorInput
   description:
   contentfulAmazingHampiDataDescriptionTextNod
   entryFees: StringQueryOperatorInput
   featured: BooleanQueryOperatorInput
   id: StringQueryOperatorInput
   images: ContentfulAssetFilterListInput
   internal: InternalFilterInput
   name: StringQueryOperatorInput
   node_locale: StringQueryOperatorInput
   parent: NodeFilterInput
   slug: StringQueryOperatorInput
   spaceId: StringQueryOperatorInput
   sys:
   ContentfulAmazingHampiDataSysFilterInput
   timeRequired: IntQueryOperatorInput
   timings: StringQueryOperatorInput
   updatedAt: DateQueryOperatorInput
): ContentfulAmazingHampiData

allContentfulAmazingHampiData(
   filter: ContentfulAmazingHampiDataFilterInput
   limit: Int
   skip: Int
   sort: ContentfulAmazingHampiDataSortInput
): ContentfulAmazingHampiDataConnection!

contentfulAmazingHampiDataDescriptionTextNode(
   children: NodeFilterListInput
   description: StringQueryOperatorInput
   id: StringQueryOperatorInput
   internal: InternalFilterInput
   parent: NodeFilterInput
):
contentfulAmazingHampiDataDescriptionTextNode

allContentfulAmazingHampiDataDescriptionTextNoc
```

Figure 4-34. *GraphQL*

Next, we will use environment variables to store the `spaceId` and `accessToken`. For that, we need to install the `dotenv` package. Head over to your terminal and stop `gatsby develop`. Then `npm install` the package, using the following command.

```
npm install --save dotenv
```

As per the docs,[3] we need to add the lines in Listing 4-3 to our `gatsby-config.js` file.

Listing 4-3. The gatsby-config.js File

```
require("dotenv").config({
  path: `.env.${process.env.NODE_ENV}`,
})
```

[3] https://www.gatsbyjs.org/docs/environment-variables/

```
module.exports = {
        plugins: [
                       ...
                       ...
          ]
}
```

We need to create an `.env.development` file in the root directory. Then take the keys from `gatsby-config.js` and add them (without quotation marks) to the two variables, as shown in Listing 4-4.

Listing 4-4. The env.development File

```
          CONTENTFUL_SPACE_ID=2XXXXXXXXX2
          CONTENTFUL_ACCESS_TOKEN=RXXXXXXXXXXXXXXXXXXXXE
```

Next, in `gatsby-config.js`, add this using `process.env`, as highlighted in Listing 4-5.

Listing 4-5. The gatsby-config.js File

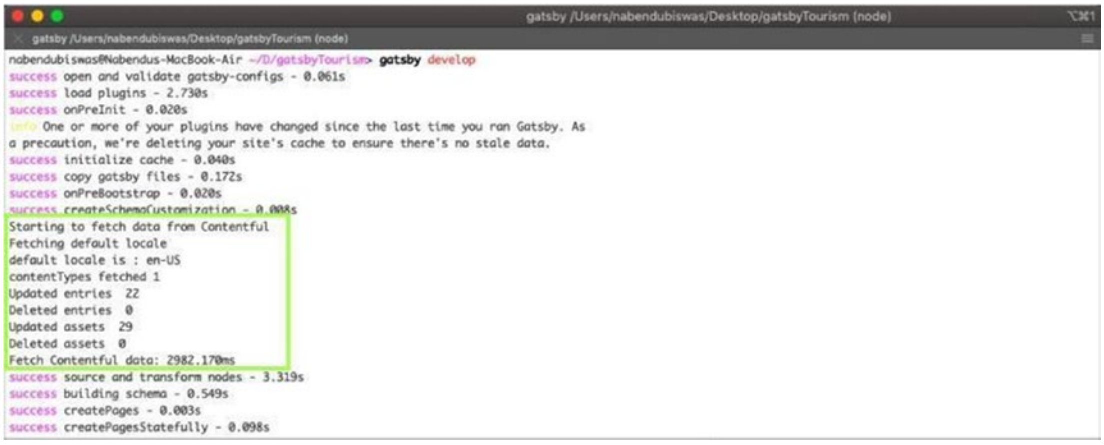
```
require("dotenv").config({
  path: `.env.${process.env.NODE_ENV}`,
})

module.exports = {
            plugins: [
            ...
            ...
            {
      resolve: `gatsby-source-contentful`,
      options: {
        spaceId: process.env.CONTENTFUL_SPACE_ID,
        accessToken: process.env.CONTENTFUL_ACCESS_TOKEN,
      },
    },
            ...
            ...
            ]
}
```

Head over to `.gitignore` and add the `.env.development` file to it, as highlighted in Listing 4-6.

Listing 4-6. The .gitignore File

```
# Logs
logs
*.log
npm-debug.log*
yarn-debug.log*
yarn-error.log*
.env.development
# Runtime data
pids
*.pid
*.seed
*.pid.lock
...
...
```

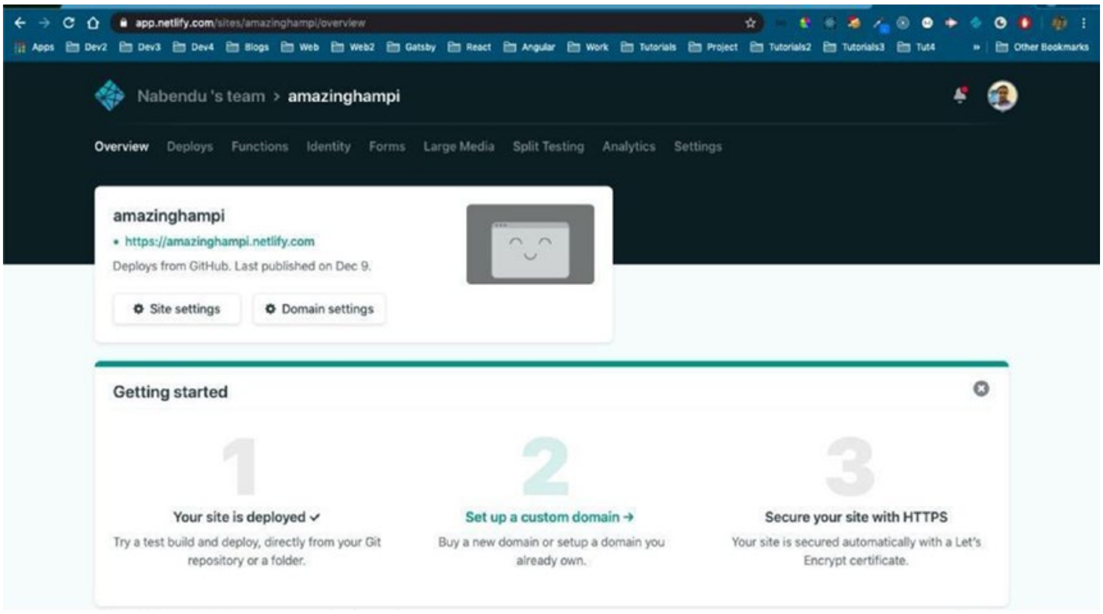
Once again, head over to the terminal and run `gatsby develop`, to check if all runs well. Figure 4-35 shows that it's running well.



```
gatsby /Users/nabendubiswas/Desktop/gatsbyTourism (node)
gatsby /Users/nabendubiswas/Desktop/gatsbyTourism (node)
nabendubiswas@Nabendus-MacBook-Air ~/D/gatsbyTourism> gatsby develop
success open and validate gatsby-configs - 0.061s
success load plugins - 2.730s
success onPreInit - 0.020s
info One or more of your plugins have changed since the last time you ran Gatsby. As
a precaution, we're deleting your site's cache to ensure there's no stale data.
success initialize cache - 0.040s
success copy gatsby files - 0.172s
success onPreBootstrap - 0.020s
success createSchemaCustomization - 0.008s
Starting to fetch data from Contentful
Fetching default locale
default locale is : en-US
contentTypes fetched 1
Updated entries  22
Deleted entries  0
Updated assets  29
Deleted assets  0
Fetch Contentful data: 2982.170ms
success source and transform nodes - 3.319s
success building schema - 0.549s
success createPages - 0.003s
success createPagesStatefully - 0.098s
```

Figure 4-35. *Run gatsby develop to check if it's running well*

Before pushing to GitHub, we need to add these variables to our Netlify deployment. Head over to your Netlify deployment, as shown in Figure 4-36.



amazinghampi

https://amazinghampi.netlify.com

Deploys from GitHub. Last published on Dec 9.

Getting started

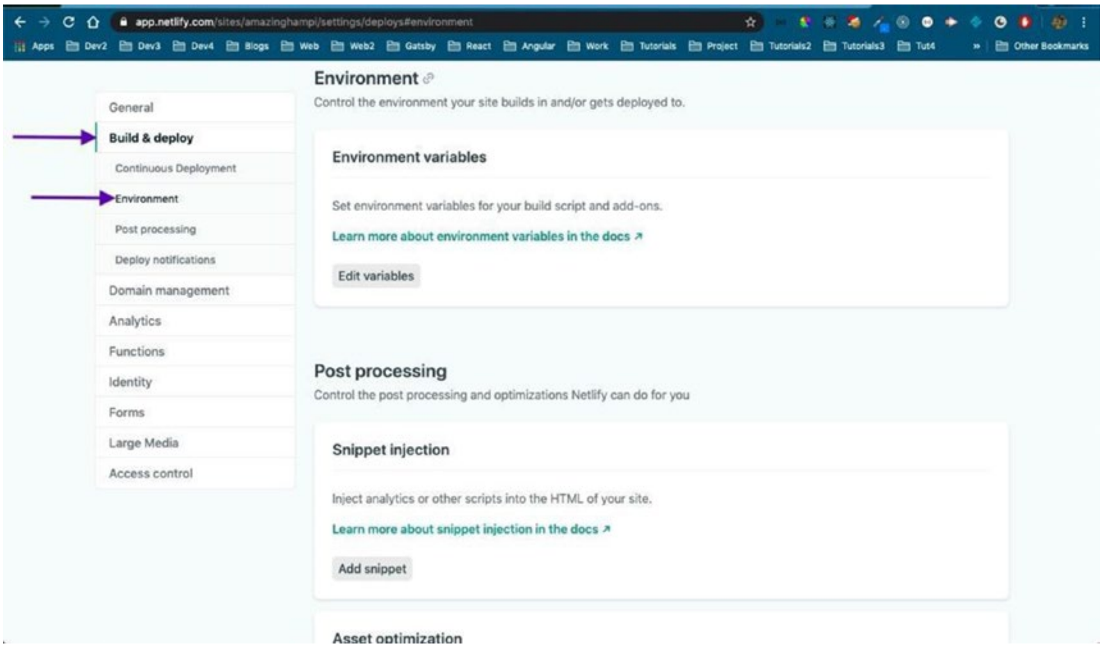
1 Your site is deployed ✓
Try a test build and deploy, directly from your Git repository or a folder.

2 Set up a custom domain →
Buy a new domain or setup a domain you already own.

3 Secure your site with HTTPS
Your site is secured automatically with a Let's Encrypt certificate.

Figure 4-36. *The Netlify deployment*

Click the Site Settings button. On the left menu, click Build & Deploy and then choose Environment, as shown in Figure 4-37.



Environment

Control the environment your site builds in and/or gets deployed to.

Environment variables

Set environment variables for your build script and add-ons.

Learn more about environment variables in the docs ↗

Post processing

Control the post processing and optimizations Netlify can do for you

Snippet injection

Inject analytics or other scripts into the HTML of your site.

Learn more about snippet injection in the docs ↗

Asset optimization

Figure 4-37. *Deploy the environment*

Click Edit Variables and add the two variables. After that, click Save, as shown in Figure 4-38.

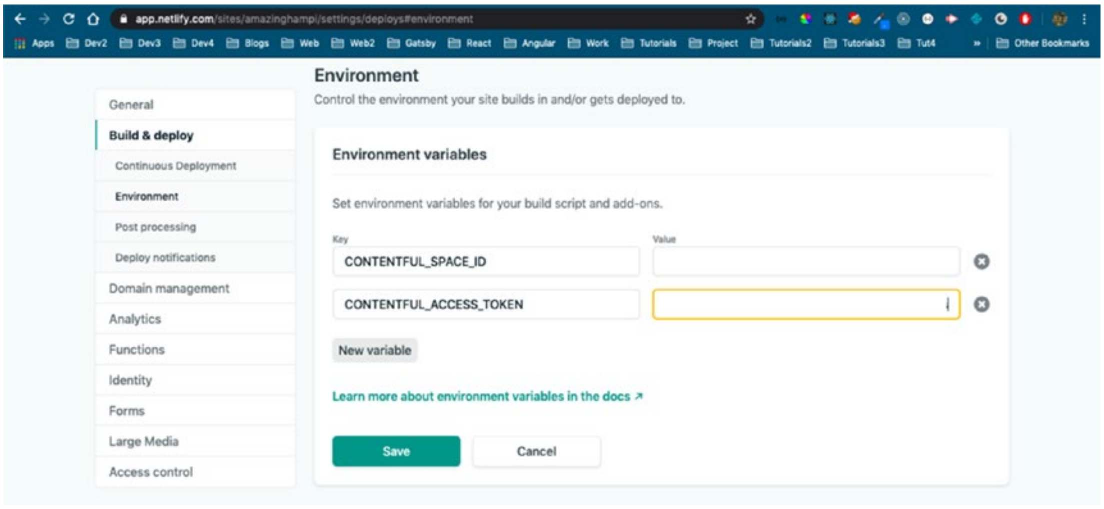


Figure 4-38. *Adding two variables*

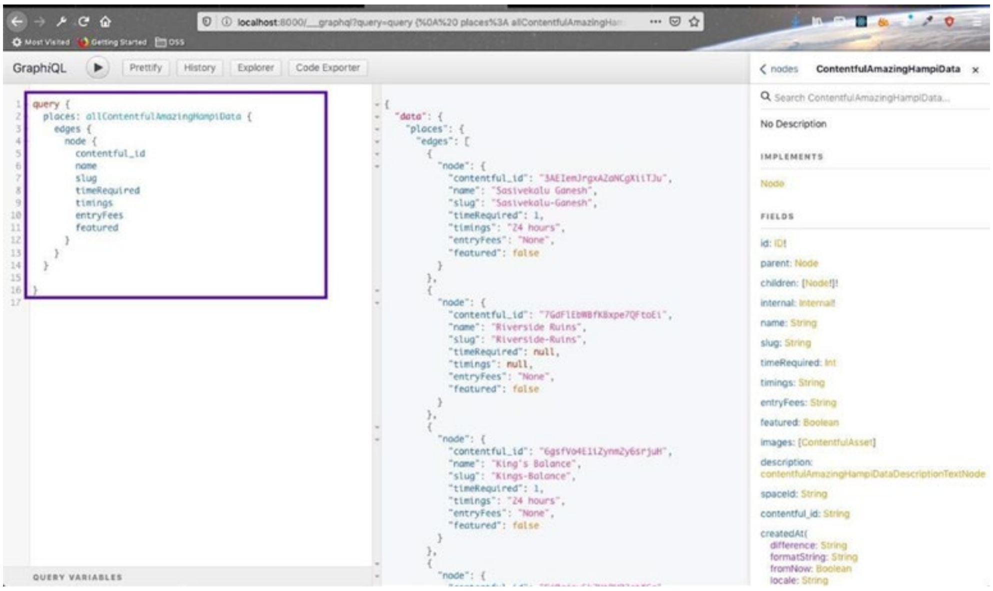
Commit your code and push it to GitHub. You can find my code in my GitHub account here[4]. In addition, because of continuous deployment, it was successfully pushed to the Netlify site[5].

Adding Queries for the Places Component

We will first create the queries in GraphQL for our places. We have a Places page on our project and will also show Featured Places on the home page. Head over to GraphiQL and type the query shown in Figure 4-39 to get all the places. We can also see the details of the query from the docs, on the right side.

[4]https://github.com/nabendu82/gatsbyTourism

[5]https://amazinghampi.netlify.com/


localhost:8000/___graphql?query=query {%0A%20 places%3A allContentfulAmazingHa
Most Visited
Getting Started
OSS
GraphiQL
Prettify
History
Explorer
Code Exporter
QUERY VARIABLES
nodes
ContentfulAmazingHampiData
Search ContentfulAmazingHampiData...

```
query {
  places: allContentfulAmazingHampiData {
    edges {
      node {
        contentful_id
        name
        slug
        timeRequired
        timings
        entryFees
        featured
      }
    }
  }

}
```

```
{
  "data": {
    "places": {
      "edges": [
        {
          "node": {
            "contentful_id": "3AEIemJrgxAZoNCgXiiTJu",
            "name": "Sasivekalu Ganesh",
            "slug": "Sasivekalu-Ganesh",
            "timeRequired": 1,
            "timings": "24 hours",
            "entryFees": "None",
            "featured": false
          }
        },
        {
          "node": {
            "contentful_id": "7GdFlEbWBfKBxpe7QFtoEi",
            "name": "Riverside Ruins",
            "slug": "Riverside-Ruins",
            "timeRequired": null,
            "timings": null,
            "entryFees": "None",
            "featured": false
          }
        },
        {
          "node": {
            "contentful_id": "6gsfVo4E1iZynm2y6srjuH",
            "name": "King's Balance",
            "slug": "Kings-Balance",
            "timeRequired": 1,
            "timings": "24 hours",
            "entryFees": "None",
            "featured": false
          }
        },
        {
          "node": {
```

No Description

IMPLEMENTS

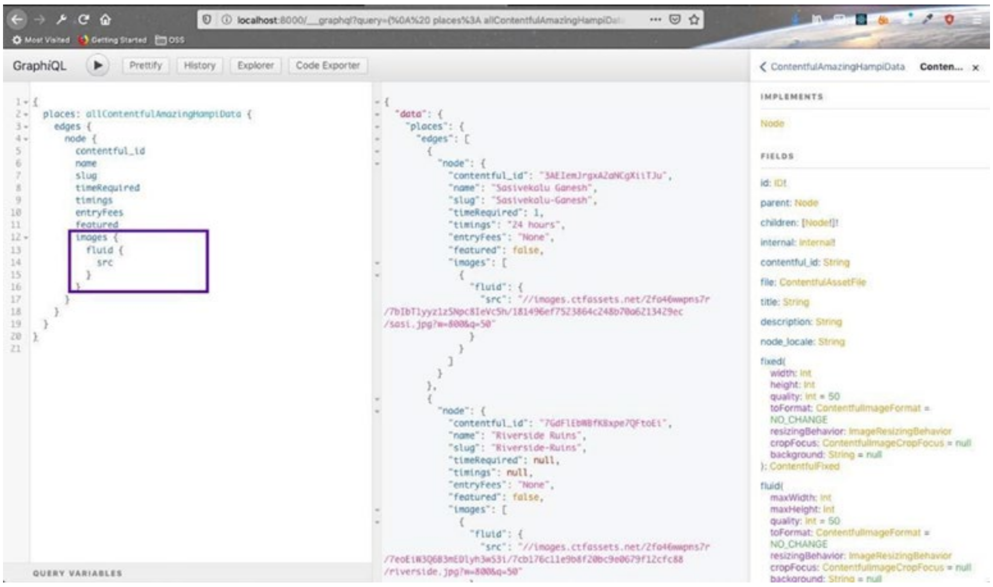
Node

FIELDS

id: ID!
parent: Node
children: [Node!]!
internal: Internal!
name: String
slug: String
timeRequired: Int
timings: String
entryFees: String
featured: Boolean
images: [ContentfulAsset]
description: contentfulAmazingHampiDataDescriptionTextNode
spaceId: String
contentful_id: String
createdAt(
  difference: String
  formatString: String
  fromNow: Boolean
  locale: String

Figure 4-39. *Using GraphiQL*

Let's also search for images. As per the docs, we can have fluid or fixed images. But we cannot use the fragment in GraphiQL as usual and we will use `src`, which we are going to later change in the code. This is shown in Figure 4-40.


localhost:8000/___graphql?query={%0A%20 places%3A allContentfulAmazingHampiDa
Most Visited
Getting Started
OSS
GraphiQL
Prettify
History
Explorer
Code Exporter
QUERY VARIABLES
ContentfulAmazingHampiData
Conten...

```
{
  places: allContentfulAmazingHampiData {
    edges {
      node {
        contentful_id
        name
        slug
        timeRequired
        timings
        entryFees
        featured
        images {
          fluid {
            src
          }
        }
      }
    }
  }
}
```

```
{
  "data": {
    "places": {
      "edges": [
        {
          "node": {
            "contentful_id": "3AEIemJrgxAZoNCgXiiTJu",
            "name": "Sasivekalu Ganesh",
            "slug": "Sasivekalu-Ganesh",
            "timeRequired": 1,
            "timings": "24 hours",
            "entryFees": "None",
            "featured": false,
            "images": [
              {
                "fluid": {
                  "src": "//images.ctfassets.net/2fo46wwpns7r
/7bIbTlyyz1zSNpc8IeVc5h/181496ef7523864c248b70a6213429ec
/sasi.jpg?w=800&q=50"
                }
              }
            ]
          }
        },
        {
          "node": {
            "contentful_id": "7GdFlEbWBfKBxpe7QFtoEi",
            "name": "Riverside Ruins",
            "slug": "Riverside-Ruins",
            "timeRequired": null,
            "timings": null,
            "entryFees": "None",
            "featured": false,
            "images": [
              {
                "fluid": {
                  "src": "//images.ctfassets.net/2fo46wwpns7r
/7eoEiW3Q683mE0lyh3w53i/7cb176c11e9b8f20bc9e0679f12cfc88
/riverside.jpg?w=800&q=50"
```

IMPLEMENTS

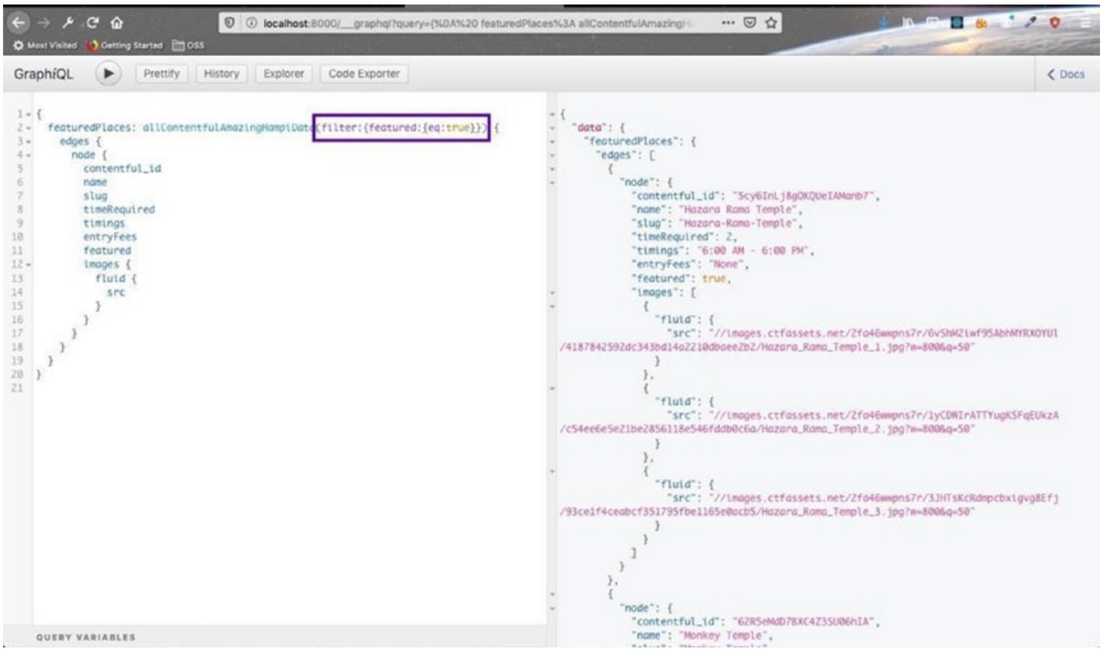
Node

FIELDS

id: ID!
parent: Node
children: [Node!]!
internal: Internal!
contentful_id: String
file: ContentfulAssetFile
title: String
description: String
node_locale: String
fixed(
  width: Int
  height: Int
  quality: Int = 50
  toFormat: ContentfulImageFormat = NO_CHANGE
  resizingBehavior: ImageResizingBehavior
  cropFocus: ContentfulImageCropFocus = null
  background: String = null
): ContentfulFixed
fluid(
  maxWidth: Int
  maxHeight: Int
  quality: Int = 50
  toFormat: ContentfulImageFormat = NO_CHANGE
  resizingBehavior: ImageResizingBehavior
  cropFocus: ContentfulImageCropFocus = null
  background: String = null

Figure 4-40. *Searching for images*

Let's also add a query for Featured Places, as shown in Figure 4-41. Here we are using the `filter` option to get only the featured places (there are four).



```
localhost:8000/___graphql?query={%0A%20 featuredPlaces%3A allContentfulAmazingH
```

```
{
  featuredPlaces: allContentfulAmazingHampiData(filter:{featured:{eq:true}}) {
    edges {
      node {
        contentful_id
        name
        slug
        timeRequired
        timings
        entryFees
        featured
        images {
          fluid {
            src
          }
        }
      }
    }
  }
}
```

```
{
  "data": {
    "featuredPlaces": {
      "edges": [
        {
          "node": {
            "contentful_id": "5cy6InLj8gOKQUeIAMonb7",
            "name": "Hazara Rama Temple",
            "slug": "Hazara-Rama-Temple",
            "timeRequired": 2,
            "timings": "6:00 AM - 6:00 PM",
            "entryFees": "None",
            "featured": true,
            "images": [
              {
                "fluid": {
                  "src": "//images.ctfassets.net/2fo46wwpns7r/6vShM2iwf95AbhMYRXOYU1
/4187842592dc343bd14a2210dbaee2b2/Hazara_Rama_Temple_1.jpg?w=800&q=50"
                }
              },
              {
                "fluid": {
                  "src": "//images.ctfassets.net/2fo46wwpns7r/1yCDWIrATTYugKSFqEUkzA
/c54ee6e5e21be2856118e546fddb0c6a/Hazara_Rama_Temple_2.jpg?w=800&q=50"
                }
              },
              {
                "fluid": {
                  "src": "//images.ctfassets.net/2fo46wwpns7r/3JHTsKcRdmpcbxigvg8Efj
/93ce1f4ceabcf351795fbe1165e0acb5/Hazara_Rama_Temple_3.jpg?w=800&q=50"
                }
              }
            ]
          }
        },
        {
          "node": {
            "contentful_id": "62R5eMdD7BXC4Z3SU06hIA",
            "name": "Monkey Temple",
```

Figure 4-41. *Getting only the featured places*

Next, move to your code editor and create a folder called `places` inside the `components` folder. Create three files—`Places.js`, `PlaceList.js`, and `Place.js`. Also, create a file called `FeaturedPlaces.js` inside the `home` folder. Add an `items.module.css` file to the `css` folder. The contents of this file are shown Listing 4-7.

Listing 4-7. The items.module.css File

```
.places {
  padding: 4rem 0;
  text-align: center;
}
.sub-text {
  text-transform: uppercase;
  font-size: 1.3rem;
  text-align: center;
  letter-spacing: 7px;
}
.sub-text span{
  color: var(--primaryColor);
}
.center {
  width: 80vw;
  margin: 3rem auto;
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(280px, 1fr));
  grid-column-gap: 2rem;
  grid-row-gap: 2rem;
}
@media screen and (min-width: 576px) {
  .center {
    grid-template-columns: repeat(auto-fill, minmax(368.66px, 1fr));
  }
}
@media screen and (min-width: 1200px) {
  .center {
    width: 100%;
    max-width: 1170px;
  }
}
```

Adding the Featured Places Component

Next, let's update the `FeaturedPlaces.js` file. Here, we will add our query for featured places, which we created in GraphiQL. The only thing that changes is the `GatsbyContent-fulFluid_tracedSVG` fragment, instead of `src`. We also check to see if we are getting data correctly, by placing a `console.log()` in the code. The whole code is shown in Listing 4-8.

Listing 4-8. The FeaturedPlaces.js File

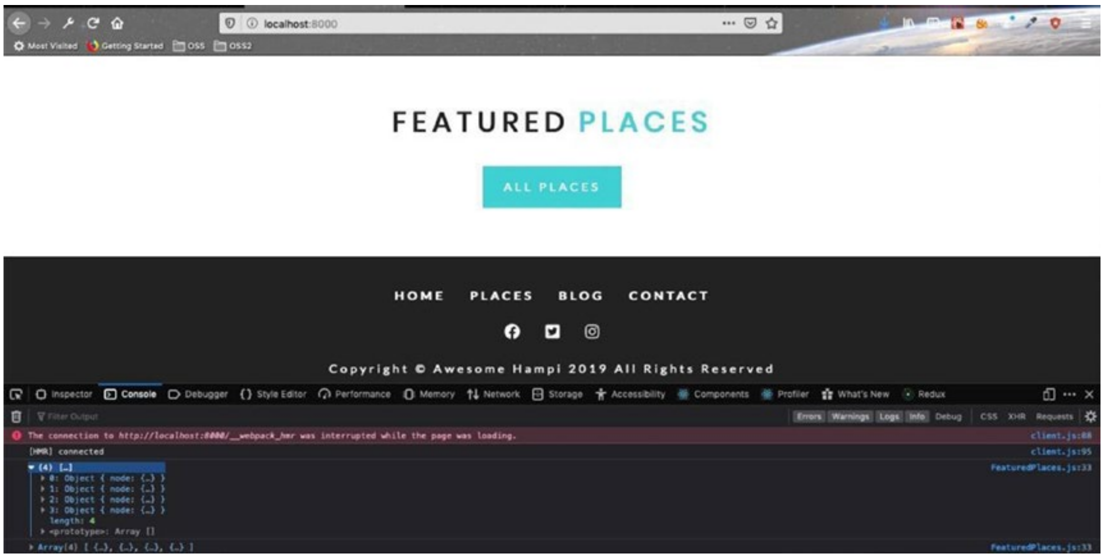
```
import React from 'react'
import { useStaticQuery, graphql } from "gatsby"
import Title from "../Title"
import styles from "../../css/items.module.css"
import AniLink from "gatsby-plugin-transition-link/AniLink"

const getFeaturedPlaces = graphql`
query{
featuredPlaces: allContentfulAmazingHampiData(filter:{featured:{eq:true}})
{
    edges {
        node {
            contentful_id
            name
            slug
            timeRequired
            timings
            entryFees
            featured
        images {
            fluid {
                ...GatsbyContentfulFluid
                }
            }
        }
        }
    }
}
`;

const FeaturedPlaces = () => {
    const response = useStaticQuery(getFeaturedPlaces)
    const places = response.featuredPlaces.edges
          console.log(places);
    return (
        <section className={styles.places}>
            <Title title="featured" subtitle="places" />
            <AniLink fade to="/places" className="btn-primary">
                all places
            </AniLink>
        </section>
    )
}

export default FeaturedPlaces
```

Next, head over to `index.js` to add the `FeaturedPlaces` component, as highlighted in Listing 4-9.

Listing 4-9. FeaturedPlaces in the index.js File

```
...
...
import { graphql } from 'gatsby'
import FeaturedPlaces from "../components/Home/FeaturedPlaces"

...
...

export default ({ data }) => (
    <Layout>
        <StyledHero home="true" img={data.defaultBcg.childImageSharp.fluid}>
                       ...
                       ...
        </StyledHero>
        <About />
        <Tips />
        <FeaturedPlaces />
    </Layout>
)
```

In the browser, open the home page and choose Developer Tools ➤ Console. We can see data coming from Contentful, as shown in Figure 4-42.



```
The connection to http://localhost:8000/__webpack_hmr was interrupted while the page was loading.   client.js:88
[HMR] connected                                                                                      client.js:95
▼ (4) […]                                                                                 FeaturedPlaces.js:33
  ▸ 0: Object { node: {…} }
  ▸ 1: Object { node: {…} }
  ▸ 2: Object { node: {…} }
  ▸ 3: Object { node: {…} }
    length: 4
  ▸ <prototype>: Array []
▸ Array(4) [ {…}, {…}, {…}, {…} ]                                                         FeaturedPlaces.js:33
```

Figure 4-42. *We can see data coming from Contentful using the console*

Let's show the `places` array from the `FeaturedPlaces` component by adding the highlighted part in the `FeaturedPlaces.js` file, as shown in Listing 4-10.

Listing 4-10. The Places Array in FeaturedPlaces.js

```
...
...
const FeaturedPlaces = () => {
    const response = useStaticQuery(getFeaturedPlaces)
    const places = response.featuredPlaces.edges

    return (
        <section className={styles.places}>
            <Title title="featured" subtitle="places" />
            <div className={styles.center}>
                {places.map(({ node }) => {
                    return <Place key={node.contentful_id} place={node} />
                })}
            </div>
            <AniLink fade to="/places" className="btn-primary">
                all places
            </AniLink>
        </section>
    )
}

export default FeaturedPlaces
```

We will create a simple `Place` component for now. Create a `Place.js` file inside the `Places` folder, using the content in Listing 4-11.

Listing 4-11. The Place.js File

```
import React from 'react'
const Place = () => {
    return (
        <div>
           Single place
        </div>
    )
}

export default Place
```

When we move to the browser, we can see four single places, as shown in Figure 4-43.

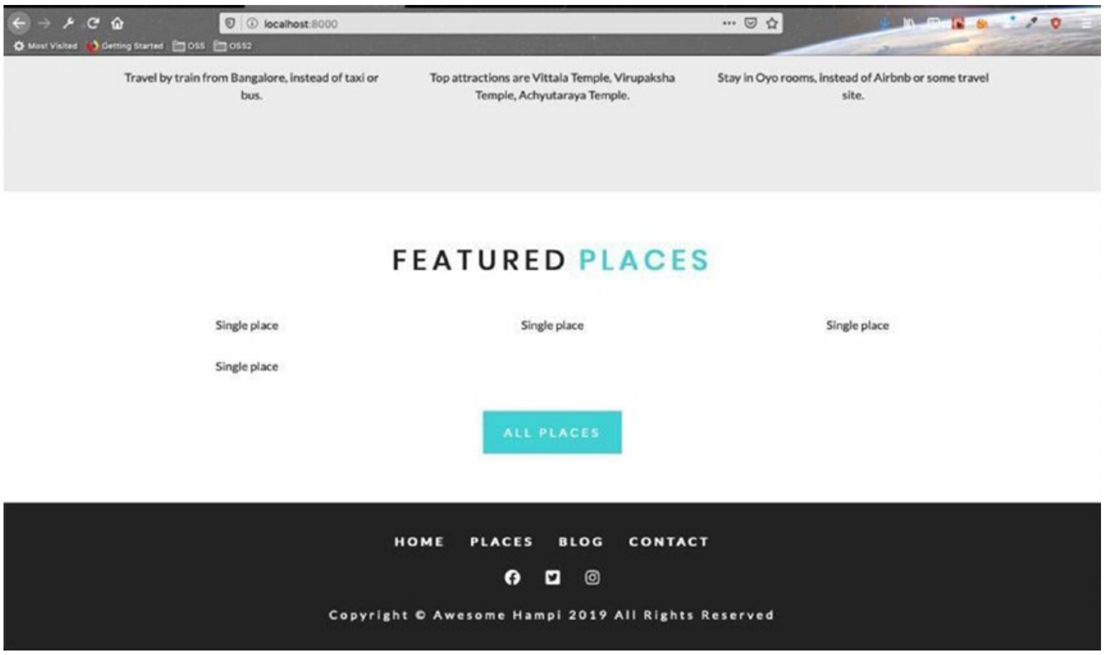

localhost:8000
Travel by train from Bangalore, instead of taxi or bus.
Top attractions are Vittala Temple, Virupaksha Temple, Achyutaraya Temple.
Stay in Oyo rooms, instead of Airbnb or some travel site.
FEATURED PLACES
Single place
Single place
Single place
Single place
ALL PLACES
HOME
PLACES
BLOG
CONTACT
Copyright © Awesome Hampi 2019 All Rights Reserved

Figure 4-43. *Four single places*

Adding the Place Component

Next, we will start to work on our Place component. First, let's add `place.module.css` inside the `css` folder. The content of it is shown in Listing 4-12.

Listing 4-12. The place.module.css File

```
.place {
  box-shadow: var(--lightShadow);
  transition: var(--mainTransition);
}
.place:hover {
  box-shadow: var(--darkShadow);
}

.img-container {
  position: relative;
  background: var(--primaryColor);
  transition: var(--mainTransition);
}
.img {
  transition: var(--mainTransition);
}
.img-container:hover .img {
  opacity: 0.3;
}

.link {
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  opacity: 0;
  text-transform: uppercase;
  letter-spacing: var(--mainSpacing);
  color: var(--mainWhite);
  border: 2px solid var(--mainWhite);
  padding: 0.9rem 1.6rem;
  display: inline-block;
  transition: var(--mainTransition);
  cursor: pointer;
}
.link:hover {
  background: var(--mainWhite);
  color: var(--primaryColor);
}
.img-container:hover .link {
  opacity: 1;
}

.footer {
  padding: 1rem;
  text-align: left;
}
```

```
.footer h3 {
  text-transform: capitalize;
  margin-bottom: 0;
}
.info {
  display: flex;
  flex-wrap: wrap;
  justify-content: space-between;
  text-transform: uppercase;
  align-items: center;
  margin-top: 0.5rem;
}
.info h6,
.info h4 {
  margin-bottom: 0;
}
.country {
  text-transform: capitalize;
  color: var(--primaryColor);
  display: flex;
  align-items: center;
}
.icon {
  margin-right: 0.4rem;
}
.details {
  color: var(--darkGrey);
  text-transform: uppercase;
  text-align: right;
}
```

Next, let's add some code to `Place.js` to show the images. The content is shown in Listing 4-13.

Listing 4-13. Code in Place.js

```
import React from 'react'
import Image from "gatsby-image"
import styles from "../../css/place.module.css"
import AniLink from "gatsby-plugin-transition-link/AniLink"

const Place = ({ place }) => {
    const { name, slug, images } = place;
    let mainImage = images[0].fluid;

    return (
        <article className={styles.place}>
            <div className={styles.imgContainer}>
                <Image fluid={mainImage} className={styles.img} alt="single
                place" />
            </div>
        </article>
    )
}

export default Place
```

This code will show all our three images on the Featured Places page, as shown in Figure 4-44.

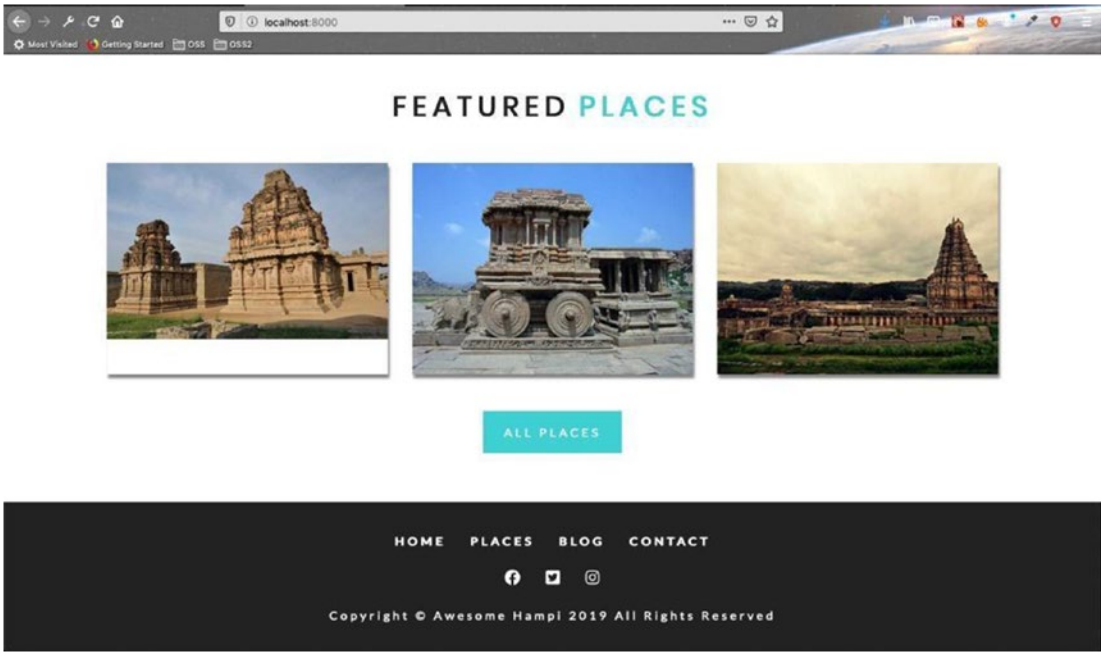


Figure 4-44. *Images of the featured places*

Let's complete the code by adding the `AniLink`, as highlighted in Listing 4-14, which will take visitors to the slug when they click it. We also show the name after the image.

Listing 4-14. AniLink in Place.js

```
import React from 'react'
import Image from "gatsby-image"
import styles from "../../css/place.module.css"
import AniLink from "gatsby-plugin-transition-link/AniLink"

const Place = ({ place }) => {
    const { name, slug, images } = place;
    let mainImage = images[0].fluid;

    return (
        <article className={styles.place}>
            <div className={styles.imgContainer}>
                <Image fluid={mainImage} className={styles.img} alt="single
                place" />
                <AniLink fade className={styles.link}
                to={`/places/${slug}`}>details</AniLink>
            </div>
            <div className={styles.footer}>
                <h3>{name}</h3>
            </div>
        </article>
    )
}

export default Place
```

This code will show the Featured Places with the names of the places below them, as shown in Figure 4-45.

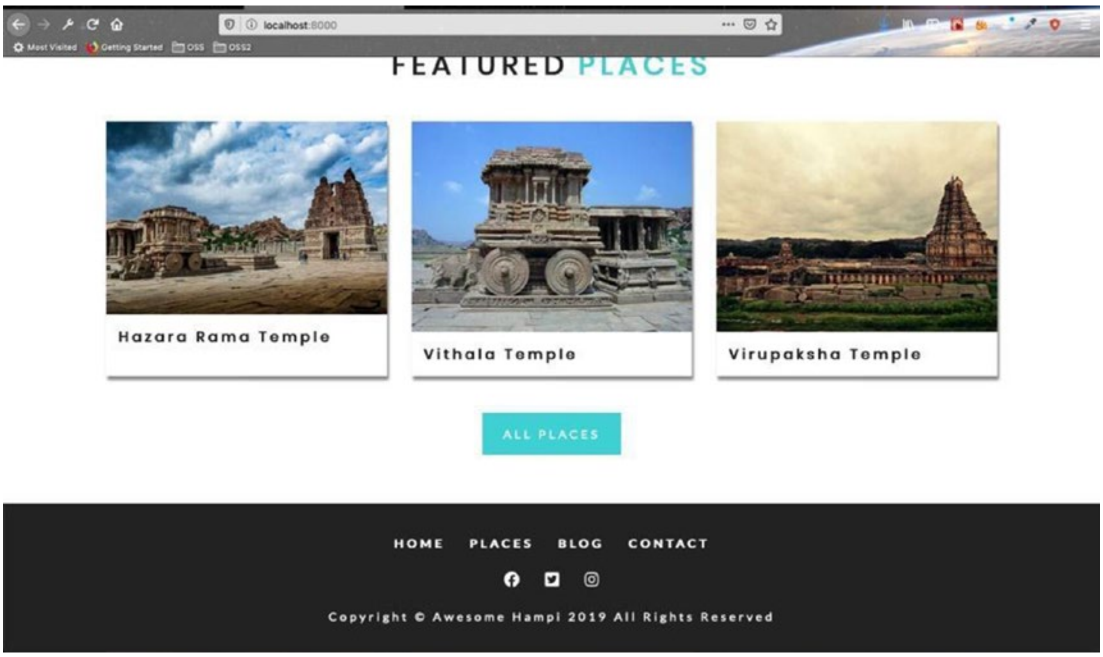


Figure 4-45. *Featured places with descriptive names*

Creating the Places Component

Let's start by creating our Places component. Open your `Places.js` file and update it as shown in Listing 4-15. We are basically using the GraphQL query, which we created in GraphiQL in the previous section.

Listing 4-15. The Places.js File

```
import React from "react"
import PlaceList from "./PlaceList"
import { useStaticQuery, graphql } from "gatsby"

const getPlaces = graphql`
query {
    places: allContentfulAmazingHampiData {
        edges {
            node {
                name
                timeRequired
                slug
                timings
                contentful_id
                entryFees
                images {
                    fluid {
                        ...GatsbyContentfulFluid
                }
            }
        }
    }
}
}
`
```

```
const Places = () => {
    const { places } = useStaticQuery(getPlaces)
    return <PlaceList places={places} />
}

export default Places
```

As we are getting the places, we are passing them to the `PlaceList` component. Let's create a simple `PlaceList` component now. Create a `PlaceList.js` file inside the `places` folder and add the content from Listing 4-16 to it.

Listing 4-16. The PlaceList.js File

```
import React, { Component } from 'react'

class PlaceList extends Component {
    render() {
        return (
                                 <div>
                                       PlaceList...
                               </div>
        )
    }
}
export default PlaceList
```

Next, we will show the `places` component in the `places.js` page. It is highlighted in Listing 4-17.

Listing 4-17. The Places component in places.js

```
import React from 'react'
import Layout from "../components/Layout"
import StyledHero from "../components/StyledHero"
import { graphql } from 'gatsby'
import Places from '../components/Places/Places'

...
...
```

```
export default function places({ data }) {
    return (
        <Layout>
            <StyledHero img={data.defaultBcg.childImageSharp.fluid}>
            </StyledHero>
            <Places />
        </Layout>
    )
}
```

Let's now update `PlaceList.js` to show all the places. Here we are using a class-based component and accessing the `places` passes from the Places component, by `this.props.places`.

We also have two local states of `places` and `sortedPlaces`. This is updated by `this.props.places` once the components loads and is updated by a React lifecycle called `componentDidMount()`.

After that, we sort over `sortedPlaces` and pass its value to an already created Place component. The contents are shown in Listing 4-18.

Listing 4-18. The Code in PlaceList.js

```
import React, { Component } from 'react'
import styles from "../../css/items.module.css"
import Place from "./Place"
import Title from "../Title"

class PlaceList extends Component {
    state = {
        places: [],
        sortedPlaces: [],
    }

    componentDidMount() {
        this.setState({
            places: this.props.places.edges,
            sortedPlaces: this.props.places.edges
        })
    }
    render() {
        return (
        <section className={styles.tours}>
            <Title title="hampi" subtitle="places" />
            <div className={styles.center}>
                {this.state.sortedPlaces && this.state.sortedPlaces.map(({
                node }) => {
                    return <Place key={node.contentful_id} place={node} />
                })}
            </div>
        </section>
        )
    }
}
export default PlaceList
```

When we move to the Places page[6], it will show all 22 places, as shown in Figure 4-46.

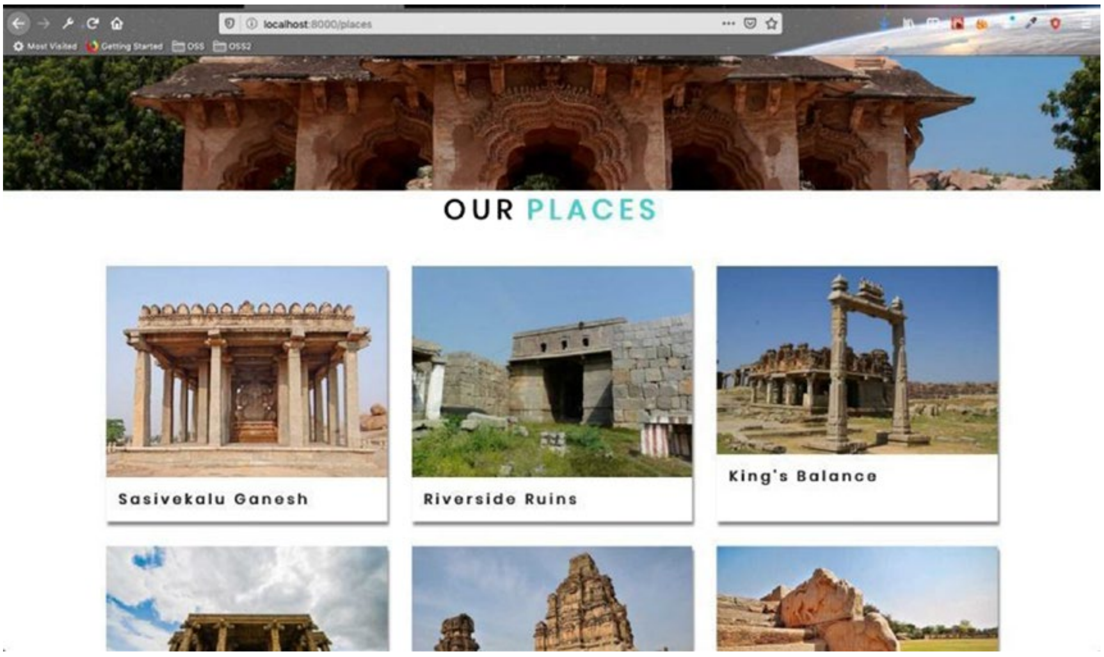


Figure 4-46. *All 22 places are shown on this page*

[6]`http://localhost:8000/places`

Create a Place Template

Next, we will create a template to show when we go to a single place. Head over to your code editor and create a `templates` folder inside the `src` folder. Create a file called **place-**`template.js` inside that folder. Add the basic code shown in Listing 4-19 to it.

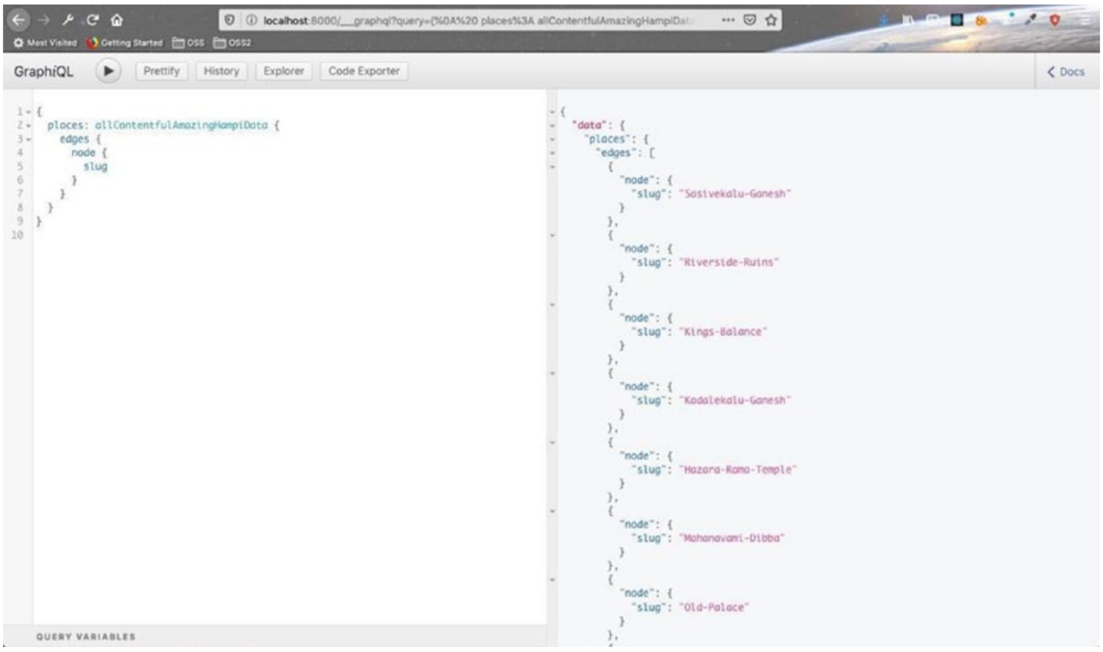
Listing 4-19. The place-template.js File

```
import React from 'react'

const placeTemplate = () =>  {
        return (
                    <div>
                         Dummy Place
                  </div>
        )
}

export default placeTemplate
```

We are basically creating our pages programmatically through the templates. Create a file called `gatsby-node.js` in the root directory.

Let's first create the query in GraphiQL. Head over to GraphQL[7] and create the query shown in Figure 4-47, which lists all the slugs.

[7]`http://localhost:8000/graphql`



```
{
  places: allContentfulAmazingHampiData {
    edges {
      node {
        slug
      }
    }
  }
}
```

```
{
  "data": {
    "places": {
      "edges": [
        {
          "node": {
            "slug": "Sasivekalu-Ganesh"
          }
        },
        {
          "node": {
            "slug": "Riverside-Ruins"
          }
        },
        {
          "node": {
            "slug": "Kings-Balance"
          }
        },
        {
          "node": {
            "slug": "Kadalekalu-Ganesh"
          }
        },
        {
          "node": {
            "slug": "Hazara-Rama-Temple"
          }
        },
        {
          "node": {
            "slug": "Mahanavami-Dibba"
          }
        },
        {
          "node": {
            "slug": "Old-Palace"
          }
        },
```

Figure 4-47. *The slugs query*

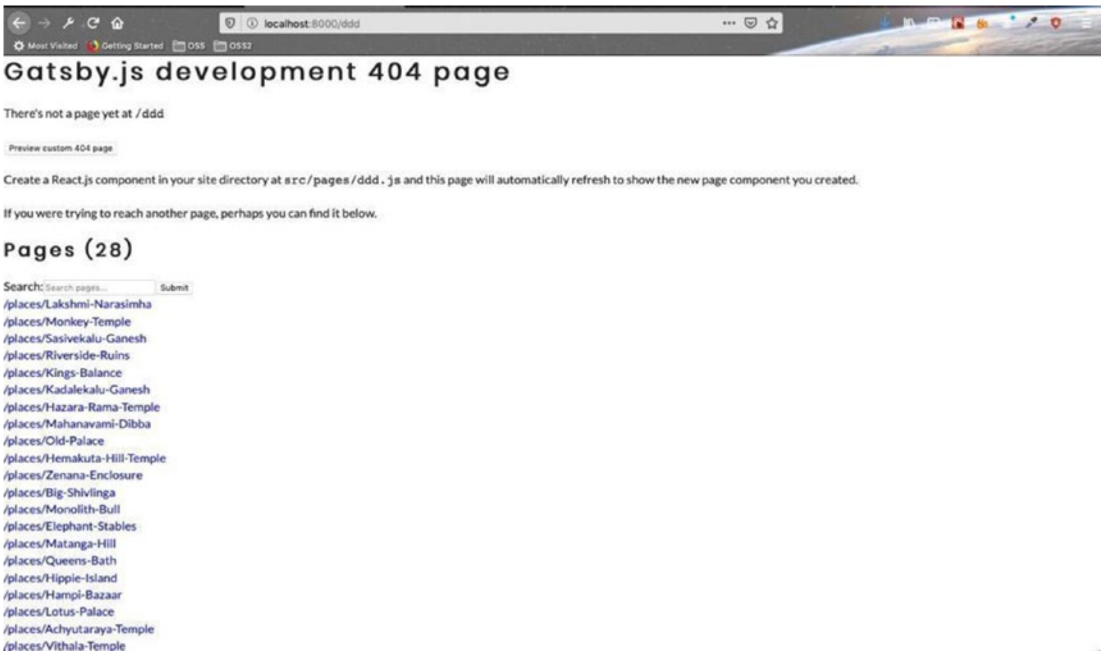
Head over to the `gatsby-node.js` file and add the content in Listing 4-20. Here, we are using `async-await` syntax to make the call. We are using the GraphQL query we created in the GraphiQL playground.

Once we receive the data, we loop over it and create the pages with the slug and template.

Listing 4-20. The gatsby-node.js File

```
const path = require("path")

exports.createPages = async ({ graphql, actions }) => {
const { createPage } = actions

const { data } = await graphql(`
query {
    places: allContentfulAmazingHampiData {
        edges {
        node {
                slug
            }
        }
    }
}
`)

data.places.edges.forEach(({ node }) => {
    createPage({
            path: `places/${node.slug}`,
            component: path.resolve("./src/templates/place-template.js"),
            context: {
                slug: node.slug,
            },
        })
    })
}
```

After saving the file, we need to restart the server. After restarting the browser, go to any nonexistent page and you will be shown the 404 page with all pages. We can see all the pages dynamically created, as shown in Figure 4-48.



Gatsby.js development 404 page

There's not a page yet at /ddd

Create a React.js component in your site directory at `src/pages/ddd.js` and this page will automatically refresh to show the new page component you created.

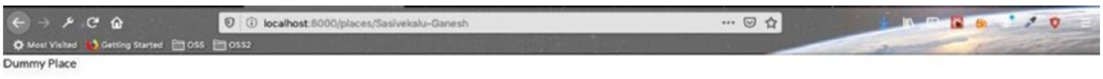
If you were trying to reach another page, perhaps you can find it below.

Pages (28)

/places/Lakshmi-Narasimha
/places/Monkey-Temple
/places/Sasivekalu-Ganesh
/places/Riverside-Ruins
/places/Kings-Balance
/places/Kadalekalu-Ganesh
/places/Hazara-Rama-Temple
/places/Mahanavami-Dibba
/places/Old-Palace
/places/Hemakuta-Hill-Temple
/places/Zenana-Enclosure
/places/Big-Shivlinga
/places/Monolith-Bull
/places/Elephant-Stables
/places/Matanga-Hill
/places/Queens-Bath
/places/Hippie-Island
/places/Hampi-Bazaar
/places/Lotus-Palace
/places/Achyutaraya-Temple
/places/Vithala-Temple

Figure 4-48. *All pages*

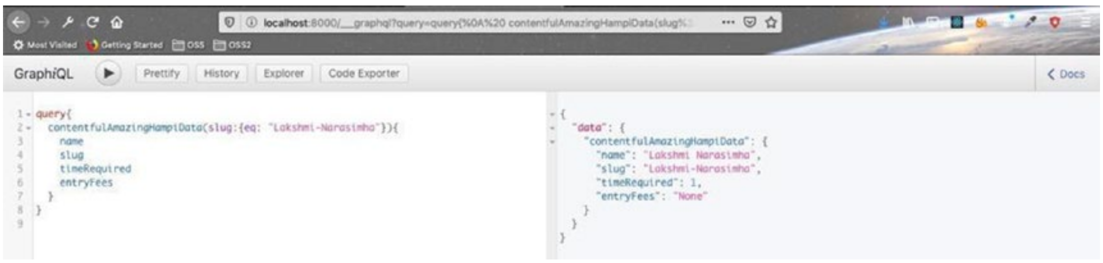
If we click any area of the page, it will show us data from `place-template.js`, as shown in Figure 4-49.



Dummy Place

Figure 4-49. Dummy place

We will create the `place-template.js` file next. In this part, we will start by creating the query for the template page. We will first create the query to get data of a single slug, as shown in Figure 4-50.

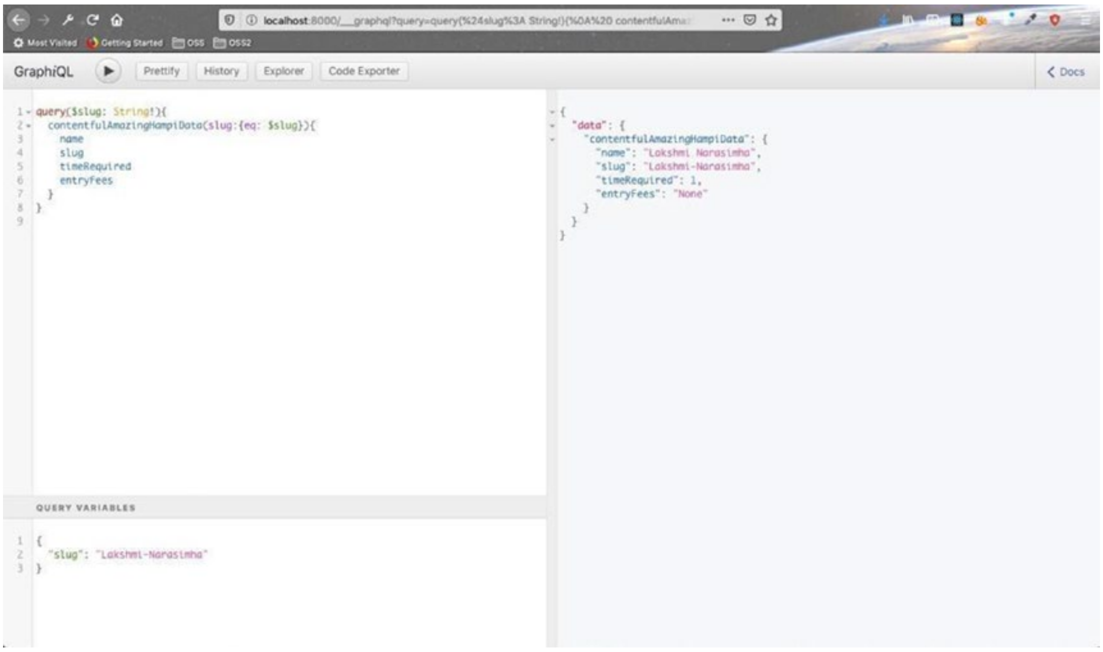


```
query{
  contentfulAmazingHampiData(slug:{eq: "Lakshmi-Narasimha"}){
    name
    slug
    timeRequired
    entryFees
  }
}
```

```
{
  "data": {
    "contentfulAmazingHampiData": {
      "name": "Lakshmi Narasimha",
      "slug": "Lakshmi-Narasimha",
      "timeRequired": 1,
      "entryFees": "None"
    }
  }
}
```

Figure 4-50. Single slug

We need to dynamically pass a variable to this query from our code. To test this, we will update the query and pass the variable from the Query Variables screen. This is shown in Figure 4-51.


localhost:8000/___graphql?query=query(%24slug%3A String!)(%0A%20 contentfulAma
GraphiQL
Prettify
History
Explorer
Code Exporter
Docs
QUERY VARIABLES

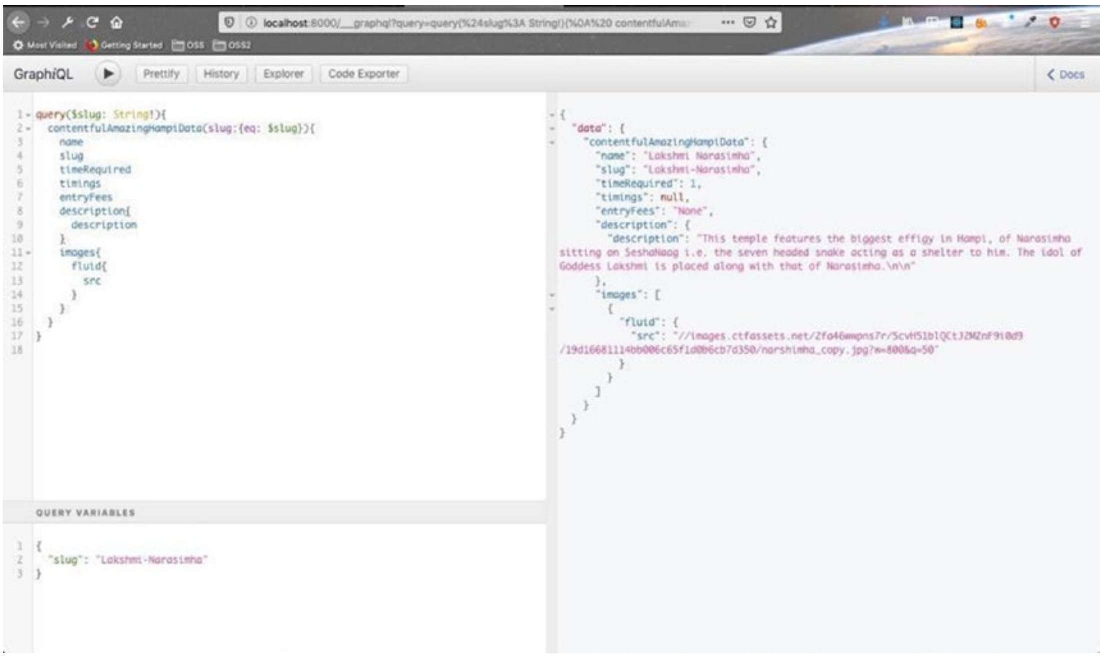
```
query($slug: String!){
  contentfulAmazingHampiData(slug:{eq: $slug}){
    name
    slug
    timeRequired
    entryFees
  }
}
```

```
{
  "data": {
    "contentfulAmazingHampiData": {
      "name": "Lakshmi Narasimha",
      "slug": "Lakshmi-Narasimha",
      "timeRequired": 1,
      "entryFees": "None"
    }
  }
}
```

```
{
  "slug": "Lakshmi-Narasimha"
}
```

Figure 4-51. *Passing variables*

Let's complete this query by adding all the fields, as shown in Figure 4-52.


localhost:8000/___graphql?query=query(%24slug%3A String!)(%0A%20 contentfulAma
GraphiQL
Prettify
History
Explorer
Code Exporter
Docs
QUERY VARIABLES

```
query($slug: String!){
  contentfulAmazingHampiData(slug:{eq: $slug}){
    name
    slug
    timeRequired
    timings
    entryFees
    description{
      description
    }
    images{
      fluid{
        src
      }
    }
  }
}
```

```
{
  "data": {
    "contentfulAmazingHampiData": {
      "name": "Lakshmi Narasimha",
      "slug": "Lakshmi-Narasimha",
      "timeRequired": 1,
      "timings": null,
      "entryFees": "None",
      "description": {
        "description": "This temple features the biggest effigy in Hampi, of Narasimha
sitting on SeshaNaag i.e. the seven headed snake acting as a shelter to him. The idol of
Goddess Lakshmi is placed along with that of Narasimha.\n\n"
      },
      "images": [
        {
          "fluid": {
            "src": "//images.ctfassets.net/2fa46wmpns7r/5cvH51b1QCtJ2MZnF9i0d9
/19d16681114bb006c65f1d0b6cb7d350/narshimha_copy.jpg?w=800&q=50"
          }
        }
      ]
    }
  }
}
```

```
{
  "slug": "Lakshmi-Narasimha"
}
```

Figure 4-52. *All fields added to the query*

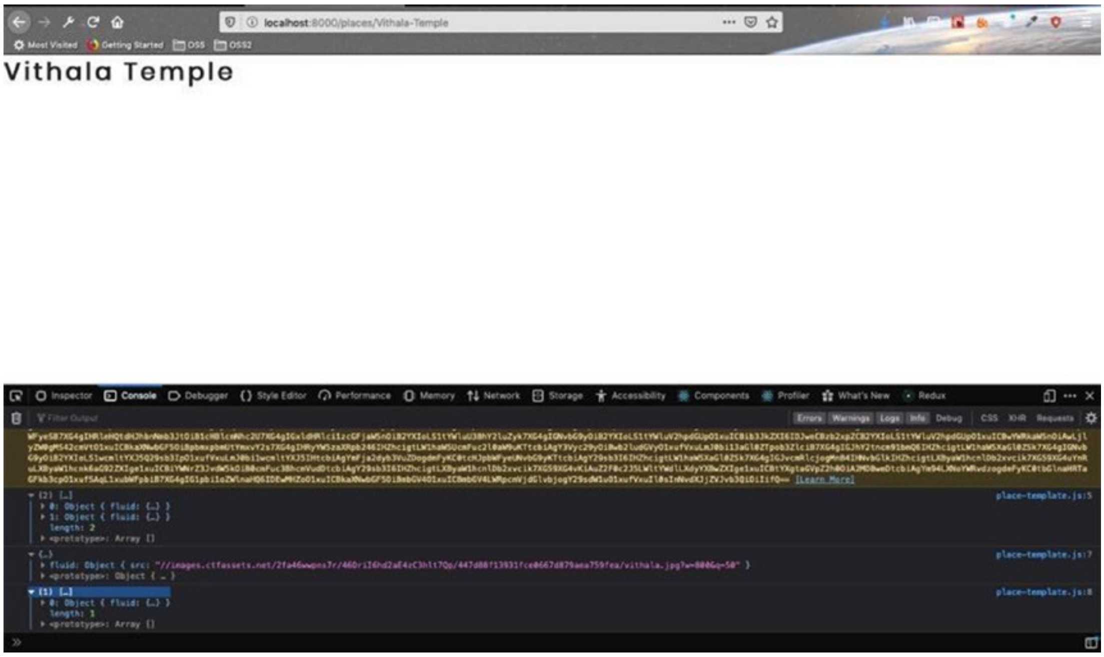
Next, we will add the query to `place-template.js` and display it. We are destructuring the data, which we receive from the query.

Some of our places have multiple images, so we are destructuring the images. We keep the first image as `mainImage` and the rest of the images in the array as `placeImages`. The code is shown in Listing 4-21.

Listing 4-21. The Code in place-template.js

```
import React from "react"
import { graphql } from "gatsby"

const Template = ({ data }) => {
    const { name, timeRequired, timings, entryFees, description: {
    description }, images } = data.place;
            console.log(images);
    const [mainImage, ...placeImages] = images
             console.log(mainImage);
             console.log(placeImages);
    return <h1>{name}</h1>
 }

export const query = graphql`
query($slug: String!){
    place: contentfulAmazingHampiData(slug:{eq: $slug}){
        name
        slug
        timeRequired
        timings
        entryFees
        description{
            description
        }
        images{
            fluid{
                src
            }
        }
    }
}
`

export default Template
```

Once we move to a place with multiple images and open the console, we can see the variables. The name is also displayed on the page, as shown in Figure 4-53.



```
[illegible] [Learn More]
▼ (2) […]                                                         place-template.js:5
  ▶ 0: Object { fluid: {…} }
  ▶ 1: Object { fluid: {…} }
    length: 2
  ▶ <prototype>: Array []
▼ {…}                                                             place-template.js:7
  ▶ fluid: Object { src: "//images.ctfassets.net/2fa46wwpns7r/460riI6hd2aE4zC3hlt7Qp/447d88f13931fce0667d879aea759fea/vithala.jpg?w=800&q=50" }
  ▶ <prototype>: Object { … }
▼ (1) […]                                                         place-template.js:8
  ▶ 0: Object { fluid: {…} }
    length: 1
  ▶ <prototype>: Array []
```

Figure 4-53. *The variables at work*

Let's add the styles for the templates by creating a file called `template.module.css` inside the `css` folder. The file's contents are shown in Listing 4-22.

Listing 4-22. The template.module.css File

```
.template {
  padding: 4rem 0;
}
.center {
  width: 80vw;
  margin: 0 auto;
}
```

```
.images {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
  grid-column-gap: 1rem;
  grid-row-gap: 1rem;
  margin-bottom: 2rem;
}
.image {
  box-shadow: var(--lightShadow);
}
.template h2 {
  text-transform: capitalize;
  letter-spacing: var(--mainSpacing);
  margin-bottom: 1rem;
}
.info {
  display: flex;
  flex-wrap: wrap;
}
.info p {
  display: flex;
  align-items: center;
  margin-right: 2rem;
  text-transform: capitalize;
}
.icon {
  color: var(--primaryColor);
  font-size: 1.4rem;
  margin-right: 0.5rem;
}
.desc {
  line-height: 2;
}
.template h4 {
  text-transform: capitalize;
}
```

```
.template h2 {
  margin: 2rem 0;
}
.journey {
  margin: 3rem 0;
}
@media screen and (min-width: 992px) {
  .journey,
  .desc {
    width: 70vw;
  }
}
@media screen and (min-width: 1200px) {
  .center {
    width: 95vw;
    max-width: 1170vw;
  }
  .images {
    grid-template-columns: repeat(auto-fit, minmax(340px, 1fr));
    grid-column-gap: 50px;
  }
}
```

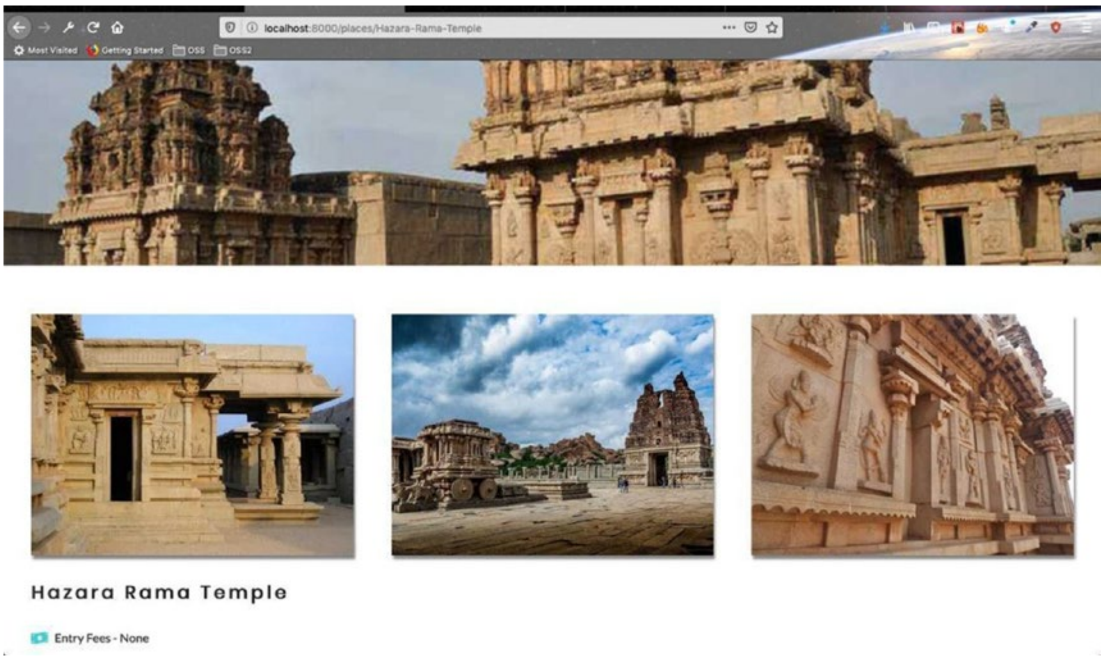
Next, we need to update `place-template.js` to show more fields. Also, we need to change our `src` in the query to a fragment or the code will give an error. The updated code is marked in bold in Listing 4-23.

Listing 4-23. More Fields in place-template.js

```
import React from "react"
import { graphql } from "gatsby"
import Layout from "../components/Layout"
import StyledHero from "../components/StyledHero"
import styles from "../css/template.module.css"
import Img from "gatsby-image"
import { FaMoneyBillWave } from "react-icons/fa"
```

```
const Template = ({ data }) => {
    const { name, timeRequired, timings, entryFees, description: {
    description }, images } = data.place;
    const [mainImage, ...placeImages] = images

    return (
        <Layout>

            <StyledHero img={mainImage.fluid} />
            <section className={styles.template}>
                <div className={styles.center}>
                    <div className={styles.images}>
                        {placeImages && placeImages.map((item,index) =>{
                            return <Img key={index} fluid={item.fluid}
                            alt="single" className={styles.image}/>
                        })}
                    </div>
                    <h2>{name}</h2>
                    <div className={styles.info}>
                        <p>
                            <FaMoneyBillWave className={styles.icon} />
                            Entry Fees - {entryFees}
                        </p>
                    </div>
                </div>
            </section>
        </Layout>
    )
}

export const query = graphql`
query($slug: String!){
    place: contentfulAmazingHampiData(slug:{eq: $slug}){
        name
        slug
        timeRequired
        timings
        entryFees
        description{
            description
        }
        images{
            fluid{
                ...GatsbyContentfulFluid_tracedSVG
            }
        }
    }
}
`
export default Template
```

Once we go to a page, we can see our images and the text, as shown in Figure 4-54.



Hazara Rama Temple

Entry Fees - None

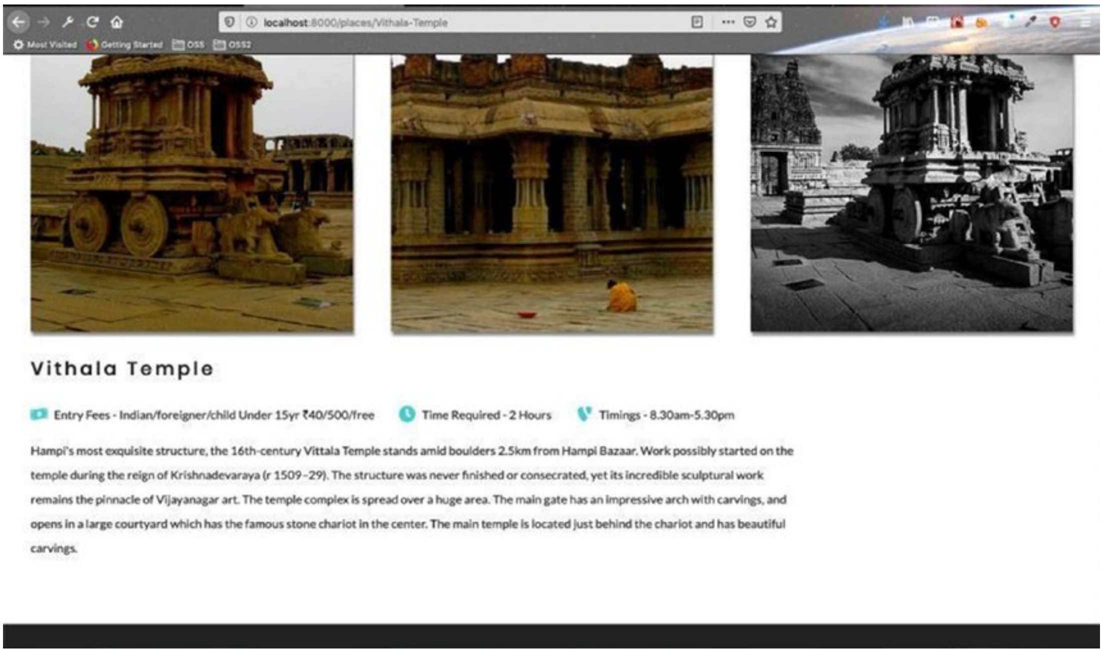
Figure 4-54. *Images and text are now displayed*

Next, let's add all the fields to the `place-template.js` for our places. It will contain the `timeRequired`, `timings`, and `description` fields. The updated code is marked in bold in Listing 4-24.

Listing 4-24. Time Fields in place-template.js

```
...
...
import { FaMoneyBillWave, FaClock, FaTypo3 } from "react-icons/fa"

const Template = ({ data }) => {
    const { name, timeRequired, timings, entryFees, description: {
description }, images } = data.place;
    const [mainImage, ...placeImages] = images

    return (
        <Layout>
            <SEO title={name} />
            <StyledHero img={mainImage.fluid} />
            <section className={styles.template}>
                <div className={styles.center}>
                    <div className={styles.images}>
                        {placeImages && placeImages.map((item,index) =>{
                            return <Img key={index} fluid={item.fluid}
                            alt="single" className={styles.image}/>
                        })}
                    </div>
                    <h2>{name}</h2>
                    <div className={styles.info}>
                        <p>
                            <FaMoneyBillWave className={styles.icon} />
                            Entry Fees - {entryFees}
                        </p>
                        <p>
                            <FaClock className={styles.icon} />Time
                            Required - {timeRequired} hours
                        </p>
                        {timings ?
                            <p>
                                <FaTypo3 className={styles.icon} />
                                Timings - {timings}
                            </p>:

                        }
                    </div>
             <p className={styles.desc}>{description}</p>
                </div>
            </section>
        </Layout>
    )
}
...
```

Our page now looks almost complete, as shown in Figure 4-55.



localhost:8000/places/Vithala-Temple

Vithala Temple

Entry Fees - Indian/foreigner/child Under 15yr ₹40/500/free Time Required - 2 Hours Timings - 8.30am-5.30pm

Hampi's most exquisite structure, the 16th-century Vittala Temple stands amid boulders 2.5km from Hampi Bazaar. Work possibly started on the temple during the reign of Krishnadevaraya (r 1509–29). The structure was never finished or consecrated, yet its incredible sculptural work remains the pinnacle of Vijayanagar art. The temple complex is spread over a huge area. The main gate has an impressive arch with carvings, and opens in a large courtyard which has the famous stone chariot in the center. The main temple is located just behind the chariot and has beautiful carvings.

Figure 4-55. *The page is nearly complete*

One last thing to do is to add a button that allows visitors to go back to the Places page. Add an `AniLink` to `place-template.js`. The updated code is marked in bold in Listing 4-25.

Listing 4-25. AniLink in place-template.js

```
...
...
import { FaMoneyBillWave, FaClock, FaTypo3 } from "react-icons/fa"
import AniLink from "gatsby-plugin-transition-link/AniLink"

const Template = ({ data }) => {
    const { name, timeRequired, timings, entryFees, description: {
    description }, images } = data.place;
    const [mainImage, ...placeImages] = images

    return (
        <Layout>
            <SEO title={name} />
            <StyledHero img={mainImage.fluid} />
            <section className={styles.template}>
                <div className={styles.center}>
                                              ...
                                              ...
                    <h2>{name}</h2>
                    <div className={styles.info}>
                                         ...
                                         ...
                    </div>
              <p className={styles.desc}>{description}</p>
                    <AniLink fade to="/places" className="btn-primary">back
                    to places</AniLink>
                </div>
            </section>
        </Layout>
    )
}

...
```

This will display a nice return button (called Back to Tours), as shown in Figure 4-56.

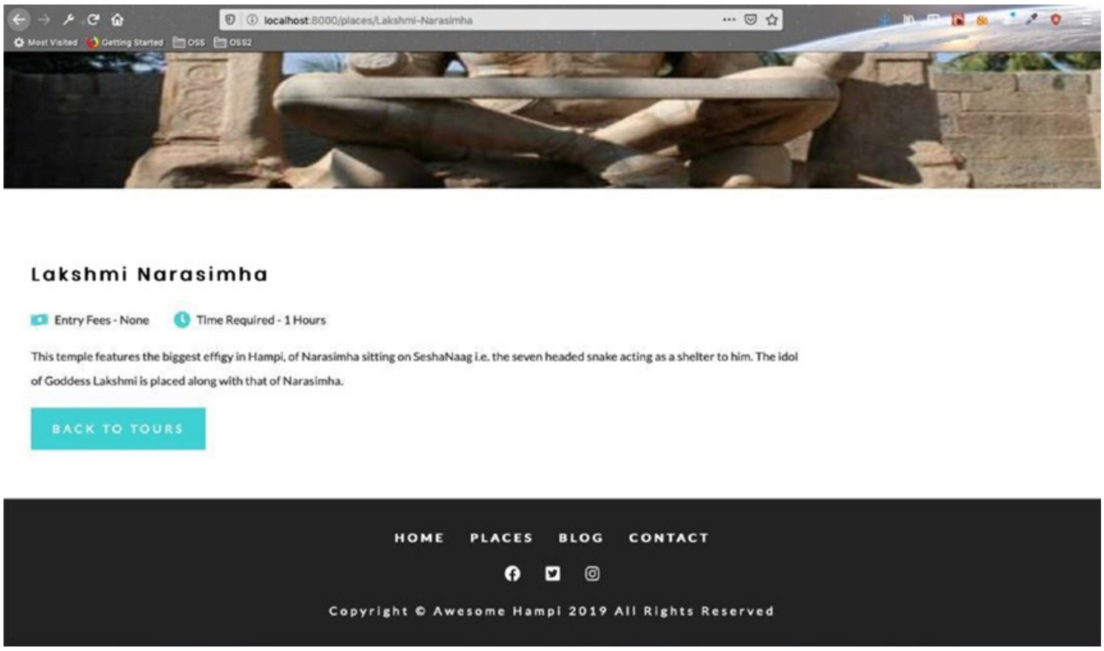
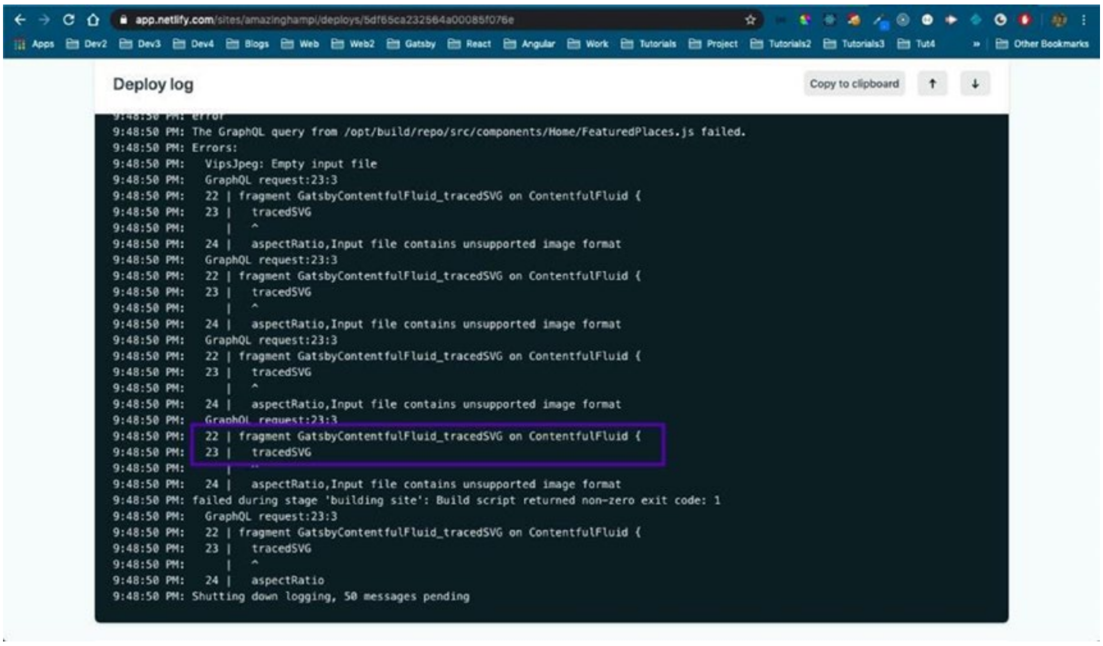

localhost:8000/places/Lakshmi-Narasimha
Lakshmi Narasimha
Entry Fees - None
Time Required - 1 Hours
This temple features the biggest effigy in Hampi, of Narasimha sitting on SeshaNaag i.e. the seven headed snake acting as a shelter to him. The idol of Goddess Lakshmi is placed along with that of Narasimha.
BACK TO TOURS
HOME
PLACES
BLOG
CONTACT
Copyright © Awesome Hampi 2019 All Rights Reserved

Figure 4-56. *The Back to Tours button is displayed*

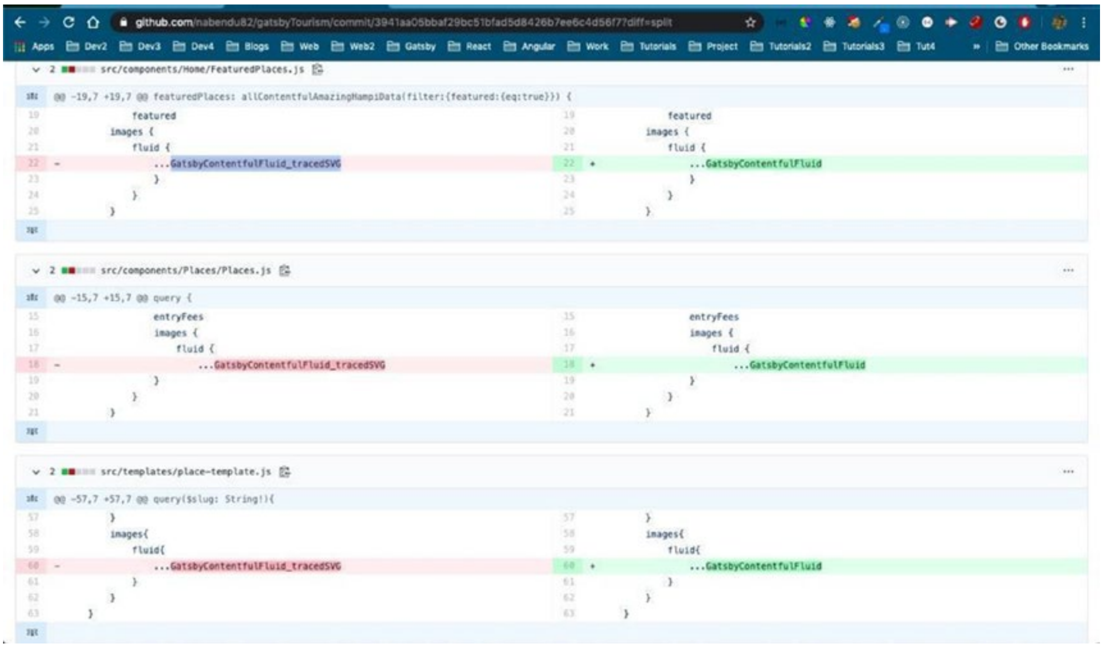
Once it has pushed the code to GitHub, it starts the automatic deployment to Netlify. My Netlify build failed because of the `GatsbyContentfulFluid_tracedSVG` fragment that's used in three places (see Figure 4-57).



```
9:48:50 PM: The GraphQL query from /opt/build/repo/src/components/Home/FeaturedPlaces.js failed.
9:48:50 PM: Errors:
9:48:50 PM:   VipsJpeg: Empty input file
9:48:50 PM:   GraphQL request:23:3
9:48:50 PM:   22 | fragment GatsbyContentfulFluid_tracedSVG on ContentfulFluid {
9:48:50 PM:   23 |   tracedSVG
9:48:50 PM:      |   ^
9:48:50 PM:   24 |   aspectRatio,Input file contains unsupported image format
9:48:50 PM:   GraphQL request:23:3
9:48:50 PM:   22 | fragment GatsbyContentfulFluid_tracedSVG on ContentfulFluid {
9:48:50 PM:   23 |   tracedSVG
9:48:50 PM:      |   ^
9:48:50 PM:   24 |   aspectRatio,Input file contains unsupported image format
9:48:50 PM:   GraphQL request:23:3
9:48:50 PM:   22 | fragment GatsbyContentfulFluid_tracedSVG on ContentfulFluid {
9:48:50 PM:   23 |   tracedSVG
9:48:50 PM:      |   ^
9:48:50 PM:   24 |   aspectRatio,Input file contains unsupported image format
9:48:50 PM:   GraphQL request:23:3
9:48:50 PM:   22 | fragment GatsbyContentfulFluid_tracedSVG on ContentfulFluid {
9:48:50 PM:   23 |   tracedSVG
9:48:50 PM:      |   ^
9:48:50 PM:   24 |   aspectRatio,Input file contains unsupported image format
9:48:50 PM: failed during stage 'building site': Build script returned non-zero exit code: 1
9:48:50 PM:   GraphQL request:23:3
9:48:50 PM:   22 | fragment GatsbyContentfulFluid_tracedSVG on ContentfulFluid {
9:48:50 PM:   23 |   tracedSVG
9:48:50 PM:      |   ^
9:48:50 PM:   24 |   aspectRatio
9:48:50 PM: Shutting down logging, 50 messages pending
```

Figure 4-57. *The Netlify error*

I fixed the error by changing the fragment from `GatsbyContentfulFluid_tracedSVG` to `GatsbyContentfulFluid` in all three files, as shown in Figure 4-58.



```
github.com/nabendu82/gatsbyTourism/commit/3941aa05bbaf29bc51bfad5d8426b7ee6c4d56f77diff=split
```

```
2  src/components/Home/FeaturedPlaces.js
@@ -19,7 +19,7 @@ featuredPlaces: allContentfulAmazingHampiData(filter:{featured:{eq:true}}) {
            featured                                    featured
          images {                                    images {
            fluid {                                     fluid {
-             ...GatsbyContentfulFluid_tracedSVG    +     ...GatsbyContentfulFluid
            }                                           }
          }                                           }
        }                                           }
```

```
2  src/components/Places/Places.js
@@ -15,7 +15,7 @@ query {
              entryFees                                   entryFees
              images {                                    images {
                fluid {                                     fluid {
-                 ...GatsbyContentfulFluid_tracedSVG    +     ...GatsbyContentfulFluid
              }                                           }
            }                                           }
          }                                           }
```

```
2  src/templates/place-template.js
@@ -57,7 +57,7 @@ query($slug: String!){
          }                                           }
          images{                                     images{
            fluid{                                      fluid{
-             ...GatsbyContentfulFluid_tracedSVG    +     ...GatsbyContentfulFluid
            }                                           }
          }                                           }
        }                                           }
```

Figure 4-58. *Changing this fragment fixes the error*

You can find my code in my GitHub account here[8]. The site is successfully live here[9].

Summary

This completes Chapter 4 and the second part of the tourism site using Contentful. We covered the following topics in this chapter:

- Setting up the Contentful CMS and connecting to the project
- Creating the Places component, which shows data stored in the Contentful CMS

In the next chapter, we continue with the tourism site using Contentful. We will create the Blog and Photos components in that chapter.

[8]https://github.com/nabendu82/gatsbyTourism
[9]https://amazinghampi.netlify.com/

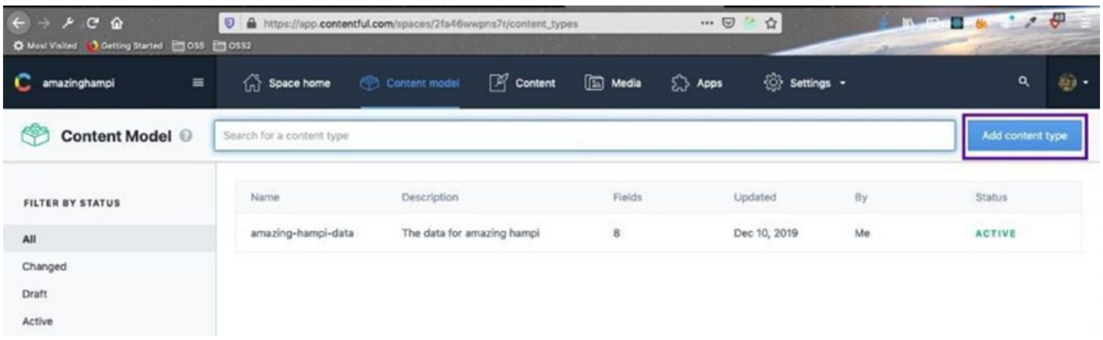
CHAPTER 5

Creating a Tourism Site with Contentful: Part Three

Welcome to Chapter 5. In this chapter, we will add two more components to the tourism site: the Blog and Photos components. They will show data stored in the Contentful CMS. Let's start by creating a new content model for the blog on the site.

Creating the Blog Component

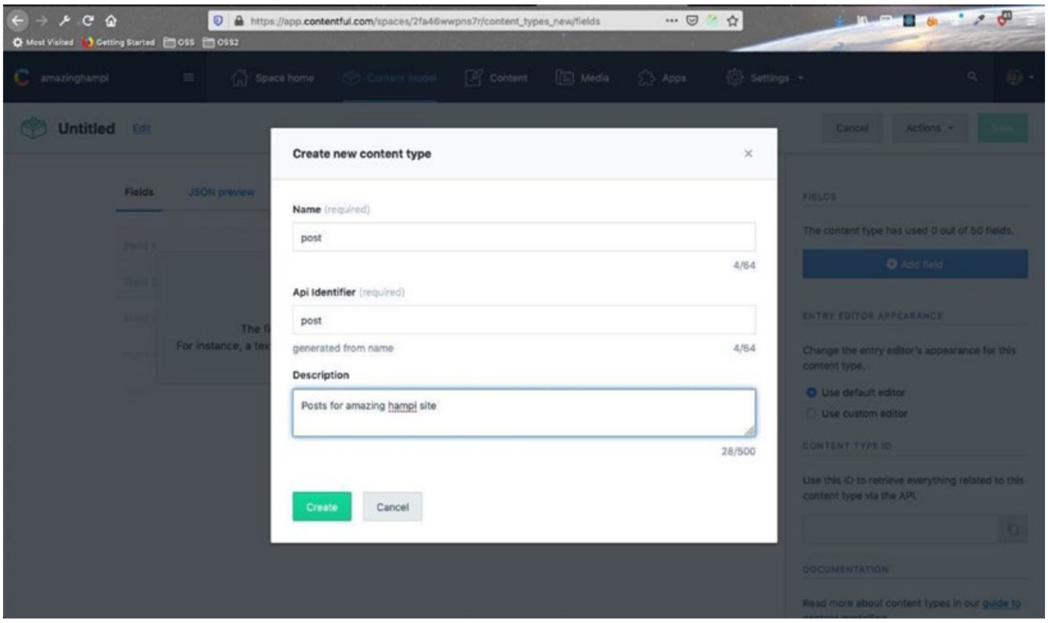
From the Contentful dashboard, click the Content Model tab and then click the Add Content Type button. This is shown in Figure 5-1.



Name	Description	Fields	Updated	By	Status
amazing-hampi-data	The data for amazing hampi	8	Dec 10, 2019	Me	ACTIVE

Figure 5-1. *Choose to add content type from the Contentful dashboard*

In the popup window shown in Figure 5-2, provide a name and description and then click Create.



Create new content type

Name (required)

post

4/64

Api Identifier (required)

post

generated from name 4/64

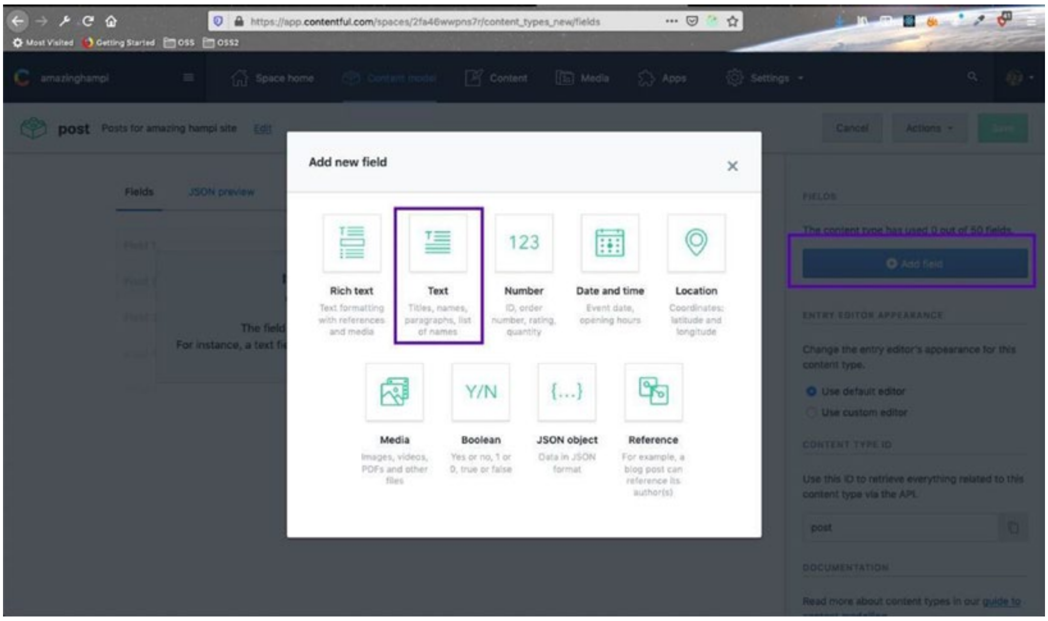
Description

Posts for amazing hampi site

28/500

Figure 5-2. Create new content type

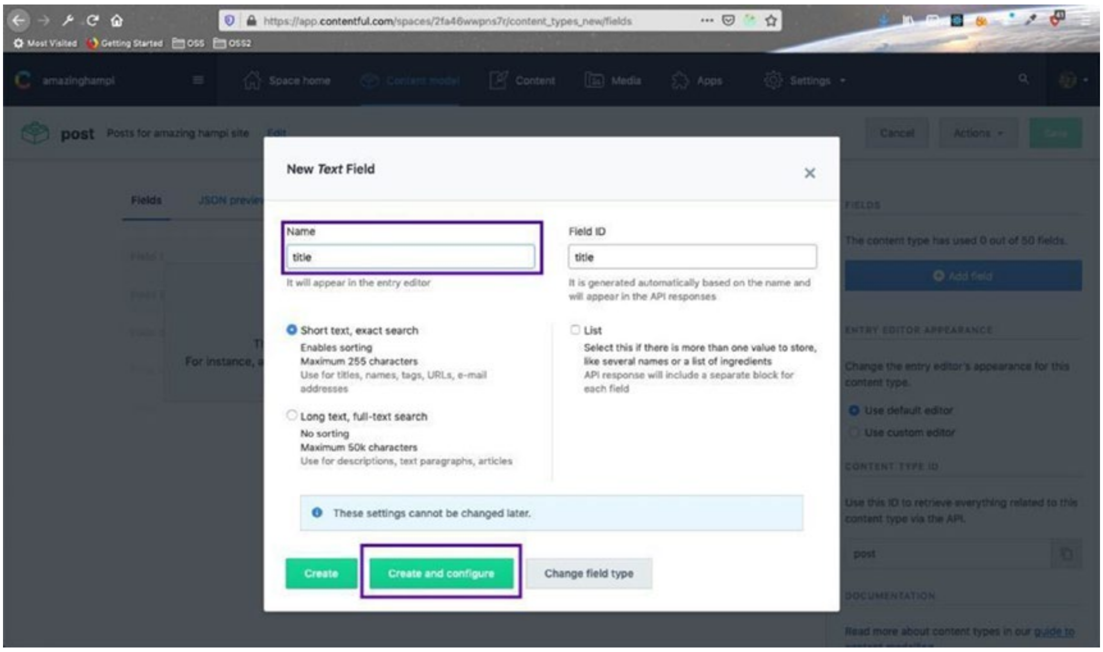
After you click Add Field, you will get the popup shown in Figure 5-3. Click Text in this window.



post
Posts for amazing hampi site
Add new field
Rich text
Text formatting with references and media
Text
Titles, names, paragraphs, list of names
Number
ID, order number, rating, quantity
Date and time
Event date, opening hours
Location
Coordinates: latitude and longitude
Media
Images, videos, PDFs and other files
Boolean
Yes or no, 1 or 0, true or false
JSON object
Data in JSON format
Reference
For example, a blog post can reference its author(s)
Add field

Figure 5-3. Add a Text field

Provide a name and click Create and Configure, as shown in Figure 5-4.


New *Text* Field

Name: title

It will appear in the entry editor

Field ID: title

It is generated automatically based on the name and will appear in the API responses

● Short text, exact search
Enables sorting
Maximum 255 characters
Use for titles, names, tags, URLs, e-mail addresses

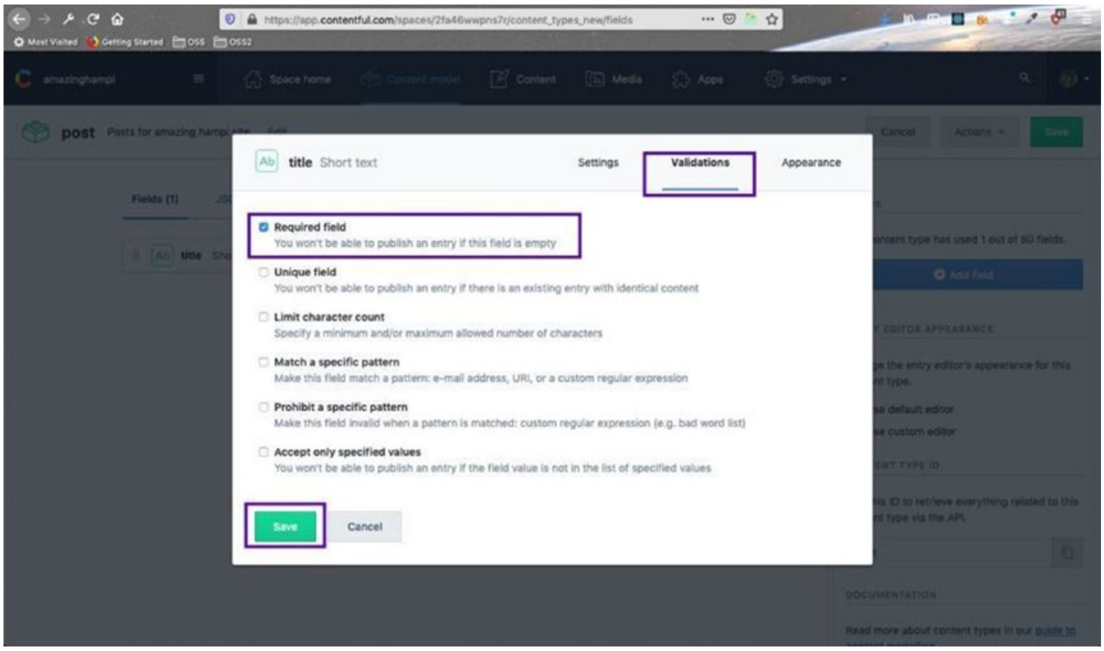
○ Long text, full-text search
No sorting
Maximum 50k characters
Use for descriptions, text paragraphs, articles

☐ List
Select this if there is more than one value to store, like several names or a list of ingredients
API response will include a separate block for each field

These settings cannot be changed later.

Figure 5-4. *Choose a name for your Text field*

In the next window, click the Validations tab. After that click, check the Required Field button and then click Save, as shown in Figure 5-5.



Required field
You won't be able to publish an entry if this field is empty

Unique field
You won't be able to publish an entry if there is an existing entry with identical content

Limit character count
Specify a minimum and/or maximum allowed number of characters

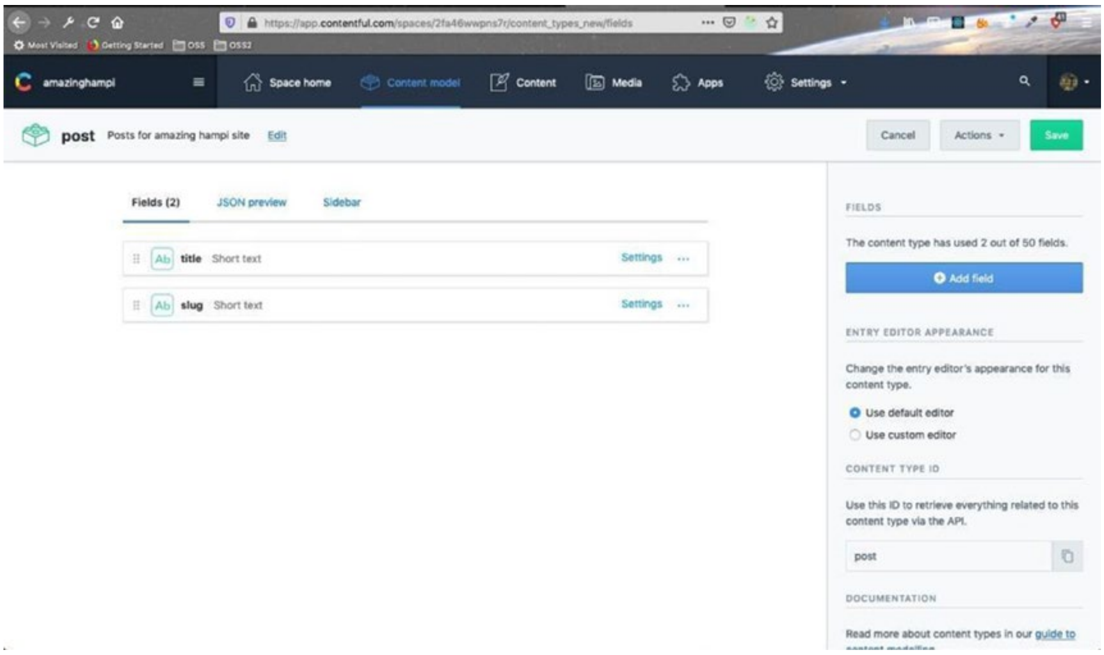
Match a specific pattern
Make this field match a pattern: e-mail address, URI, or a custom regular expression

Prohibit a specific pattern
Make this field invalid when a pattern is matched: custom regular expression (e.g. bad word list)

Accept only specified values
You won't be able to publish an entry if the field value is not in the list of specified values

Figure 5-5. *Make this field required*

Follow this exact same process to create a Slug field. It is shown in Figure 5-6.



The content type has used 2 out of 50 fields.

Change the entry editor's appearance for this content type.

Use this ID to retrieve everything related to this content type via the API.

Read more about content types in our guide to content modeling

Figure 5-6. *Create a slug field*

Next, add a Date and Time field, as shown in Figure 5-7.

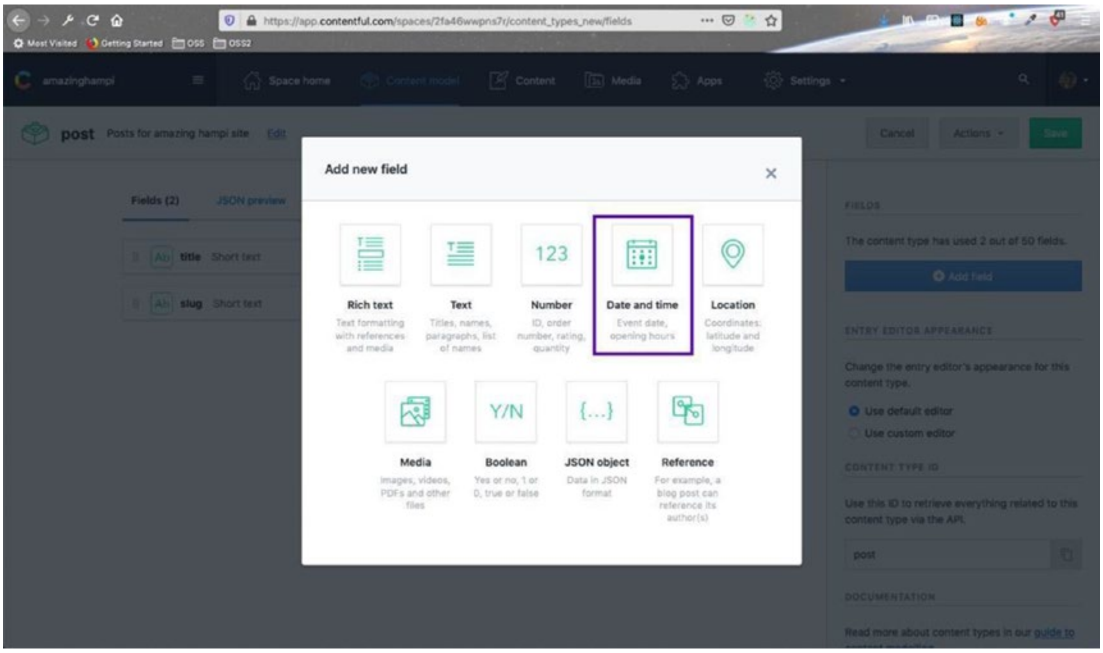
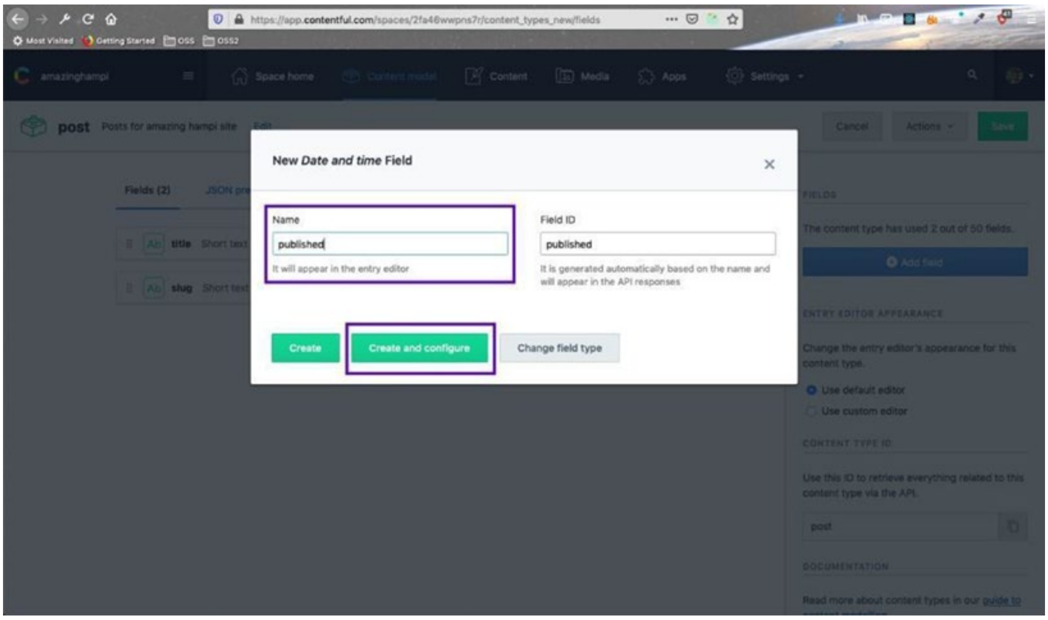

Add new field
Rich text
Text formatting with references and media
Text
Titles, names, paragraphs, list of names
Number
ID, order number, rating, quantity
Date and time
Event date, opening hours
Location
Coordinates: latitude and longitude
Media
Images, videos, PDFs and other files
Boolean
Yes or no, 1 or 0, true or false
JSON object
Data in JSON format
Reference
For example, a blog post can reference its author(s)

Figure 5-7. *Create a Date and Time field*

Give this field a name and click Create and Configure. You need to follow the process to make it a required field. This is shown in Figure 5-8.

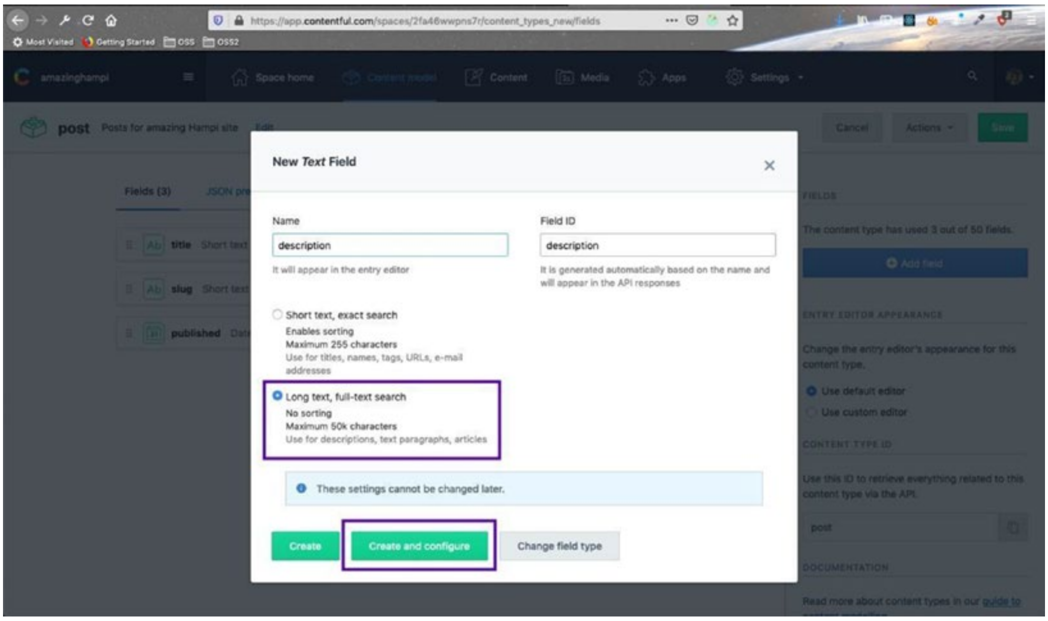

New *Date and time* Field

Name: published
It will appear in the entry editor

Field ID: published
It is generated automatically based on the name and will appear in the API responses

Figure 5-8. *The Published field*

Next, we will create a Description field. It will be Text field, but a Long text, full-text search. As usual, make it a required field, as shown in Figure 5-9.


New *Text* Field

Name: description
It will appear in the entry editor

Field ID: description
It is generated automatically based on the name and will appear in the API responses

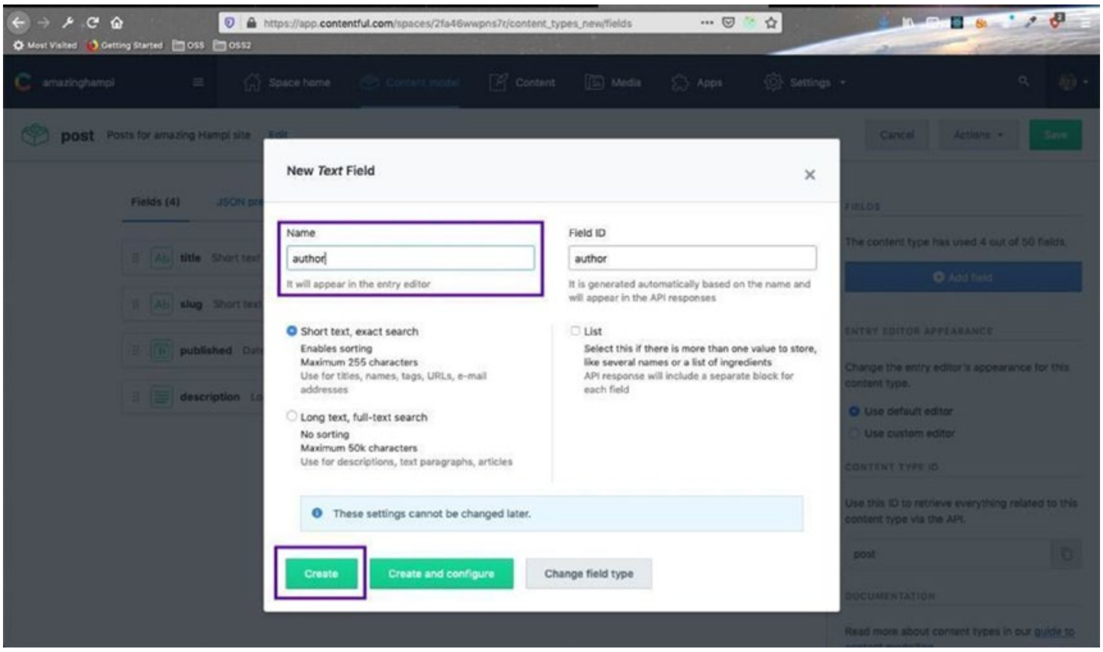
Short text, exact search
Enables sorting
Maximum 255 characters
Use for titles, names, tags, URLs, e-mail addresses

Long text, full-text search
No sorting
Maximum 50k characters
Use for descriptions, text paragraphs, articles

These settings cannot be changed later.

Figure 5-9. *The Description field*

Next, create an Author field. But don't make it required; simply click Create. This is shown in Figure 5-10.


New *Text* Field

Name: author

It will appear in the entry editor

Field ID: author

It is generated automatically based on the name and will appear in the API responses

[✓ Short text, exact search]
Enables sorting
Maximum 255 characters
Use for titles, names, tags, URLs, e-mail addresses

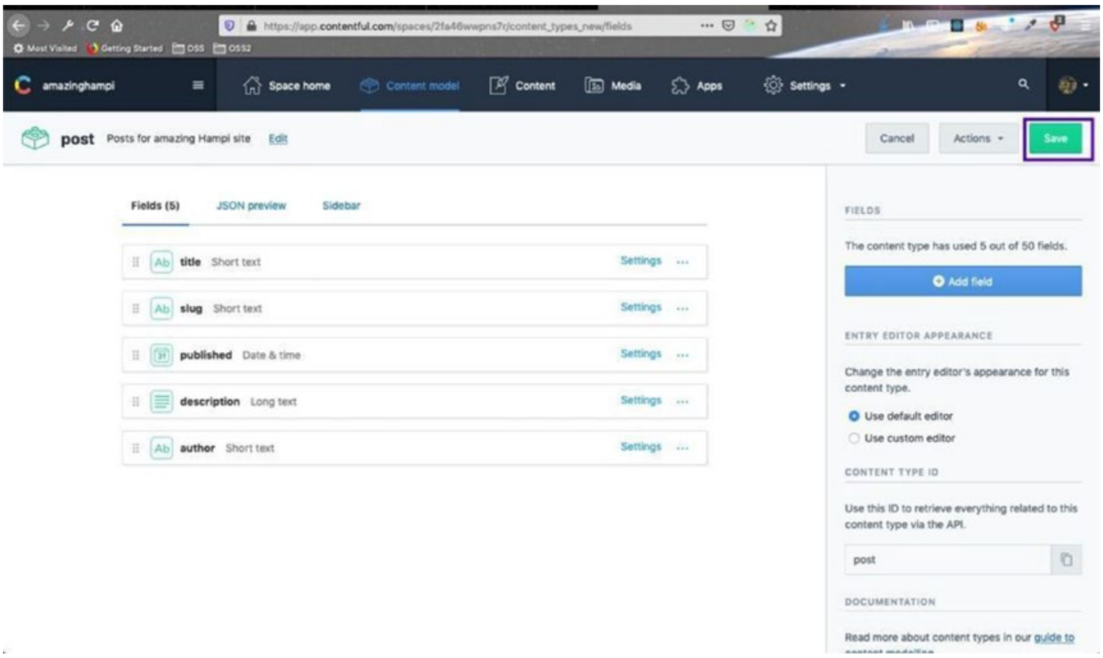
Long text, full-text search
No sorting
Maximum 50k characters
Use for descriptions, text paragraphs, articles

List
Select this if there is more than one value to store, like several names or a list of ingredients
API response will include a separate block for each field

These settings cannot be changed later.

Figure 5-10. *The Author field*

After adding all these fields, click Save in the top-right corner to save this content model, as shown in Figure 5-11.



post Posts for amazing Hampi site Edit

title Short text Settings
slug Short text Settings
published Date & time Settings
description Long text Settings
author Short text Settings

FIELDS

The content type has used 5 out of 50 fields.

ENTRY EDITOR APPEARANCE

Change the entry editor's appearance for this content type.

[✓ Use default editor]
Use custom editor

CONTENT TYPE ID

Use this ID to retrieve everything related to this content type via the API.

post

DOCUMENTATION

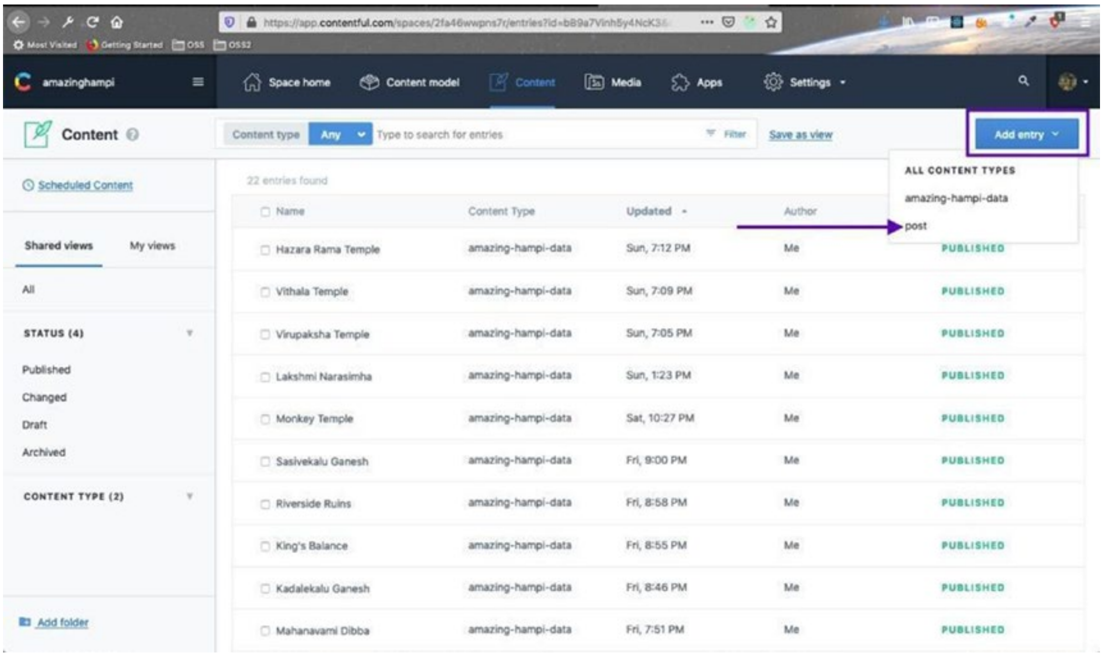
Read more about content types in our guide to

Figure 5-11. *Saving the new content model*

Next, let's add some data to the content model.

Adding Data to the Content Model

Go to the Content tab and click the Add Entry button. Then choose Post, as shown in Figure 5-12.

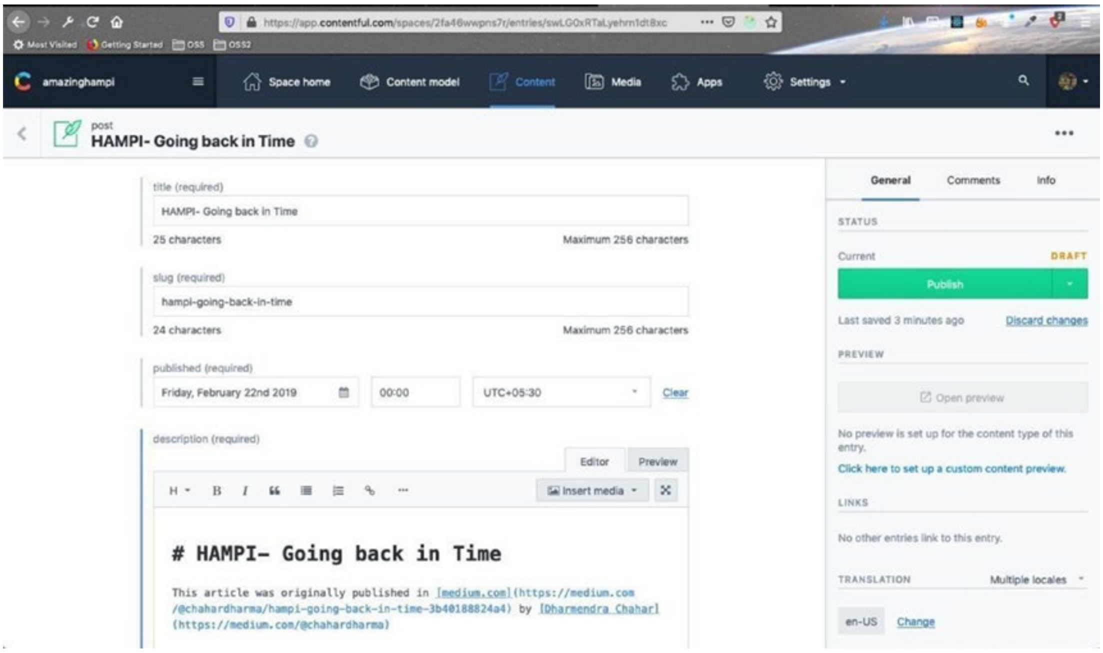


Name	Content Type	Updated	Author	
Hazara Rama Temple	amazing-hampi-data	Sun, 7:12 PM	Me	PUBLISHED
Vithala Temple	amazing-hampi-data	Sun, 7:09 PM	Me	PUBLISHED
Virupaksha Temple	amazing-hampi-data	Sun, 7:05 PM	Me	PUBLISHED
Lakshmi Narasimha	amazing-hampi-data	Sun, 1:23 PM	Me	PUBLISHED
Monkey Temple	amazing-hampi-data	Sat, 10:27 PM	Me	PUBLISHED
Sasivekalu Ganesh	amazing-hampi-data	Fri, 9:00 PM	Me	PUBLISHED
Riverside Ruins	amazing-hampi-data	Fri, 8:58 PM	Me	PUBLISHED
King's Balance	amazing-hampi-data	Fri, 8:55 PM	Me	PUBLISHED
Kadalekalu Ganesh	amazing-hampi-data	Fri, 8:46 PM	Me	PUBLISHED
Mahanavami Dibba	amazing-hampi-data	Fri, 7:51 PM	Me	PUBLISHED

***Figure 5-12.** Choose Post to add to the content model*

For this post I am taking real blogs from Medium. We will put them in the Description field, but it requires a Markdown file. We can convert the Medium post to Markdown with this[1] awesome `npm` package. It is shown in Figure 5-13.

[1] https://medium.com/@macropus/export-your-medium-posts-to-markdown-b5ccc8cb0050



post
HAMPI- Going back in Time

title (required)
HAMPI- Going back in Time
25 characters — Maximum 256 characters

slug (required)
hampi-going-back-in-time
24 characters — Maximum 256 characters

published (required)
Friday, February 22nd 2019 | 00:00 | UTC+05:30 | Clear

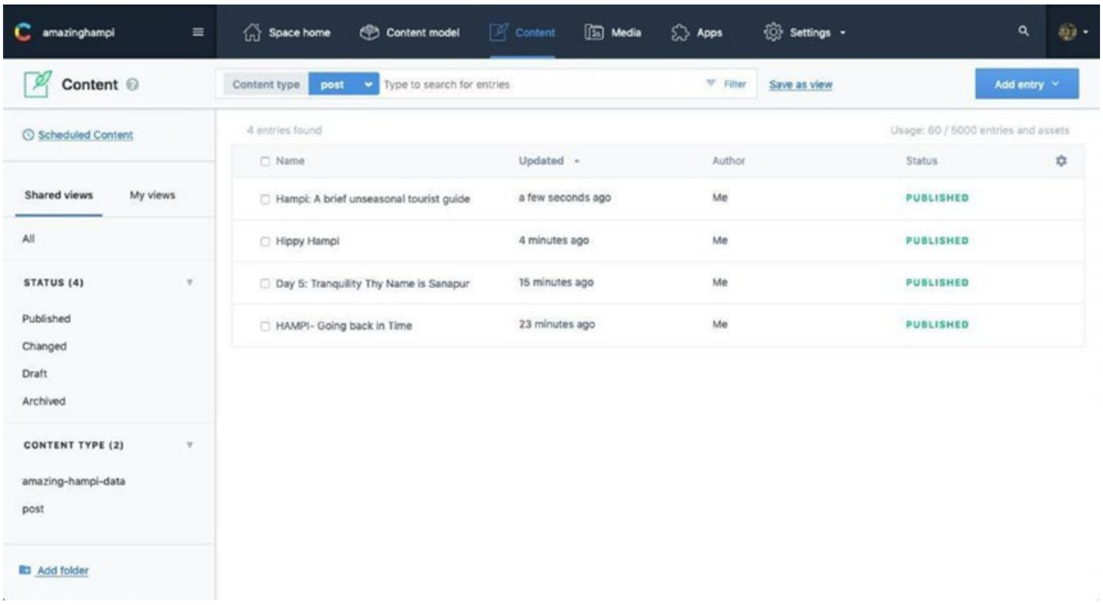
description (required)

```
# HAMPI- Going back in Time

This article was originally published in [medium.com](https://medium.com
/@chahardharma/hampi-going-back-in-time-3b40188824a4) by [Dharmendra Chahar]
(https://medium.com/@chahardharma)
```

STATUS
Current: DRAFT
Last saved 3 minutes ago
PREVIEW
No preview is set up for the content type of this entry.
Click here to set up a custom content preview.
LINKS
No other entries link to this entry.
TRANSLATION: Multiple locales
en-US

Figure 5-13. *The content*

I added four original blogs from Medium.com, as shown in Figure 5-14. I'll add more later, as I am also planning to visit Hampi soon.



4 entries found

Usage: 60 / 5000 entries and assets

Name	Updated	Author	Status
Hampi: A brief unseasonal tourist guide	a few seconds ago	Me	PUBLISHED
Hippy Hampi	4 minutes ago	Me	PUBLISHED
Day 5: Tranquility Thy Name is Sanapur	15 minutes ago	Me	PUBLISHED
HAMPI- Going back in Time	23 minutes ago	Me	PUBLISHED

Figure 5-14. *Hippy Hampi blogs*

One thing I forgot to add to the content model is an Image field. I will head over to the Content Model - Post and add a Media field. See Figure 5-15.

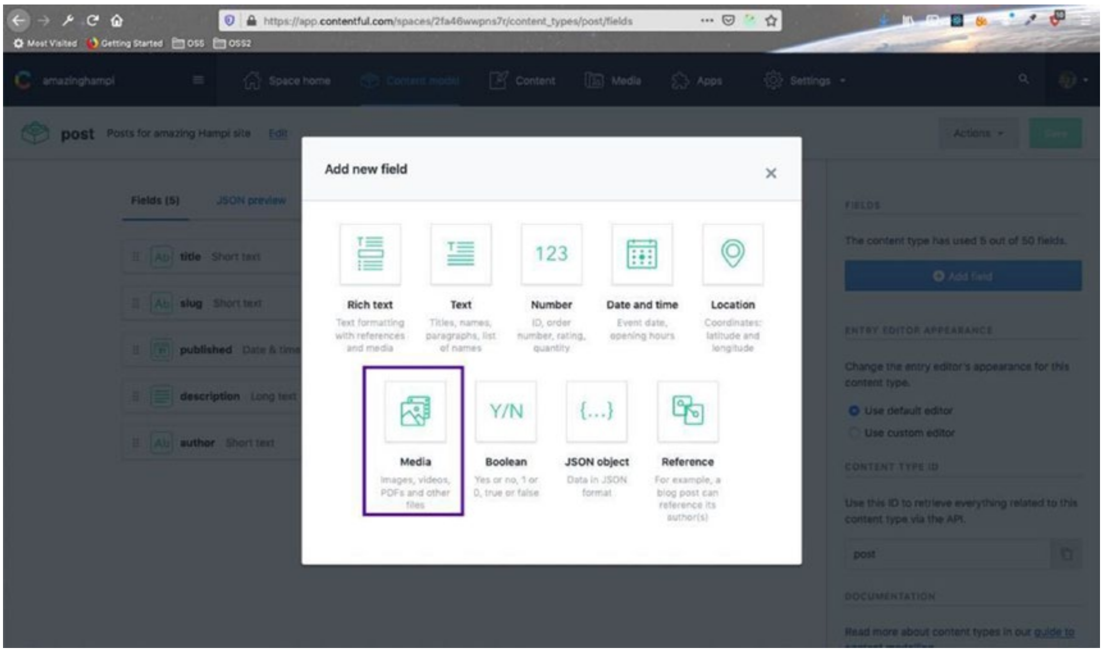
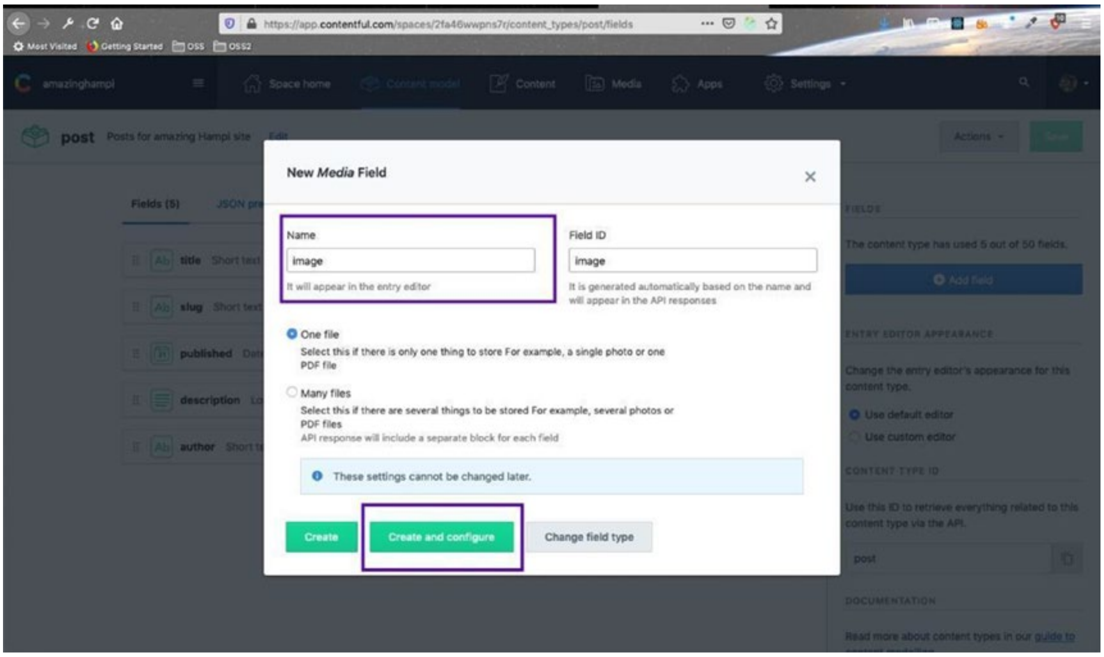

https://app.contentful.com/spaces/2fa46wwpns7r/content_types/post/fields
post
Posts for amazing Hampi site
Fields (5)
JSON preview
title Short text
slug Short text
published Date & time
description Long text
author Short text
Add new field
Rich text
Text formatting with references and media
Text
Titles, names, paragraphs, list of names
Number
ID, order number, rating, quantity
Date and time
Event date, opening hours
Location
Coordinates: latitude and longitude
Media
Images, videos, PDFs and other files
Boolean
Yes or no, 1 or 0, true or false
JSON object
Data in JSON format
Reference
For example, a blog post can reference its author(s)

Figure 5-15. *Add a Media field*

Give it the name `image` and then click on Create and Configure to make it a required field, as shown in Figure 5-16.



New *Media* Field

Name: image
It will appear in the entry editor

Field ID: image
It is generated automatically based on the name and will appear in the API responses

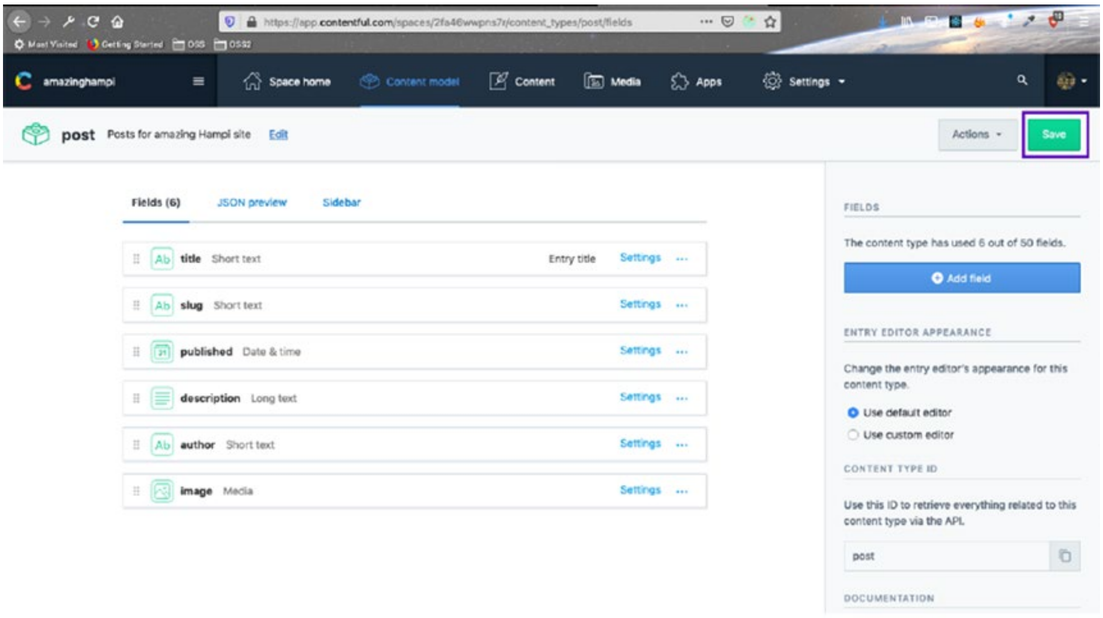
(●) One file
Select this if there is only one thing to store For example, a single photo or one PDF file

( ) Many files
Select this if there are several things to be stored For example, several photos or PDF files
API response will include a separate block for each field

These settings cannot be changed later.

Figure 5-16. *Name the field and make it required*

Next, save it by clicking the Save button, as shown in Figure 5-17.



https://app.contentful.com/spaces/2fa46wwpns7r/content_types/post/fields

post Posts for amazing Hampi site Edit

title	Short text	Entry title	Settings
slug	Short text		Settings
published	Date & time		Settings
description	Long text		Settings
author	Short text		Settings
image	Media		Settings

FIELDS

The content type has used 6 out of 50 fields.

ENTRY EDITOR APPEARANCE

Change the entry editor's appearance for this content type.

CONTENT TYPE ID

Use this ID to retrieve everything related to this content type via the API.

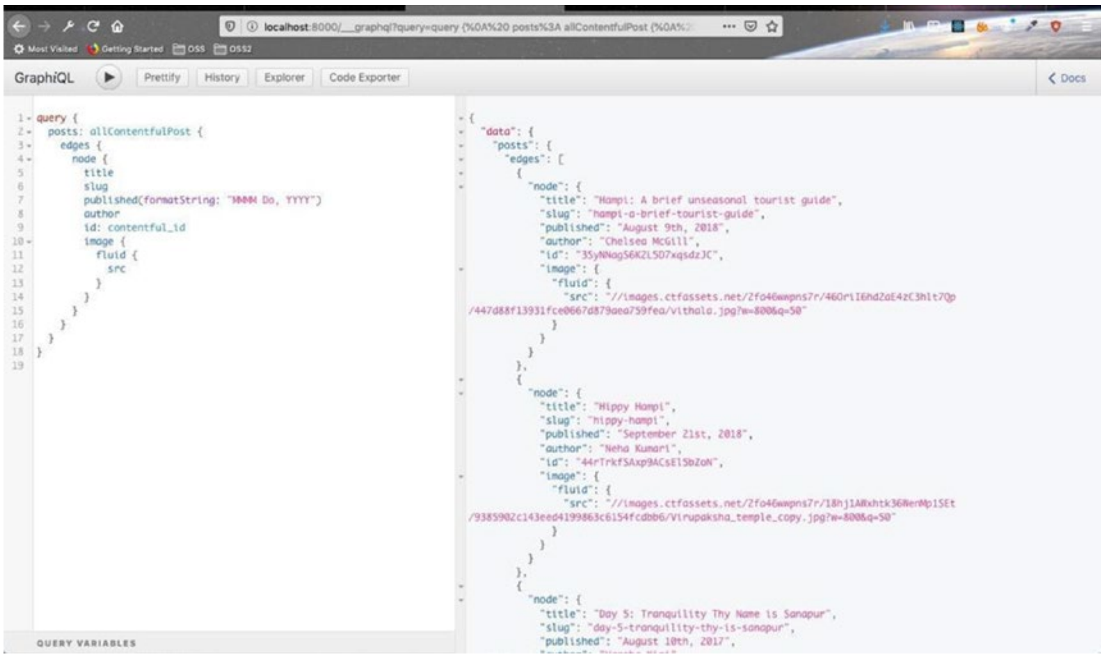
post

DOCUMENTATION

Figure 5-17. *Save the new field*

I also added the required image to all the posts. After that, to access these new posts, we need to restart the server. Go ahead and stop `gatsby develop` from the terminal and re-run it.

Also, refresh the GraphQL in the browser and write the query in Figure 5-18 to display all posts.

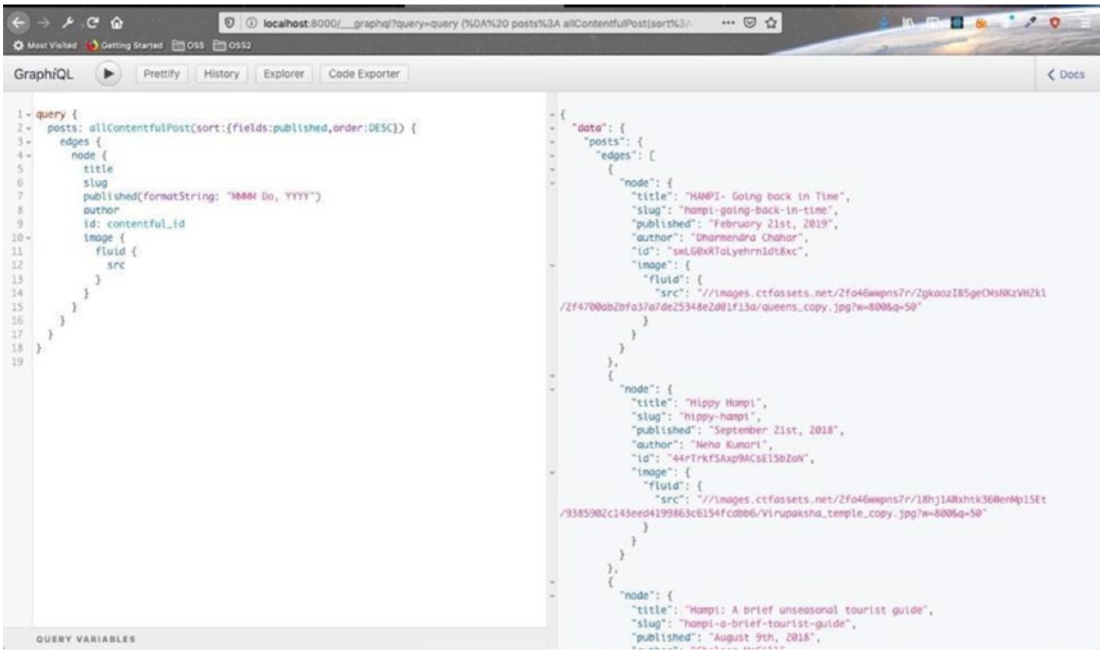


```
query {
  posts: allContentfulPost {
    edges {
      node {
        title
        slug
        published(formatString: "MMMM Do, YYYY")
        author
        id: contentful_id
        image {
          fluid {
            src
          }
        }
      }
    }
  }
}
```

```
{
  "data": {
    "posts": {
      "edges": [
        {
          "node": {
            "title": "Hampi: A brief unseasonal tourist guide",
            "slug": "hampi-a-brief-tourist-guide",
            "published": "August 9th, 2018",
            "author": "Chelsea McGill",
            "id": "35yNNagS6K2L5D7xqsdzJC",
            "image": {
              "fluid": {
                "src": "//images.ctfassets.net/2fo46wwpns7r/46OriI6hdZaE4zC3hlt7Qp
/447d88f13931fce0667d879aea759fea/vithala.jpg?w=800&q=50"
              }
            }
          }
        },
        {
          "node": {
            "title": "Hippy Hampi",
            "slug": "hippy-hampi",
            "published": "September 21st, 2018",
            "author": "Neha Kumari",
            "id": "44rTrkfSAxp9ACsEl5bZoN",
            "image": {
              "fluid": {
                "src": "//images.ctfassets.net/2fo46wwpns7r/18hj1A8xhtk36WenMp1SEt
/9385902c143eed4199863c6154fcdbb6/Virupaksha_temple_copy.jpg?w=800&q=50"
              }
            }
          }
        },
        {
          "node": {
            "title": "Day 5: Tranquility Thy Name is Sanapur",
            "slug": "day-5-tranquility-thy-is-sanapur",
            "published": "August 10th, 2017",
```

Figure 5-18. *Our query*

Before moving forward, I want to create another GraphQL query. Generally the blogs are posted in descending order, meaning that the newer blogs should come up. This is shown in Figure 5-19.


localhost:8000/___graphql?query=query {%0A%20 posts%3A allContentfulPost(sort%3A
GraphiQL
Prettify
History
Explorer
Code Exporter
Docs
QUERY VARIABLES

```
query {
  posts: allContentfulPost(sort:{fields:published,order:DESC}) {
    edges {
      node {
        title
        slug
        published(formatString: "MMMM Do, YYYY")
        author
        id: contentful_id
        image {
          fluid {
            src
          }
        }
      }
    }
  }
}
```

```
{
  "data": {
    "posts": {
      "edges": [
        {
          "node": {
            "title": "HAMPI- Going back in Time",
            "slug": "hampi-going-back-in-time",
            "published": "February 21st, 2019",
            "author": "Dharmendra Chahar",
            "id": "swLG0xRTaLyehrn1dt8xc",
            "image": {
              "fluid": {
                "src": "//images.ctfassets.net/2fo46wwpns7r/2gkaozIB5geCMsNKzVHZk1
/2f4700ab2bfa37a7de25348e2d01f13a/queens_copy.jpg?w=800&q=50"
              }
            }
          }
        },
        {
          "node": {
            "title": "Hippy Hampi",
            "slug": "hippy-hampi",
            "published": "September 21st, 2018",
            "author": "Neha Kumari",
            "id": "44rTrkfSAxp9ACsEl5bZoN",
            "image": {
              "fluid": {
                "src": "//images.ctfassets.net/2fo46wwpns7r/18hj1AWxhtk36WenMp15Et
/9385902c143eed4199863c6154fcdbb6/Virupaksha_temple_copy.jpg?w=800&q=50"
              }
            }
          }
        },
        {
          "node": {
            "title": "Hampi: A brief unseasonal tourist guide",
            "slug": "hampi-a-brief-tourist-guide",
            "published": "August 9th, 2018",
```

Figure 5-19. *The posts are listed in descending order*

Displaying the Blog Component

It's time to write the code to display the Blog component. Create a new `blog` folder inside the **components** folder. Inside it, create two files called `BlogList.js` and `BlogCard.js`.

Let's first add the CSS required for this component in the `css` folder. Add a file called `blog.module.css` to the `css` folder and add the contents in Listing 5-1 to it.

Listing 5-1. The blog.module.css File

```
.blog {
  padding: 4rem 0;
}
.center {
  width: 80vw;
  margin: 3rem auto;
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(280px, 1fr));
  grid-column-gap: 2rem;
  grid-row-gap: 2rem;
}
.links {
  width: 80vw;
  margin: 0 auto 5rem auto;
  display: flex;
  flex-wrap: wrap;
  justify-content: space-between;
}
.link {
  text-transform: uppercase;
  letter-spacing: var(--mainSpacing);
  background: var(--primaryColor);
  color: var(--mainWhite);
  border: 2px solid var(--primaryColor);
  padding: 0.25rem 0.5rem;
  border-radius: 0.5rem;
  display: inline-block;
  transition: var(--mainTransition);
  cursor: pointer;
}
.link:hover {
  background: transparent;
  color: var(--primaryColor);
}
.active {
  background: var(--mainWhite);
  color: var(--primaryColor);
}

@media screen and (min-width: 576px) {
  .center {
    grid-template-columns: repeat(auto-fill, minmax(368.66px, 1fr));
  }
  .links {
    width: 60vw;
  }
}
@media screen and (min-width: 1200px) {
  .center {
    width: 100%;
    max-width: 1170px;
  }
}
```

For now, add this dummy content to `BlogCard.js`, which we will change soon. The code is shown in Listing 5-2.

Listing 5-2. Dummy Content for BlogCard.js

```
import React from "react"

const BlogCard = () => {
    return <div>This is a blog card</div>
}

export default BlogCard
```

Next, add the content in Listing 5-3 to the `BlogList.js` file. Here, we are using the GraphQL query we created earlier in our GraphiQL playground. We are mapping through the responses we are getting from this query.

Listing 5-3. The BlogList.js File

```
import React from "react"
import BlogCard from "./BlogCard"
import Title from "../Title"
import { useStaticQuery, graphql } from "gatsby"
import styles from "../../css/blog.module.css"

const getPosts = graphql`
query {
posts: allContentfulPost(sort:{fields:published,order:DESC}) {
    edges {
        node {
            title
            slug
            published(formatString: "MMMM Do, YYYY")
            author
            id: contentful_id
            image {
                fluid {
                    ...GatsbyContentfulFluid
                }
            }
        }
    }
}
}
`
const BlogList = () => {
    const { posts } = useStaticQuery(getPosts)

    return (
        <section className={styles.blog}>
            <Title title="hampi" subtitle="blogs" />
            <div className={styles.center}>
                {posts.edges.map(({ node }) => {
                    return <BlogCard key={node.id} blog={node} />
                })}
            </div>
        </section>
    )
}

export default BlogList
```

Finally, add this component to the `blog.js` file so that it can be rendered when we click on the Blog page. The component's code is highlighted in bold in Listing 5-4.

Listing 5-4. The BlogList in blog.js

```
import React from 'react'
import Layout from "../components/Layout"
import StyledHero from "../components/StyledHero"
import { graphql } from 'gatsby'
import BlogList from '../components/Blog/BlogList'

export const query = graphql`
query {
    blogBcg: file(relativePath: {eq: "blogBcg.jpeg"}) {
        childImageSharp {
            fluid(quality: 90, maxWidth: 4160) {
                ...GatsbyImageSharpFluid_withWebp
            }
        }
    }
}
`;

export default function blog({ data }) {
    return (
        <Layout>
            <StyledHero img={data.blogBcg.childImageSharp.fluid} />
            <BlogList />
        </Layout>
    )
}
```

Creating the BlogCard Component

We will work on the `BlogCard` component now. Let's start by adding some styles to it. Create a new file called `blog-card.module.css` inside the `css` folder. Add the content in Listing 5-5 to it.

Listing 5-5. The blog-card.module.css File

```
.blog {
  box-shadow: var(--lightShadow);
  transition: var(--mainTransition);
}
.blog:hover {
  box-shadow: var(--darkShadow);
}

.img-container {
  position: relative;
  background: var(--primaryColor);
  transition: var(--mainTransition);
}
.img {
  transition: var(--mainTransition);
}
.img-container:hover .img {
  opacity: 0.3;
}

.link {
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  opacity: 0;
  text-transform: uppercase;
  letter-spacing: var(--mainSpacing);
  color: var(--mainWhite);
  border: 2px solid var(--mainWhite);
  padding: 0.5rem 0.7rem;
  display: inline-block;
  transition: var(--mainTransition);
  cursor: pointer;
}
```

```
.link:hover {
  background: var(--mainWhite);
  color: var(--primaryColor);
}
.img-container:hover .link {
  opacity: 1;
}

.footer {
  padding: 1rem;
  text-align: left;
}
.footer h4 {
  text-transform: capitalize;
  margin-bottom: 0;
}
.date {
  position: absolute;
  left: 0;
  top: 75%;
  background: var(--primaryColor);
  padding: 0.3rem 0.5rem;
  border-top-right-radius: 1rem;
  border-bottom-right-radius: 1rem;
}
```

Next, let's update the `BlogCard.js` file with real code, as shown in Listing 5-6. Here, we are just adding some styles from the `css` folder. We are getting the `blog` props from the `BlogList` component. We are destructuring it by taking the title image and publishing it to show the blog.

Once we click on the post, we will use the slug to display it. (We will do that in the next section.)

Listing 5-6. Code in BlogCard.js

```
import React from "react"
import styles from "../../css/blog-card.module.css"
import Image from "gatsby-image"
import AniLink from "gatsby-plugin-transition-link/AniLink"

const BlogCard = ({ blog }) => {
    const { slug, title, image, published } = blog
    return (
        <article className={styles.blog}>
            <div className={styles.imgContainer}>
                <Image fluid={image.fluid} className={styles.img}
                alt="single post" />
                <AniLink fade className={styles.link} to={`/blog/${slug}`}>
                    read more
                </AniLink>
                <h6 className={styles.date}>{published}</h6>
            </div>
            <div className={styles.footer}>
                <h4>{title}</h4>
            </div>
        </article>
    )
}

export default BlogCard
```

This code will show all four blogs from Contentful, as shown in Figure 5-20.

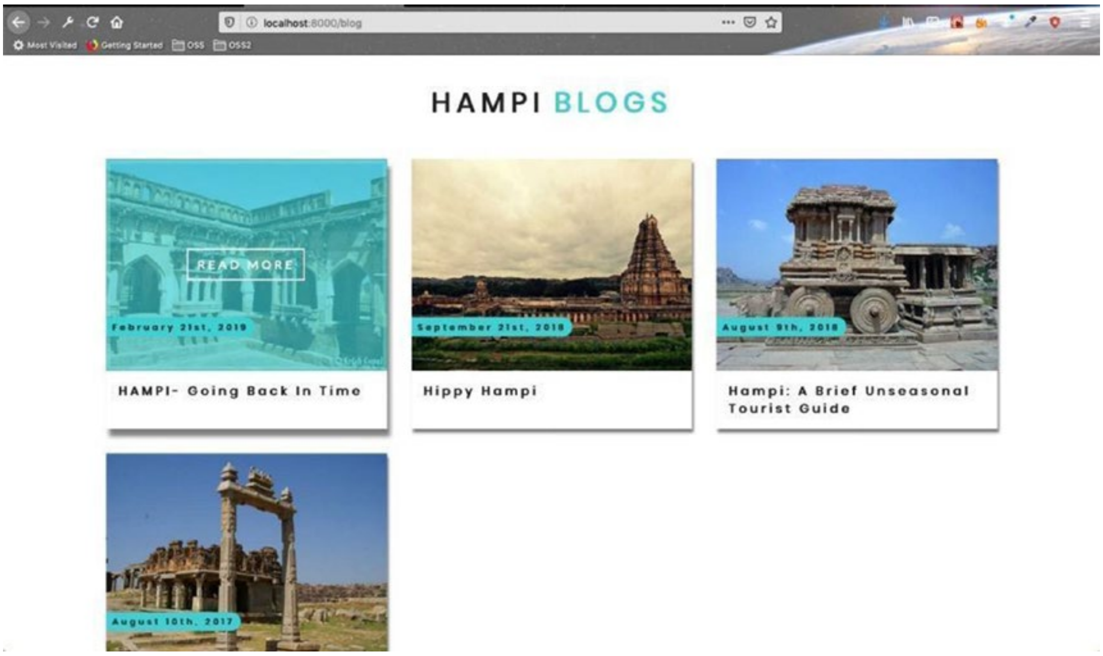


Figure 5-20. *Hampi blogs shown in a browser*

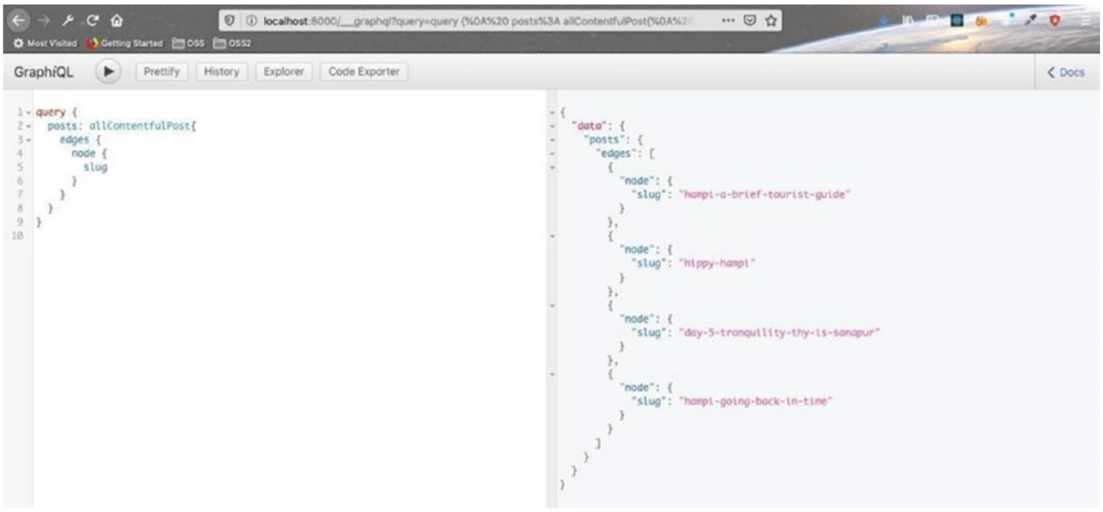
Creating the Single Blog Page

We will now show the blog pages when a user clicks on them. To do this, we will follow the same template approach that we followed on the tour. Create a file called `blog-template.js` inside the `templates` folder. For now, add the dummy data from Listing 5-7 to it.

Listing 5-7. The Dummy Code for blog-template.js

```
import React from "react"

const Blog = () => {
    return <div>this is single blog page</div>
}

export default Blog
```

Let's write our GraphQL query in the GraphiQL playground, as shown in Figure 5-21.



```
query {
  posts: allContentfulPost{
    edges {
      node {
        slug
      }
    }
  }
}
```

```
{
  "data": {
    "posts": {
      "edges": [
        {
          "node": {
            "slug": "hampi-a-brief-tourist-guide"
          }
        },
        {
          "node": {
            "slug": "hippy-hampi"
          }
        },
        {
          "node": {
            "slug": "day-5-tranquility-thy-is-sanapur"
          }
        },
        {
          "node": {
            "slug": "hampi-going-back-in-time"
          }
        }
      ]
    }
  }
}
```

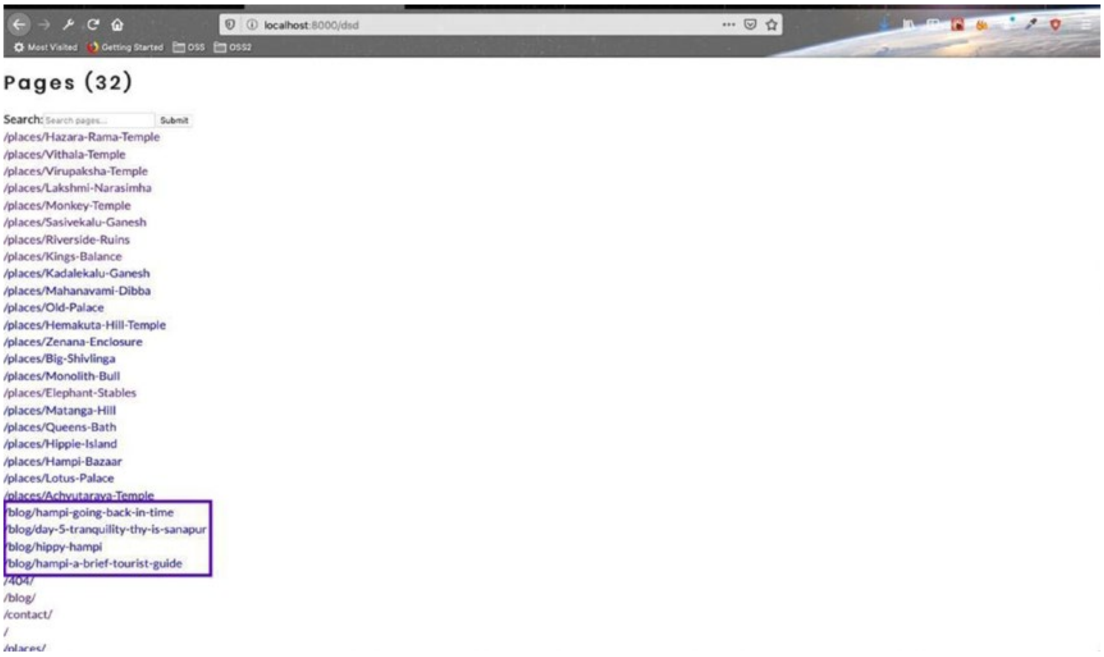
Figure 5-21. *The GraphQL query*

Next, open `gatsby-node.js` and add this new query. We will also loop through the node and create the page. The updated code is marked in bold in Listing 5-8.

Listing 5-8. Blog in gatsby-node.js

```
const path = require("path")

exports.createPages = async ({ graphql, actions }) => {
const { createPage } = actions

const { data } = await graphql(`
query {
    places: allContentfulAmazingHampiData {
        edges {
        node {
                slug
            }
        }
    }
    posts: allContentfulPost {
        edges {
            node {
                slug
            }
        }
    }
}
`)

data.places.edges.forEach(({ node }) => {
    createPage({
            path: `places/${node.slug}`,
            component: path.resolve("./src/templates/place-template.js"),
            context: {
                slug: node.slug,
            },
        })
    })
    data.posts.edges.forEach(({ node }) => {
        createPage({
            path: `blog/${node.slug}`,
            component: path.resolve("./src/templates/blog-template.js"),
            context: {
                slug: node.slug,
            },
        })
    })
}
```

To test whether everything is working, we need to stop and start `gatsby develop` in the terminal. After that, move to any nonexistent page in the browser and you can see that those pages have been created. See Figure 5-22.



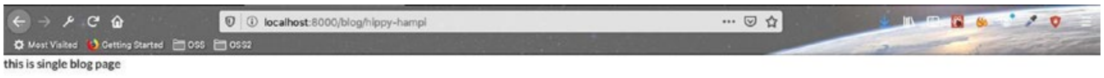
Pages (32)

Search: Search pages... Submit

/places/Hazara-Rama-Temple
/places/Vithala-Temple
/places/Virupaksha-Temple
/places/Lakshmi-Narasimha
/places/Monkey-Temple
/places/Sasivekalu-Ganesh
/places/Riverside-Ruins
/places/Kings-Balance
/places/Kadalekalu-Ganesh
/places/Mahanavami-Dibba
/places/Old-Palace
/places/Hemakuta-Hill-Temple
/places/Zenana-Enclosure
/places/Big-Shivlinga
/places/Monolith-Bull
/places/Elephant-Stables
/places/Matanga-Hill
/places/Queens-Bath
/places/Hippie-Island
/places/Hampi-Bazaar
/places/Lotus-Palace
/places/Achyutaraya-Temple
/blog/hampi-going-back-in-time
/blog/day-5-tranquility-thy-is-sanapur
/blog/hippy-hampi
/blog/hampi-a-brief-tourist-guide
/404/
/blog/
/contact/
/
/places/

Figure 5-22. *New pages were created*

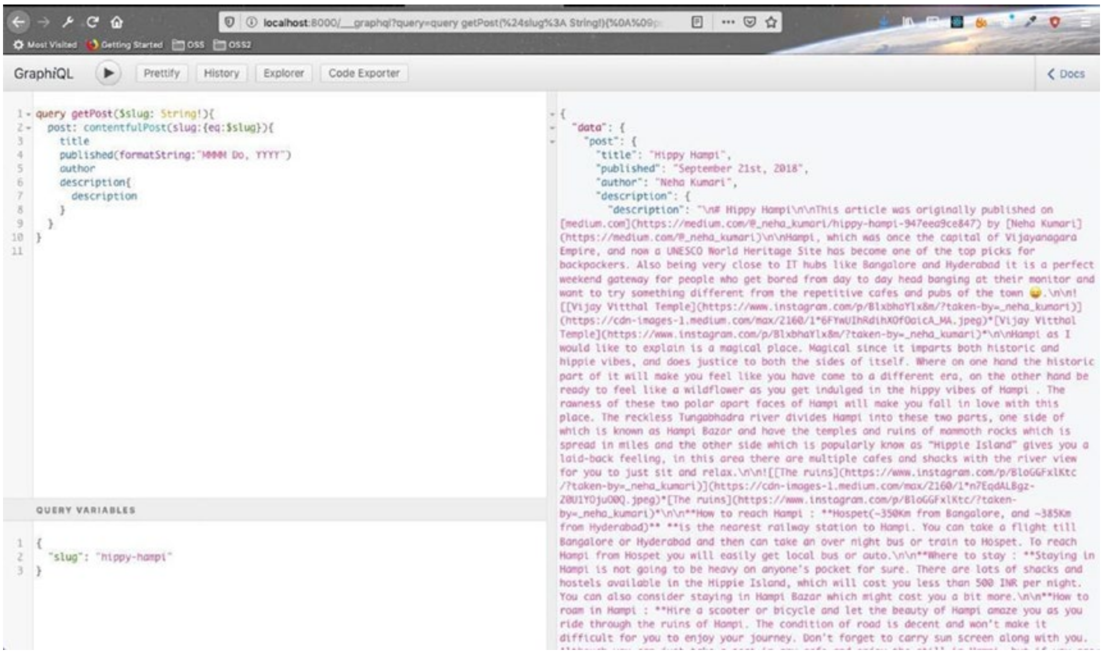
Click on any one of them and you will see the dummy text, as shown in Figure 5-23.



this is single blog page

Figure 5-23. *The dummy text is shown for now*

Let's create the query to show the single blog post in the playground, as shown in Figure 5-24.



```
query getPost($slug: String!){
  post: contentfulPost(slug:{eq:$slug}){
    title
    published(formatString:"MMMM Do, YYYY")
    author
    description{
      description
    }
  }
}
```

```
{
  "slug": "hippy-hampi"
}
```

```
{
  "data": {
    "post": {
      "title": "Hippy Hampi",
      "published": "September 21st, 2018",
      "author": "Neha Kumari",
      "description": {
        "description": "\n# Hippy Hampi\n\nThis article was originally published on
[medium.com](https://medium.com/@_neha_kumari/hippy-hampi-947eea9ce847) by [Neha Kumari]
(https://medium.com/@_neha_kumari)\n\nHampi, which was once the capital of Vijayanagara
Empire, and now a UNESCO World Heritage Site has become one of the top picks for
backpackers. Also being very close to IT hubs like Bangalore and Hyderabad it is a perfect
weekend gateway for people who get bored from day to day head banging at their monitor and
want to try something different from the repetitive cafes and pubs of the town 😜.\n\n!
[[Vijay Vitthal Temple](https://www.instagram.com/p/BlxbhaYlx8m/?taken-by=_neha_kumari)]
(https://cdn-images-1.medium.com/max/2160/1*6FYwUIhRdihXOfOaicA_MA.jpeg)*[Vijay Vitthal
Temple](https://www.instagram.com/p/BlxbhaYlx8m/?taken-by=_neha_kumari)*\n\nHampi as I
would like to explain is a magical place. Magical since it imparts both historic and
hippie vibes, and does justice to both the sides of itself. Where on one hand the historic
part of it will make you feel like you have come to a different era, on the other hand be
ready to feel like a wildflower as you get indulged in the hippy vibes of Hampi . The
rawness of these two polar apart faces of Hampi will make you fall in love with this
place. The reckless Tungabhadra river divides Hampi into these two parts, one side of
which is known as Hampi Bazar and have the temples and ruins of mammoth rocks which is
spread in miles and the other side which is popularly know as "Hippie Island" gives you a
laid-back feeling, in this area there are multiple cafes and shacks with the river view
for you to just sit and relax.\n\n![[The ruins](https://www.instagram.com/p/BloGGFxlKtc
/?taken-by=_neha_kumari)](https://cdn-images-1.medium.com/max/2160/1*n7EqdALBgz-
Z0U1YOjuO0Q.jpeg)*[The ruins](https://www.instagram.com/p/BloGGFxlKtc/?taken-
by=_neha_kumari)*\n\n**How to reach Hampi : **Hospet(~350Km from Bangalore, and ~385Km
from Hyderabad)** **is the nearest railway station to Hampi. You can take a flight till
Bangalore or Hyderabad and then can take an over night bus or train to Hospet. To reach
Hampi from Hospet you will easily get local bus or auto.\n\n**Where to stay : **Staying in
Hampi is not going to be heavy on anyone's pocket for sure. There are lots of shacks and
hostels available in the Hippie Island, which will cost you less than 500 INR per night.
You can also consider staying in Hampi Bazar which might cost you a bit more.\n\n**How to
roam in Hampi : **Hire a scooter or bicycle and let the beauty of Hampi amaze you as you
ride through the ruins of Hampi. The condition of road is decent and won't make it
difficult for you to enjoy your journey. Don't forget to carry sun screen along with you.
```

Figure 5-24. *Query to show the single blog post in the playground*

Let's also add the styles to the `css` folder and call it `single-blog.module.css`. The contents are shown in Listing 5-9.

Listing 5-9. The single-blog.module.css File

```
.blog {
  padding: 4rem 0;
}
.center {
  width: 80vw;
  margin: 0 auto;
}
.blog h1,
.blog h4 {
  text-transform: capitalize;
}
.post {
  margin: 2rem 0;
}
```

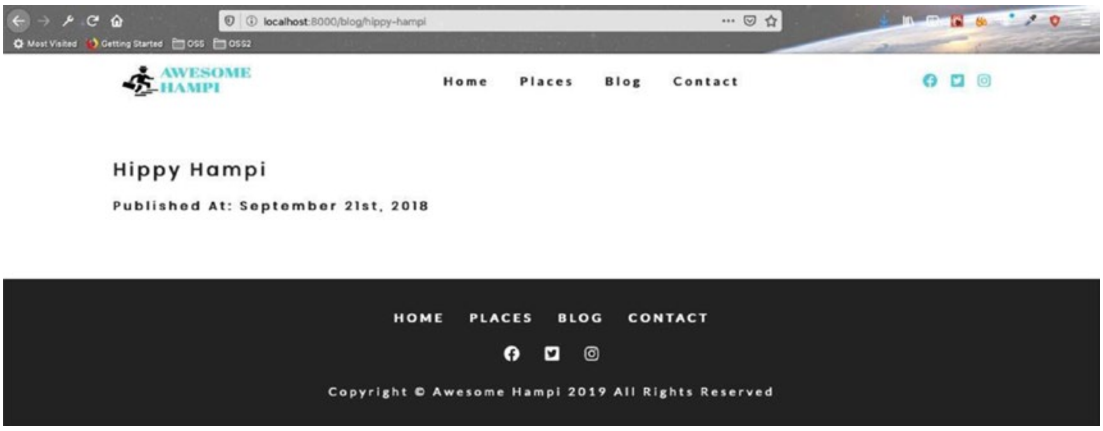
```
.post img {
  max-width: 70vw;
}
```

We will now start to update our `blog-template.js` file. We are using the query we created in the playground. After that, we destructure the data we are receiving. We are right now showing the Title and Published fields in the blog. The code is shown in Listing 5-10.

Listing 5-10. The Updated blog-template.js File

```
import React from "react"
import { graphql } from "gatsby"
import Layout from "../components/Layout"
import styles from "../css/single-blog.module.css"
import AniLink from "gatsby-plugin-transition-link/AniLink"

const Blog = ({ data }) => {
    const { title, published, author, description: {description}} = data.post;
    return <Layout>
                <section className={styles.blog}>
                    <div className={styles.center}>
<h1>{title}</h1>
                        <h4>Published at: {published}</h4>
                    </div>
                </section>
        </Layout>
}

export const query = graphql`
    query getPost($slug: String!){
    post: contentfulPost(slug:{eq:$slug}){
    title
    published(formatString:"MMMM Do, YYYY")
    author
    description{
        description
    }
}
}
`;

export default Blog
```

When we head over to a blog post, we can see that it is rendered properly, as shown in Figure 5-25.


localhost:8000/blog/hippy-hampi
AWESOME HAMPI
Home
Places
Blog
Contact
HOME
PLACES
BLOG
CONTACT
Copyright © Awesome Hampi 2019 All Rights Reserved

Hippy Hampi

Published At: September 21st, 2018

Figure 5-25. Hippy hampi is being properly rendered

Let's also add an Author field and a link to go back to the Blog page. The updated code is marked in bold in Listing 5-11.

Listing 5-11. Author and Link in the blog-template.js File

```
...
import styles from "../css/single-blog.module.css"
import AniLink from "gatsby-plugin-transition-link/AniLink"

const Blog = ({ data }) => {
    const { title, published, author, description: {description}} = data.post;
    return <Layout>
                <section className={styles.blog}>
                    <div className={styles.center}>
                 <h1>{title}</h1>
                        <h4>Published at: {published}</h4>
                        <h4>Author: {author}</h4>
```

```
                              <AniLink fade to="/blog" className="btn-
                              primary">all blogs</AniLink>
                          </div>
                      </section>
              </Layout>
}
...
```

To display the Markdown file that we are getting in the Description field, we need to do some configuration. I found this[2] blog, which was helpful to me to do the setup.

To display Markdown files, we need to `npm install` the `gatsby-transformer-remark` plugin. Head over to the terminal and stop the `gatsby develop`. Then install the plugin by running the `npm install --save gatsby-transformer-remark` command. Finally, add it to the `gatsby-config.js` file, as shown in Listing 5-12.

Listing 5-12. The gatsby-transformer-remark Plugin Added to gatsby-config.js

```
require("dotenv").config({
  path: `.env.${process.env.NODE_ENV}`,
})

  plugins: [
...
...
         ,
    `gatsby-plugin-sitemap`,
    `gatsby-plugin-styled-components`,
    `gatsby-transformer-sharp`,
    `gatsby-plugin-sharp`,
    `gatsby-plugin-transition-link`,
    `gatsby-transformer-remark`
  ]
}
```

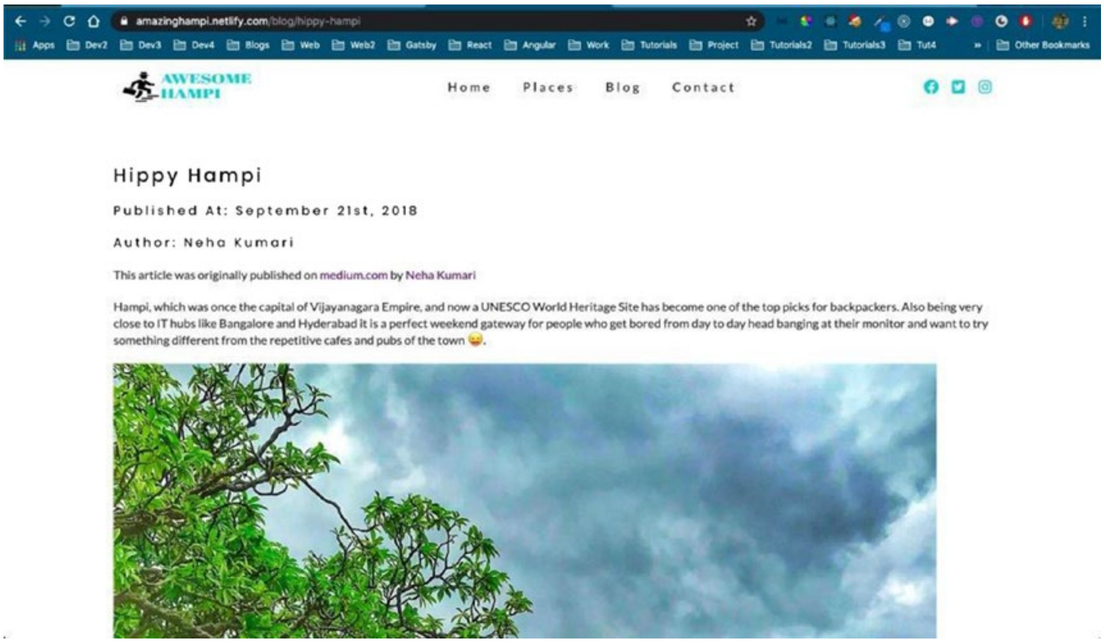
Next, move back to your `blog-template.js` file. Here we need to update the query for description a bit to add a transformer plugin. The updated code is highlighted in Listing 5-13.

[2] https://codebushi.com/gatsby-with-contentful-cms/

Listing 5-13. Markdown in blog-template.js

```
const Blog = ({ data }) => {
    const { title, published, author, description: {childMarkdownRemark}} =
    data.post;
    return <Layout>
                <section className={styles.blog}>
                    <div className={styles.center}
          <h1>{title}</h1>
                        <h4>Published at: {published}</h4>
                        <h4>Author: {author}</h4>
                        <div dangerouslySetInnerHTML={{__
                        html:childMarkdownRemark.html}} />
                        <AniLink fade to="/blog" className="btn-primary">
                        all blogs</AniLink>
                    </div>
                </section>
        </Layout>
}

export const query = graphql`
    query getPost($slug: String!){
          post: contentfulPost(slug:{eq:$slug}){
    title
    published(formatString:"MMMM Do, YYYY")
    author
    description{
        childMarkdownRemark{
            html
        }
    }
}
}
`;

export default Blog
```

Start your `gatsby develop` from the terminal and then head over to a blog post. You can see that the Markdown rendered successfully, as shown in Figure 5-26.



Hippy Hampi

Published At: September 21st, 2018

Author: Neha Kumari

This article was originally published on medium.com by Neha Kumari

Hampi, which was once the capital of Vijayanagara Empire, and now a UNESCO World Heritage Site has become one of the top picks for backpackers. Also being very close to IT hubs like Bangalore and Hyderabad it is a perfect weekend gateway for people who get bored from day to day head banging at their monitor and want to try something different from the repetitive cafes and pubs of the town 😜.

Figure 5-26. *Awesome Hampi has rendered properly*

I have been in Hampi for the past four days and created six blogs on Hampi. The four day blogs can be found here[3]. My other blogs (on saving money during your Hampi stay) can be found here[4]. Yet another blog on Rama and Shiva in Hampi can be found here[5]. I added these blogs in Contentful so they are reflected on the site, as shown in Figure 5-27.

[3] `https://medium.com/@nabendu82/my-hampi-vacation-day1-b0a2b7e26cbf`

[4] `https://medium.com/@nabendu82/how-to-save-money-and-travel-in-hampi-495a5d3f2415`

[5] `https://medium.com/@nabendu82/rama-and-shiva-in-hampi-b5cce3ac5496`

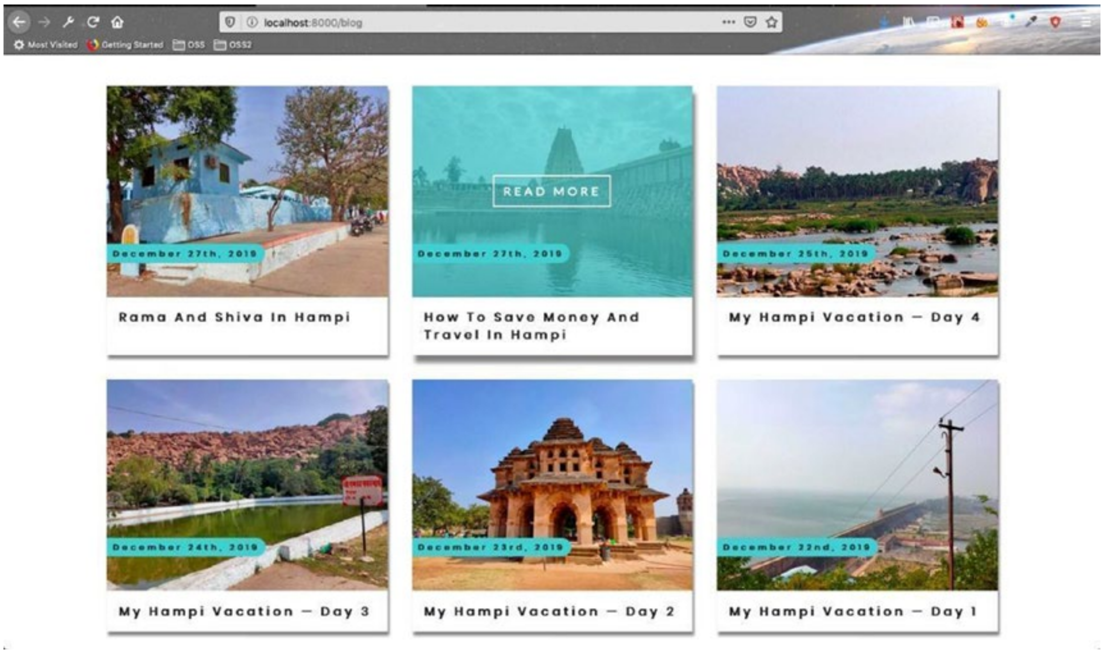


Figure 5-27. *Blogs have been added*

Similarly, I added many new places and some missing pictures for all the places, as shown in Figure 5-28.

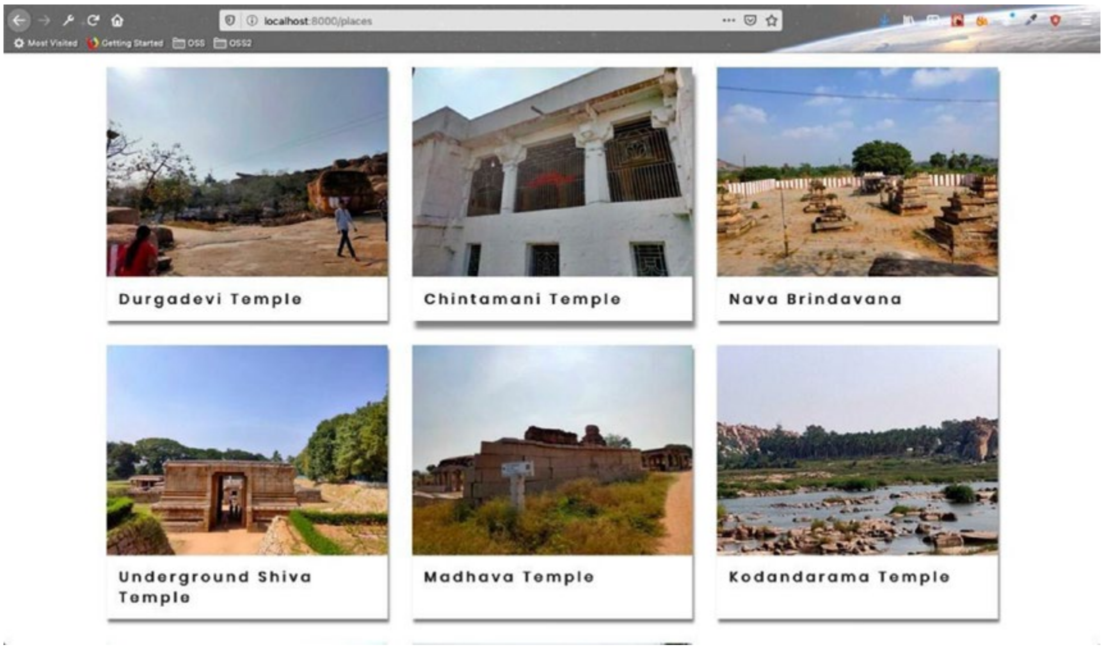


Figure 5-28. *Places and images were also added*

Upon checking the blog, I found that it will look nice if it contained the header image like the place.

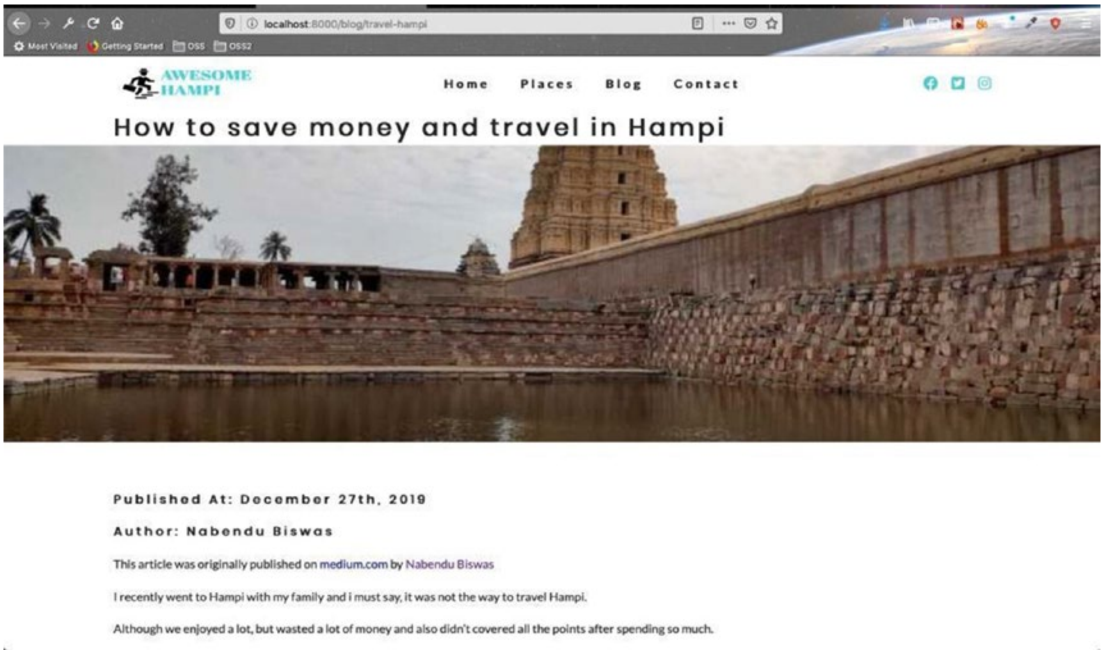
As we did with the `place-template.js` file, we need to add `StyledHero` to `blog-template.js`. The updated code is marked in bold in Listing 5-14.

Listing 5-14. StyledHero in blog-template.js

```
...
...
import AniLink from "gatsby-plugin-transition-link/AniLink"
import StyledHero from "../components/StyledHero"

const Blog = ({ data }) => {
    const { title, published, author, description: {childMarkdownRemark},
    image} = data.post;
    return <Layout>
                <h1 className={styles.center}>{title}</h1>
                <StyledHero img={image.fluid} />
                <section className={styles.blog
```

```
                        <div className={styles.center}>
                        <h4>Published at: {published}</h4>
                        <h4>Author: {author}</h4>
                        ...
                       </div>
                </section>
        </Layout>
}

export const query = graphql`
    query getPost($slug: String!){
         post: contentfulPost(slug:{eq:$slug}){
    title
    published(formatString:"MMMM Do, YYYY")
    author
    description{
        childMarkdownRemark{
            html
        }
    }
    image {
            fluid {
                ...GatsbyContentfulFluid
            }
        }
}
}
`;

export default Blog
```

This code will show a nice header image with every blog post, as shown in Figure 5-29.



How to save money and travel in Hampi

Published At: December 27th, 2019

Author: Nabendu Biswas

This article was originally published on medium.com by Nabendu Biswas

I recently went to Hampi with my family and i must say, it was not the way to travel Hampi.

Although we enjoyed a lot, but wasted a lot of money and also didn't covered all the points after spending so much.

Figure 5-29. *This header image is shown with every blog post*

Next, we will commit and push the changes to GitHub for the automatic deployment in Netlify to start. Our site is live at `https://amazinghampi.netlify.com/`,[6] as shown in Figure 5-30.

[6]`https://amazinghampi.netlify.com/`

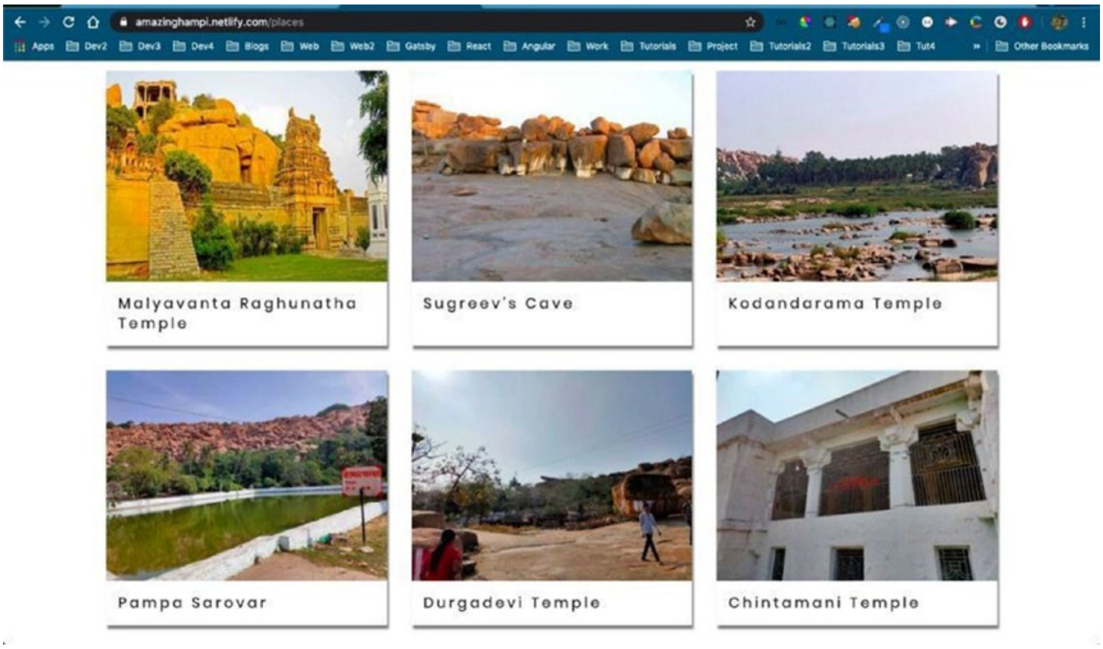


Figure 5-30. *The live amazinghampi site*

Creating the Photos Component

In this section, we'll create a new page called Photos. It will contain most of the photos that I took during my Hampi vacation[7]. All of these photos are royalty free, so feel free to use them. To create a new link in the navbar and footer components, we just need to add it to the links.js file. The changed code is marked in bold in Listing 5-15.

Listing 5-15. Photos in links.js

```
export default [
  {
    path: "/",
    text: "home",
  },
  {
    path: "/places",
```

[7]https://medium.com/@nabendu82/my-hampi-vacation-day1-b0a2b7e26cbf

```
    text: "places",
  },
  {
    path: "/blog",
    text: "blog",
  },
  {
    path: "/photos",
    text: "photos",
  },
  {
    path: "/contact",
    text: "contact",
  }
]
```

This code will be added to the navbar and the footer on the website, as shown in Figure 5-31.

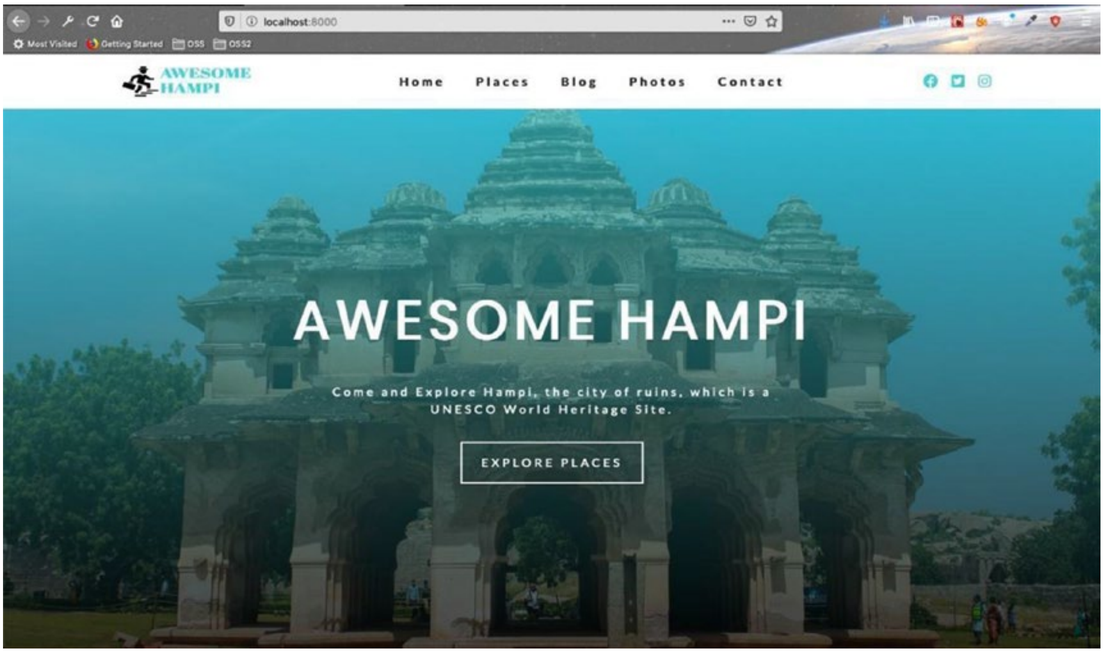

localhost:8000
AWESOME HAMPI
Home
Places
Blog
Photos
Contact
AWESOME HAMPI
Come and Explore Hampi, the city of ruins, which is a UNESCO World Heritage Site.
EXPLORE PLACES

***Figure 5-31.** The new link is added*

Let's create a photos.js file inside the pages folder. It is similar to the blog.js page, except the image is hampiPhoto.jpg in this case. The code is shown in Listing 5-16.

Listing 5-16. The photos.js File

```
import React from 'react'
import Layout from "../components/Layout"
import StyledHero from "../components/StyledHero"
import { graphql } from 'gatsby'
import PhotoList from '../components/Photos/PhotoList'

export const query = graphql`
query {
    blogBcg: file(relativePath: {eq: "hampiPhoto.jpg"}) {
        childImageSharp {
            fluid(quality: 90, maxWidth: 4160) {
                ...GatsbyImageSharpFluid_withWebp
            }
        }
    }
}
`;

export default function photos({ data }) {
    return (
        <Layout>
            <StyledHero img={data.blogBcg.childImageSharp.fluid} />
            <PhotoList />
        </Layout>
    )
}
```

Next, let's create a folder called photos inside the components directory. Then create a file called PhotoList.js inside it. The dummy code we use for now is shown in Listing 5-17.

Listing 5-17. Dummy Code for PhotoList.js

```
import React from 'react'

const PhotoList = () => {
    return (
             <div>PhotoList component</div>
        )
}
export default PhotoList;
```

When we open the Photos page, it displays everything correctly, as shown in Figure 5-32.

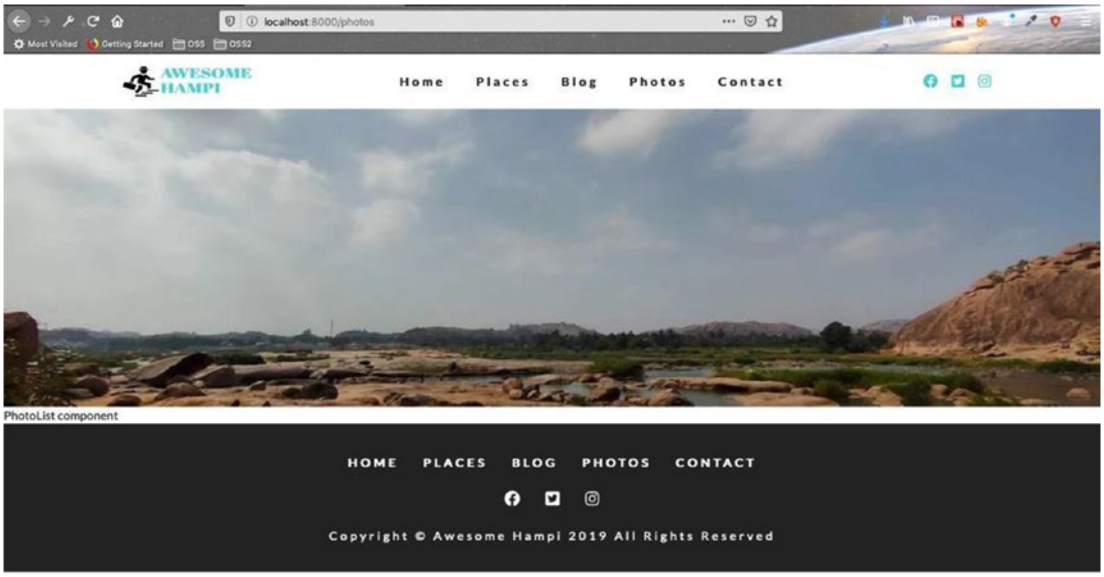


Figure 5-32. *The Photos page works as expected*

Setting Up Contentful for the Photos Component

Before adding code to the file, we need to create content in Contentful and test the query. Head over to Contentful and create a new content model, as shown in Figure 5-33.

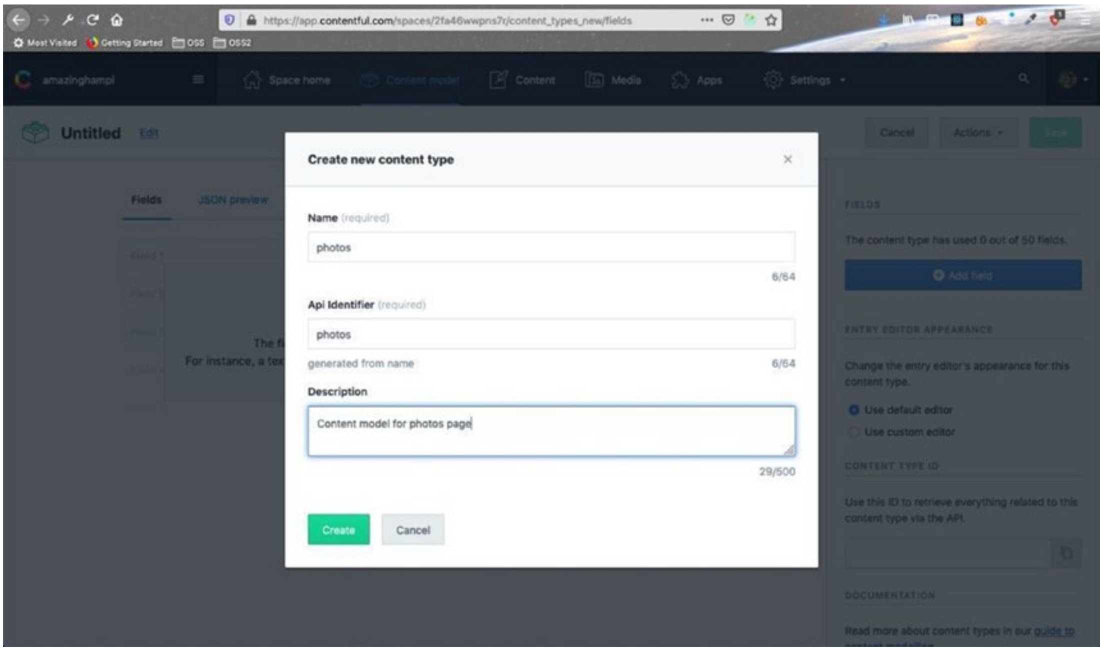


Figure 5-33. *Create a new content model*

I added four fields, all of them required. See Figure 5-34.

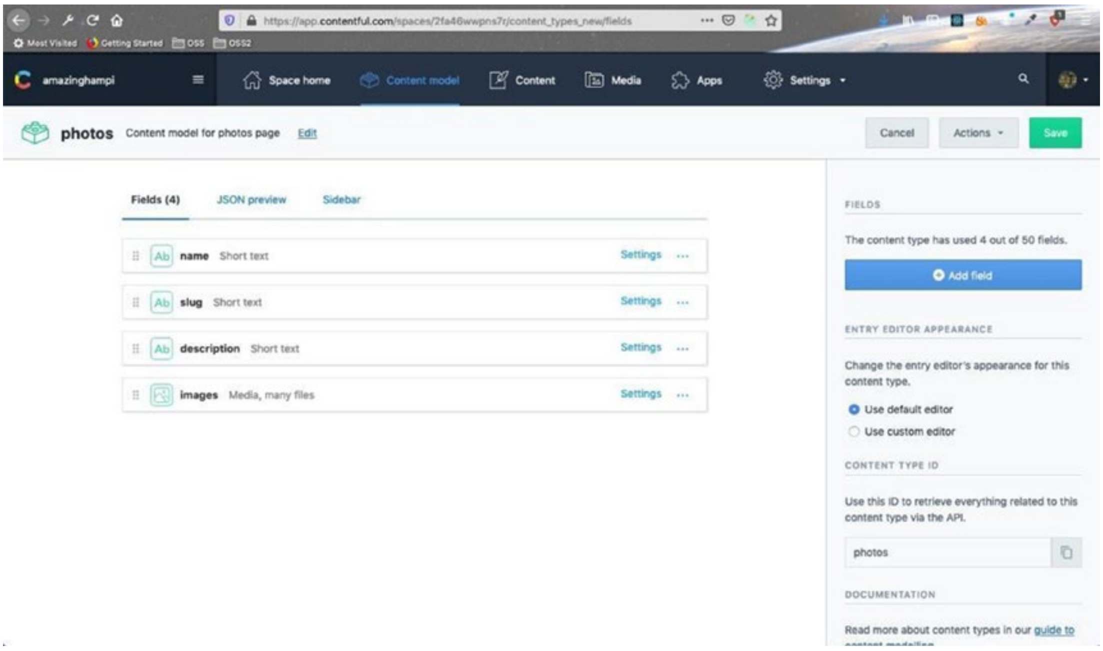
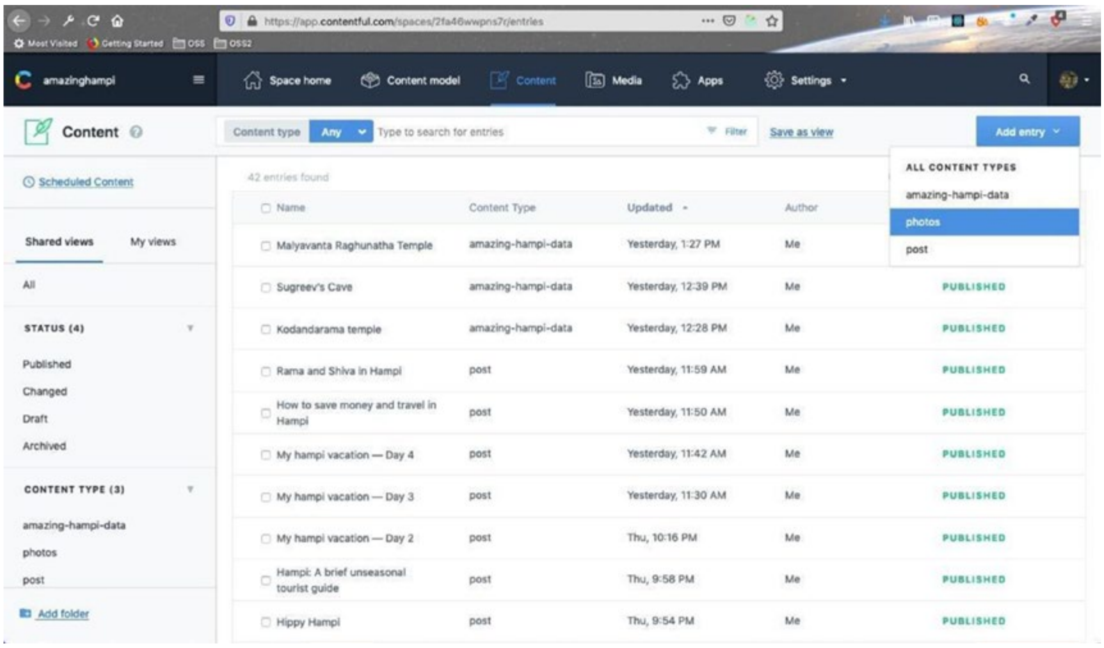


Figure 5-34. *Four required fields*

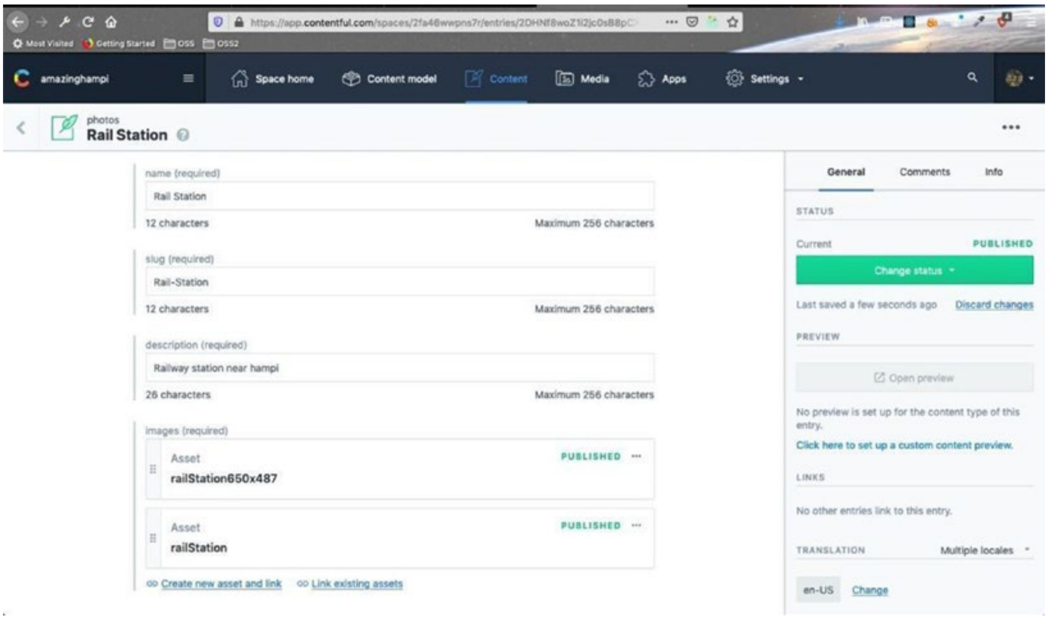
After saving by clicking the Save button, let's add some content from the Content tab, as shown in Figure 5-35.



Name	Content Type	Updated	Author	
Malyavanta Raghunatha Temple	amazing-hampi-data	Yesterday, 1:27 PM	Me	
Sugreev's Cave	amazing-hampi-data	Yesterday, 12:39 PM	Me	PUBLISHED
Kodandarama temple	amazing-hampi-data	Yesterday, 12:28 PM	Me	PUBLISHED
Rama and Shiva in Hampi	post	Yesterday, 11:59 AM	Me	PUBLISHED
How to save money and travel in Hampi	post	Yesterday, 11:50 AM	Me	PUBLISHED
My hampi vacation — Day 4	post	Yesterday, 11:42 AM	Me	PUBLISHED
My hampi vacation — Day 3	post	Yesterday, 11:30 AM	Me	PUBLISHED
My hampi vacation — Day 2	post	Thu, 10:16 PM	Me	PUBLISHED
Hampi: A brief unseasonal tourist guide	post	Thu, 9:58 PM	Me	PUBLISHED
Hippy Hampi	post	Thu, 9:54 PM	Me	PUBLISHED

Figure 5-35. *Adding content*

On the content page, I provide a name, slug, brief description, and two photos. They are the same image, but the first one is smaller and 650x487 pixels and will be displayed on the Photos page. The larger, original photo will be displayed once we go to the new page using the slug mechanism. The photo, which anyone can download for personal use, is shown in Figure 5-36.



name (required)
Rail Station
12 characters Maximum 256 characters

slug (required)
Rail-Station
12 characters Maximum 256 characters

description (required)
Railway station near hampi
26 characters Maximum 256 characters

images (required)
Asset PUBLISHED
railStation650x487
Asset PUBLISHED
railStation

STATUS
Current PUBLISHED
Last saved a few seconds ago Discard changes

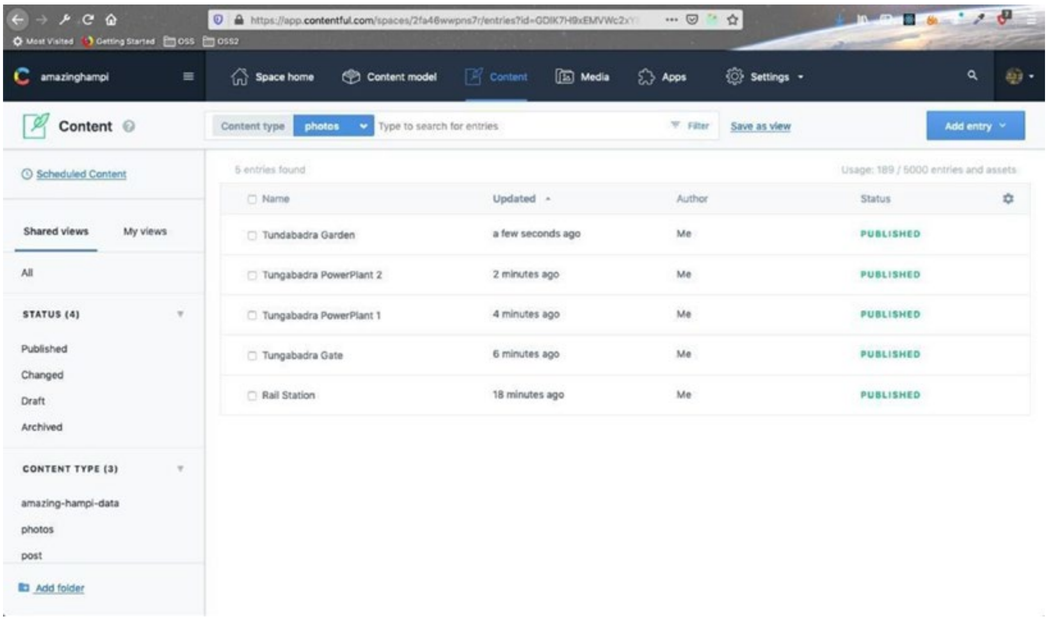
PREVIEW
No preview is set up for the content type of this entry.
Click here to set up a custom content preview.

LINKS
No other entries link to this entry.

TRANSLATION Multiple locales
en-US Change

Figure 5-36. *The content is added*

I added some content from my Hampi trip, as shown in Figure 5-37. I will add hundreds more items later.

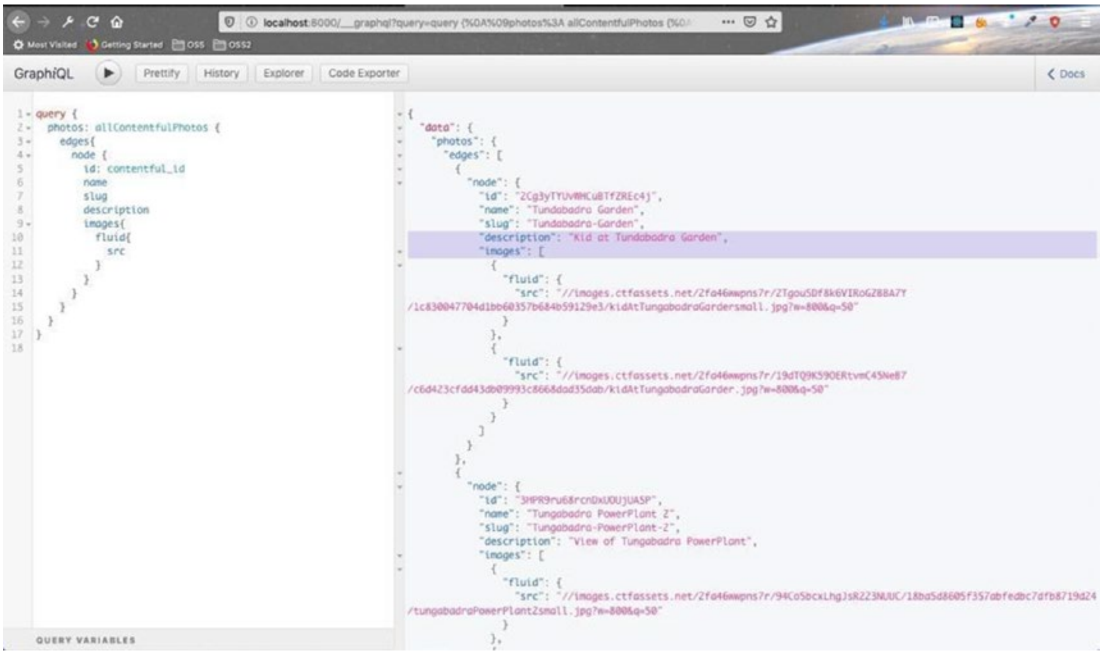


5 entries found Usage: 189 / 5000 entries and assets

Name	Updated	Author	Status
Tundabadra Garden	a few seconds ago	Me	PUBLISHED
Tungabadra PowerPlant 2	2 minutes ago	Me	PUBLISHED
Tungabadra PowerPlant 1	4 minutes ago	Me	PUBLISHED
Tungabadra Gate	6 minutes ago	Me	PUBLISHED
Rail Station	18 minutes ago	Me	PUBLISHED

Figure 5-37. *Content from my Hampi trip*

After any change in Contentful, you have to restart the DEV server by closing and starting `gatsby develop` from the terminal. Also, refresh your GraphiQL playground at GraphQL[8] to check the query. You can run the query shown in Figure 5-38 to get all the data back from Contentful.



```
query {
  photos: allContentfulPhotos {
    edges{
      node {
        id: contentful_id
        name
        slug
        description
        images{
          fluid{
            src
          }
        }
      }
    }
  }
}
```

```
{
  "data": {
    "photos": {
      "edges": [
        {
          "node": {
            "id": "2Cg3yTYUvWHCuBTfZREc4j",
            "name": "Tundabadra Garden",
            "slug": "Tundabadra-Garden",
            "description": "Kid at Tundabadra Garden",
            "images": [
              {
                "fluid": {
                  "src": "//images.ctfassets.net/2fa46wwpns7r/2TgouSDf8k6VIRoGZ8BA7Y
/1c830047704d1bb60357b684b59129e3/kidAtTungabadraGardersmall.jpg?w=800&q=50"
                }
              },
              {
                "fluid": {
                  "src": "//images.ctfassets.net/2fa46wwpns7r/19dTQ9K59OERtvmC45NeB7
/c6d423cfdd43db09993c8668dad35dab/kidAtTungabadraGarder.jpg?w=800&q=50"
                }
              }
            ]
          }
        },
        {
          "node": {
            "id": "3HPR9ru68rcnDxUOUjUASP",
            "name": "Tungabadra PowerPlant 2",
            "slug": "Tungabadra-PowerPlant-2",
            "description": "View of Tungabadra PowerPlant",
            "images": [
              {
                "fluid": {
                  "src": "//images.ctfassets.net/2fa46wwpns7r/94CoSbcxLhgJsR223NUUC/18ba5d8605f357abfedbc7dfb8719d24
/tungabadraPowerPlant2small.jpg?w=800&q=50"
                }
              },
```

Figure 5-38. *All the data from Contentful*

Creating the PhotoList Component

Let's add the query to `PhotoList.js` to get all the data from Contentful. We are using similar code we used with `BlogList.js`. It's shown in Listing 5-18.

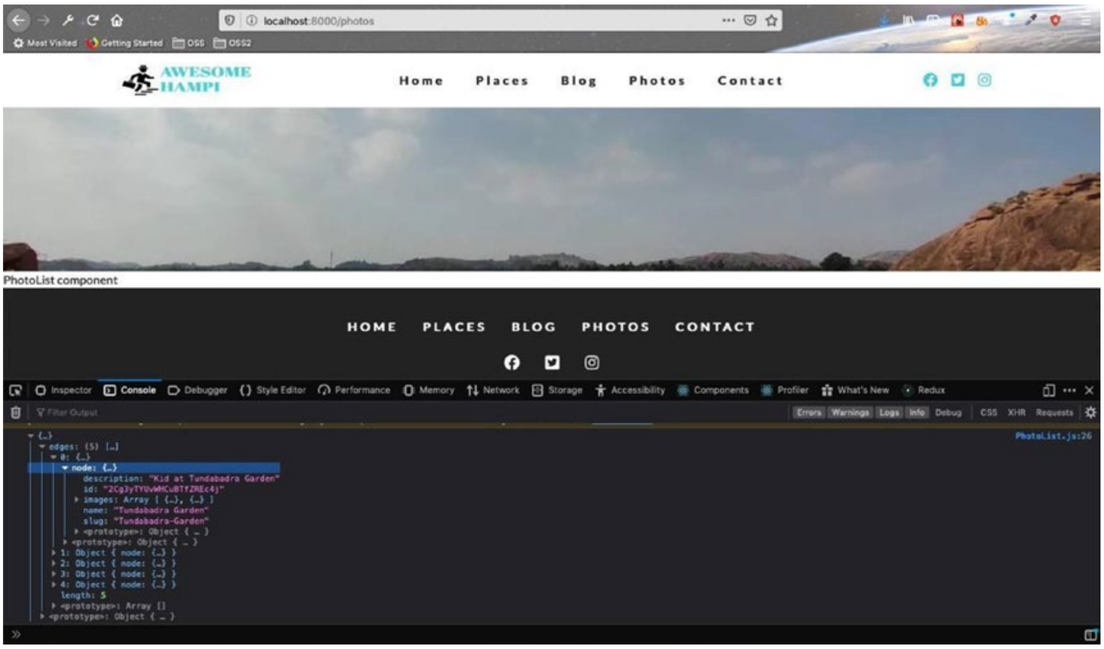
Listing 5-18. Updated PhotoList.js

```
import React from 'react'
import { useStaticQuery, graphql } from "gatsby"
```

[8] `http://localhost:8000/graphql`

```
const getPhotos = graphql`
query {
          photos: allContentfulPhotos {
    edges{
        node {
            id: contentful_id
            name
            slug
            description
            images{
                fluid{
                    src
                }
            }
        }
    }
}
}
`;

const PhotoList = () => {
    const { photos } = useStaticQuery(getPhotos);
    console.log(photos);
    return (
              <div>PhotoList component</div>
           )
}
export default PhotoList;
```

Let's check whether we are receiving the data by opening the console and going to the Photos page, as shown in Figure 5-39.



```
▾ {…}
  ▾ edges: (5) […]
    ▾ 0: {…}
      ▾ node: {…}
          description: "Kid at Tundabadra Garden"
          id: "2Cg3yTYUvWHCuBTfZREc4j"
        ▸ images: Array [ {…}, {…} ]
          name: "Tundabadra Garden"
          slug: "Tundabadra-Garden"
        ▸ <prototype>: Object { … }
      ▸ <prototype>: Object { … }
    ▸ 1: Object { node: {…} }
    ▸ 2: Object { node: {…} }
    ▸ 3: Object { node: {…} }
    ▸ 4: Object { node: {…} }
      length: 5
    ▸ <prototype>: Array []
  ▸ <prototype>: Object { … }
```

Figure 5-39. *The data is being received from Contentful*

Now that we are receiving the data, it's time to show it in the component. First, let's add some more `imports` at the top of the `PhotoList.js` file. Next, let's update inside the `return` statement to pass each node through a map to the PhotoCard component. The updated code is highlighted in Listing 5-19.

Listing 5-19. PhotoCard Component in PhotoList.js

```
import React from 'react'
import { useStaticQuery, graphql } from "gatsby"
import Title from "../Title"
import styles from "../../css/items.module.css"
import PhotoCard from './PhotoCard'

const getPhotos = graphql`
query {
          photos: allContentfulPhotos {
    edges{
        node {
            id: contentful_id
            name
            slug
            description
            images{
                fluid{
                    ...GatsbyContentfulFluid
                }
            }
        }
    }
}
}
`;

const PhotoList = () => {
    const { photos } = useStaticQuery(getPhotos);
    return (
        <section className={styles.tours}>
            <Title title="hampi" subtitle="photos" />
            <div className={styles.center}>
                {photos.edges.map(({ node }) => {
                    return <PhotoCard key={node.id} photo={node} />
                })}
            </div>
        </section>
    )
}
export default PhotoList;
```

Creating the PhotoCard Component

Create `PhotoCard.js` inside the photos folder. Add the content in Listing 5-20 to it.

Listing 5-20. PhotoCard.js

```
import React from 'react'
import Image from "gatsby-image"
import styles from "../../css/place.module.css"
import AniLink from "gatsby-plugin-transition-link/AniLink"

const PhotoCard = ({ photo }) => {
    const { name, slug, images } = photo;
    let mainImage = images[0].fluid;

    return (
        <article className={styles.place}>
            <div className={styles.imgContainer}>
                <Image fluid={mainImage} className={styles.img} alt="single
                photo" />
                <AniLink fade className={styles.link}
                to={`/photos/${slug}`}>open</AniLink>
            </div>
            <div className={styles.footer}>
                <h3>{name}</h3>
            </div>
        </article>
    )
}

export default PhotoCard
```

When we go to our Photos page in the browser, we get all the five photos, as shown in Figure 5-40.

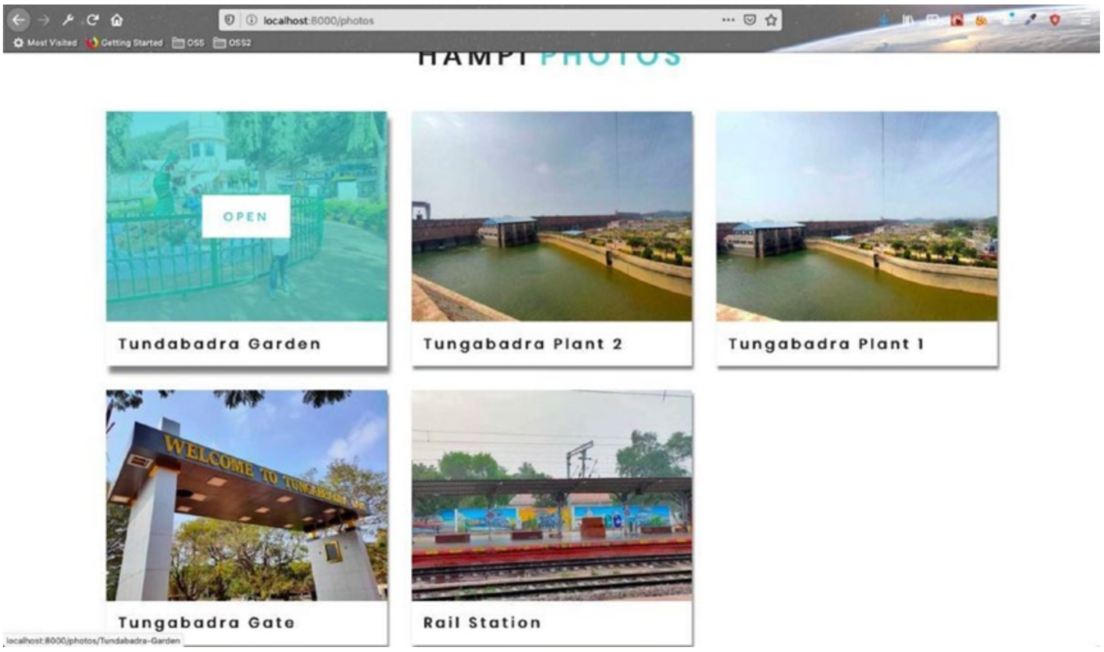


Figure 5-40. *All five photos are shown*

Creating the Photos Template

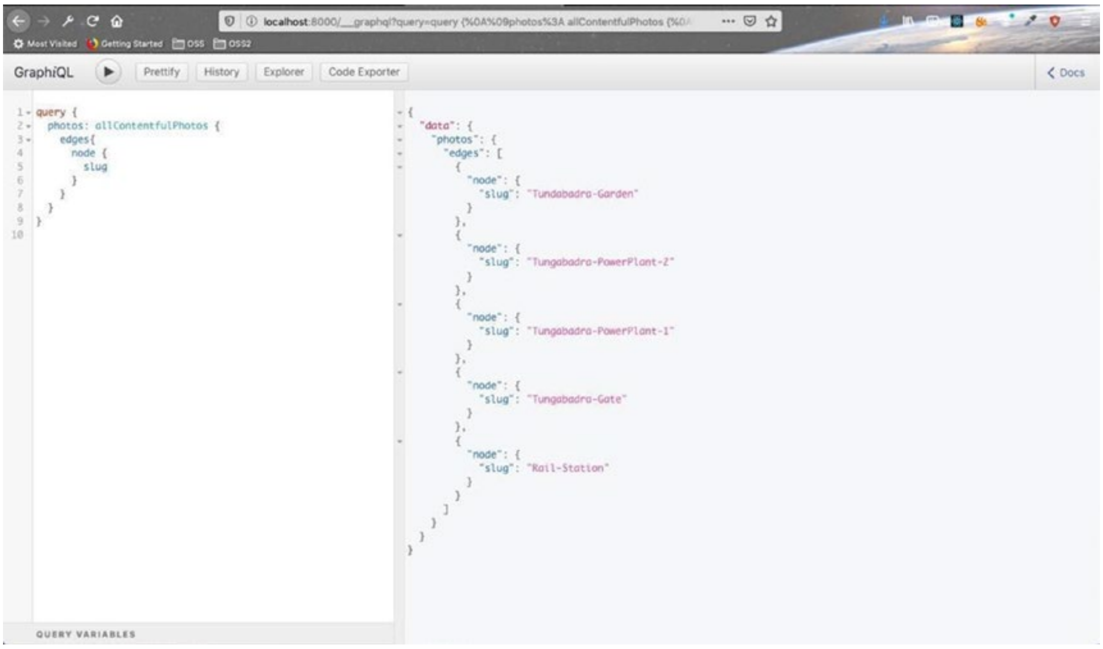
In this section, we will start adding code to display the larger image. It will appear when we click the Open button inside the photo on the Photos page.

To do this, we will follow the process we used earlier in this chapter. Create a file called `photos-template.js` inside the `templates` folder and add the contents of Listing 5-21. For now, use dummy data, which we are going to replace soon.

Listing 5-21. The photos-template.js File

```
import React from 'react'

const Photos = () => {
    return (
        <div>
            This is Photos template
        </div>
    )
}
export default Photos
```

Let's write our GraphQL query in the GraphiQL playground, as shown in Figure 5-41.



```
query {
  photos: allContentfulPhotos {
    edges{
      node {
        slug
      }
    }
  }
}
```

```
{
  "data": {
    "photos": {
      "edges": [
        {
          "node": {
            "slug": "Tundabadra-Garden"
          }
        },
        {
          "node": {
            "slug": "Tungabadra-PowerPlant-2"
          }
        },
        {
          "node": {
            "slug": "Tungabadra-PowerPlant-1"
          }
        },
        {
          "node": {
            "slug": "Tungabadra-Gate"
          }
        },
        {
          "node": {
            "slug": "Rail-Station"
          }
        }
      ]
    }
  }
}
```

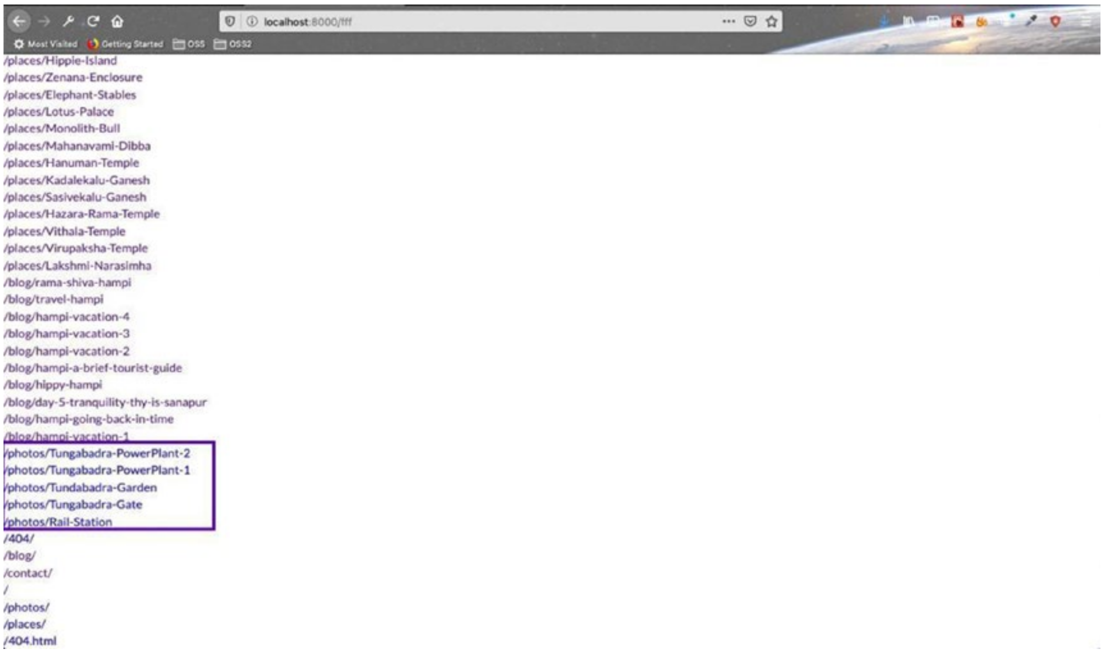
Figure 5-41. *The GraphiQL playground*

Next, open `gatsby-node.js` and add this new query. Next, we will loop through the node and create the page. The updated code is marked in bold in Listing 5-22.

Listing 5-22. Photos in gatsby-node.js

```
const path = require("path")

exports.createPages = async ({ graphql, actions }) => {
const { createPage } = actions

const { data } = await graphql(`
query {
    places: allContentfulAmazingHampiData {
        edges {
        node {
                slug
            }
        }
    }
    posts: allContentfulPost {
        edges {
            node {
                slug
            }
        }
    }
    photos: allContentfulPhotos {
        edges{
            node {
                slug
            }
        }
    }
}
`)

data.places.edges.forEach(({ node }) => {
    createPage({
            path: `places/${node.slug}`,
            component: path.resolve("./src/templates/place-template.js"),
            context: {
                slug: node.slug,
            },
        })
    })
    data.posts.edges.forEach(({ node }) => {
        createPage({
            path: `blog/${node.slug}`,
            component: path.resolve("./src/templates/blog-template.js"),
            context: {
                slug: node.slug,
            },
        })
    })
    data.photos.edges.forEach(({ node }) => {
        createPage({
            path: `photos/${node.slug}`,
            component: path.resolve("./src/templates/photos-template.js"),
            context: {
                slug: node.slug,
            },
        })
    })
}
```

To test whether everything is working, we need to stop and start `gatsby develop` in the terminal. Then move to any nonexistent page in the browser. You can see those pages being created, as shown in Figure 5-42.



```
/places/Hippie-Island
/places/Zenana-Enclosure
/places/Elephant-Stables
/places/Lotus-Palace
/places/Monolith-Bull
/places/Mahanavami-Dibba
/places/Hanuman-Temple
/places/Kadalekalu-Ganesh
/places/Sasivekalu-Ganesh
/places/Hazara-Rama-Temple
/places/Vithala-Temple
/places/Virupaksha-Temple
/places/Lakshmi-Narasimha
/blog/rama-shiva-hampi
/blog/travel-hampi
/blog/hampi-vacation-4
/blog/hampi-vacation-3
/blog/hampi-vacation-2
/blog/hampi-a-brief-tourist-guide
/blog/hippy-hampi
/blog/day-5-tranquility-thy-is-sanapur
/blog/hampi-going-back-in-time
/blog/hampi-vacation-1
/photos/Tungabadra-PowerPlant-2
/photos/Tungabadra-PowerPlant-1
/photos/Tundabadra-Garden
/photos/Tungabadra-Gate
/photos/Rail-Station
/404/
/blog/
/contact/
/
/photos/
/places/
/404.html
```

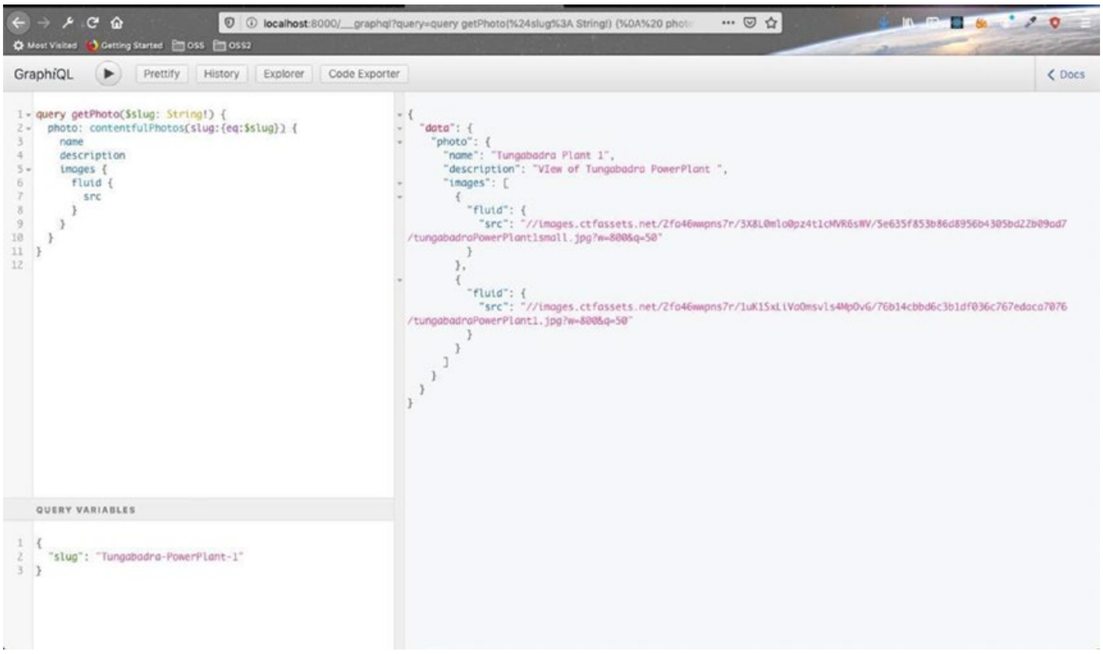
Figure 5-42. *New pages are being created*

Click any one of the pages and you will get the dummy text, as shown in Figure 5-43.

This is Photos template

Figure 5-43. *The dummy data*

Let's create the query to show the single blog photo in our playground, as shown in Figure 5-44.


localhost:8000/___graphql?query=query getPhoto(%24slug%3A String!) (%0A%20 phot
GraphiQL
Prettify
History
Explorer
Code Exporter
Docs

```
query getPhoto($slug: String!) {
  photo: contentfulPhotos(slug:{eq:$slug}) {
    name
    description
    images {
      fluid {
        src
      }
    }
  }
}
```

```
{
  "data": {
    "photo": {
      "name": "Tungabadra Plant 1",
      "description": "View of Tungabadra PowerPlant ",
      "images": [
        {
          "fluid": {
            "src": "//images.ctfassets.net/2fo46wwpns7r/3X8L0mlo0pz4tlcMVR6sWV/5e635f853b86d8956b4305bd22b09ad7
/tungabadraPowerPlant1small.jpg?w=800&q=50"
          }
        },
        {
          "fluid": {
            "src": "//images.ctfassets.net/2fo46wwpns7r/1uK15xLiVoOmsvls4MpOvG/76b14cbbd6c3b1df036c767edaca7076
/tungabadraPowerPlant1.jpg?w=800&q=50"
          }
        }
      ]
    }
  }
}
```

QUERY VARIABLES

```
{
  "slug": "Tungabadra-PowerPlant-1"
}
```

Figure 5-44. *The GraphiQL playground*

Next, let's add this query to the `photos-template.js` file. Don't forget to change `src` in `images` to `...GatsbyContentfulFluid`. Next, we will display the image along with the name and description. We need to import some components, as we did earlier, and we are also using the `css` from our Blog module. We are showing the image through the `img` from `gatsby-image`. The updated code is marked in bold in Listing 5-23.

Listing 5-23. Query in photos-template.js

```
import React from 'react'
import { graphql } from "gatsby"
import Layout from "../components/Layout"
import styles from "../css/single-blog.module.css"
import AniLink from "gatsby-plugin-transition-link/AniLink"
import Img from "gatsby-image"

const Photos = ({ data }) => {
    const { name, description, images } = data.photo;
    let mainImage = images[1].fluid;

    return (
        <Layout>
            <section className={styles.blog}>
                <h1 className={styles.center}>{name}</h1>
                <div className={styles.center}>
                    <Img fluid={mainImage} alt="single image" />
                    <h4>{description}</h4>
                    <AniLink fade to="/photos" className="btn-primary">
                    all photos</AniLink>
                </div>
            </section>
        </Layout>
    )
}

export const query = graphql`
query getPhoto($slug: String!) {
    photo: contentfulPhotos(slug:{eq:$slug}) {
        name
        description
        images {
            fluid {
                ...GatsbyContentfulFluid
            }
        }
    }}
`;

export default Photos
```

When we click any photo on the Photos page, we get the amazing full photo, as shown in Figure 5-45. Feel free to use these photos in any of your personal work.

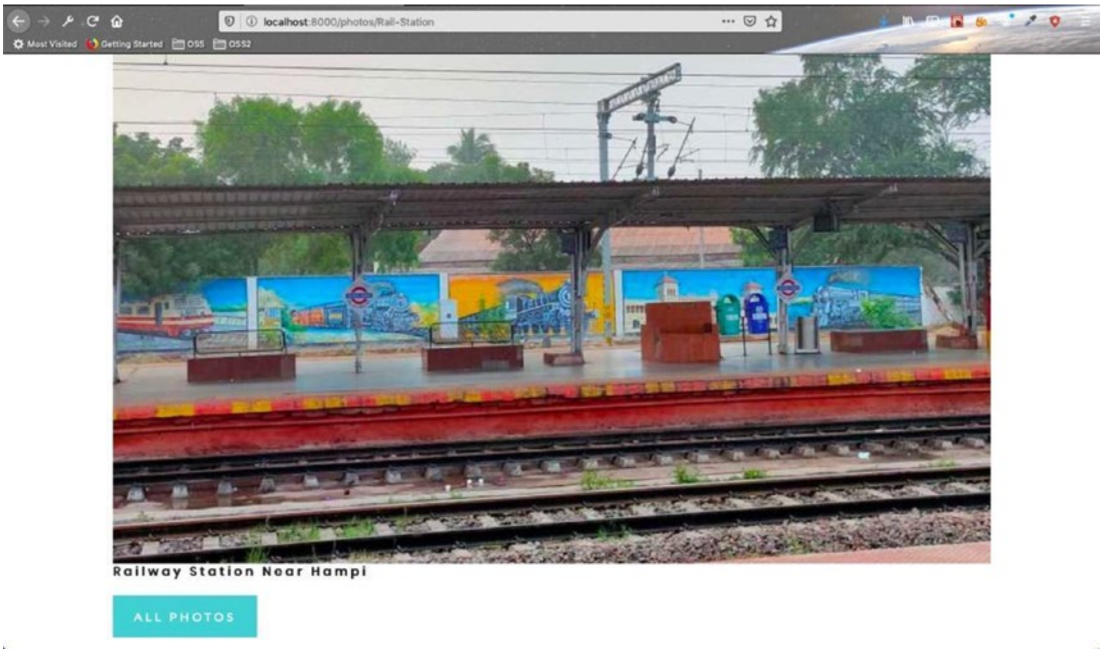

localhost:8000/photos/Rail-Station
Most Visited
Getting Started
OSS
OSS2
Railway Station Near Hampi
ALL PHOTOS

Figure 5-45. *The Hampi rail station*

You can find the code here[9].

Summary

This completes Chapter 5 and the third part of creating the tourism site using Contentful. We covered the following topics in this chapter:

- Creating the Blog component, which shows data stored in the Contentful CMS
- Creating the Photos component, which also shows data stored in the Contentful CMS

In the next chapter, we will complete the tourism site using Contentful. We will add Gatsby plugins and advertisements to the site.

[9]https://github.com/nabendu82/gatsbyTourism

CHAPTER 6

Creating a Tourism Site with Contentful: Part Four

Welcome to Chapter 6. In this chapter, we will add Gatsby plugins and advertisements to our site. Doing so will add functionality and advertising to our site, and this process is more complex than when adding these options to a normal HTML, CSS, or JS site.

Adding Gatsby Plugins

To complete this section, you need to buy a domain name and add the details to Netlify. This process is described in Chapter 2. After you're done with the setup, go to `https://amazinghampi.com/`[1] to access the project, as shown in Figure 6-1.

[1] `https://amazinghampi.com/`

© Nabendu Biswas 2021
N. Biswas, *Foundation Gatsby Projects*, https://doi.org/10.1007/978-1-4842-6558-1_6

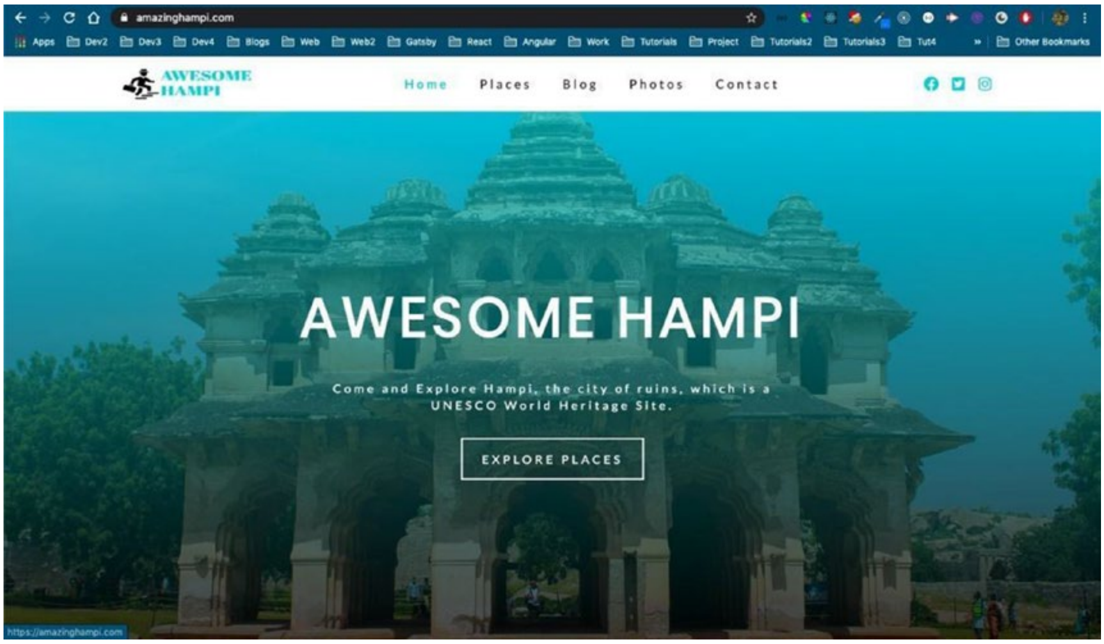


Figure 6-1. *The Hampi site, finally live*

The SEO Plugin

In this section, you'll add plugins for SEO purposes. You need to first install the `gatsby-plugin-react-helmet` plugin, which will help you control the head element of each page.

As per the documentation,[2] `npm install` the package first. Head back to the project and stop any `gatsby develop` that's running. After that, `npm install` the packages using the following command.

```
npm install --save gatsby-plugin-react-helmet react-helmet
```

As per the documentation, we need to add the code highlighted in Listing 6-1 to the `gatsby-config.js` file.

Listing 6-1. Add react-helmet to the gatsby-config.js File

```
require("dotenv").config({
  path: `.env.${process.env.NODE_ENV}`,
})
```

[2] https://www.gatsbyjs.org/packages/gatsby-plugin-react-helmet/

```
  plugins: [
         ...
...
    `gatsby-plugin-sitemap`,
    `gatsby-plugin-styled-components`,
    `gatsby-transformer-sharp`,
    `gatsby-plugin-sharp`,
    `gatsby-plugin-transition-link`,
    `gatsby-transformer-remark`,
    `gatsby-plugin-react-helmet`
  ]
}
```

We also need to add site metadata to the `gatsby-config.js` file, as it will be used by the next plugins. One thing to notice is the `image` tag. The image mentioned in it needs to be placed in a static folder. The updated code is marked in bold in Listing 6-2.

Listing 6-2. Add the Site's Metadata to the gatsby-config.js File

```
require("dotenv").config({
  path: `.env.${process.env.NODE_ENV}`,
})

module.exports = {
  siteMetadata: {
    title: "AmazingHampi",
    description: "Tips, information, blogs and photos on Hampi, the city of
    ruins, is a UNESCO World Heritage Site.",
    author: "Nabendu Biswas",
    twitterUsername: "@nabendu82",
    image:'/VirupakshaTemple20.jpg',
    siteUrl:'https://amazinghampi.com'
  },
  plugins: [
          ...
          ...
    `gatsby-plugin-sitemap`,
    `gatsby-plugin-styled-components`,
    `gatsby-transformer-sharp`,
    `gatsby-plugin-sharp`,
    `gatsby-plugin-transition-link`,
    `gatsby-transformer-remark`,
    `gatsby-plugin-react-helmet`
  ]
}
```

After this, start your `gatsby develop` and head over to the code editor. Add an `SEO.js` file inside the `components` folder. The initial contents of the file are shown in Listing 6-3. It accepts two props—`title` and `description`—from any component.

Listing 6-3. The SEO.js File

```
import React from "react"
import { Helmet } from "react-helmet"
import { useStaticQuery, graphql } from "gatsby"

const SEO = ({ title, description }) => {
    return (
        <Helmet htmlAttributes={{ lang: "en" }} title={title}>
            <meta name="description" content={description} />
        </Helmet>
        )
}

export default SEO
```

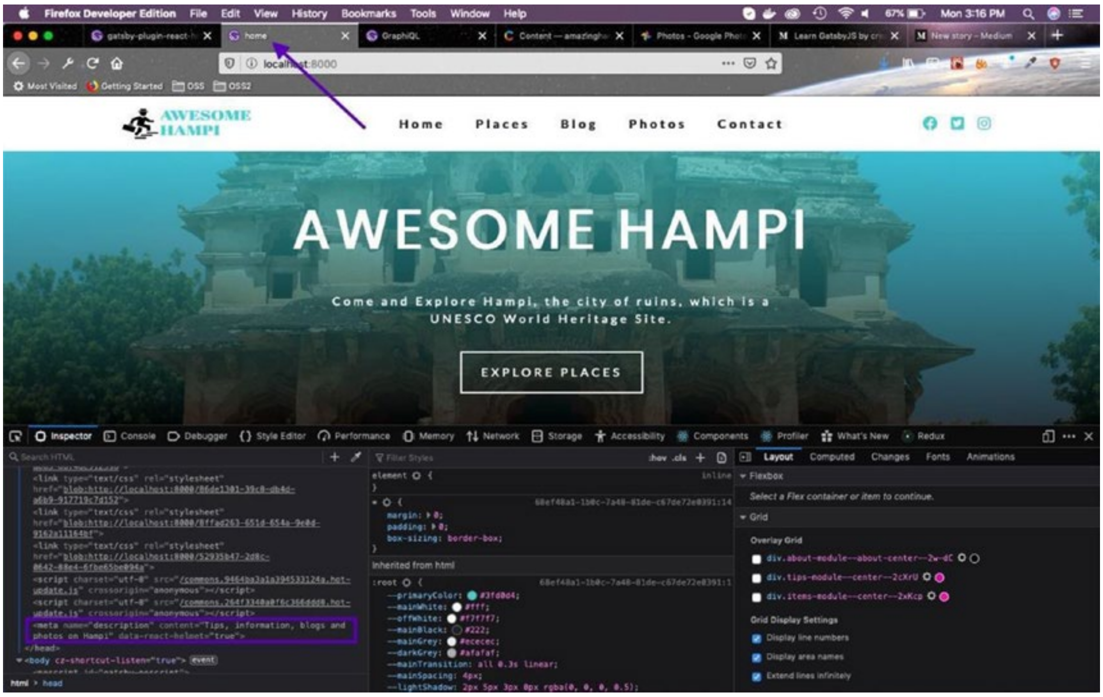
Next, head over the home page `index.js` file and import the SEO component. Use it to pass the title and description. The updated code is marked in bold in Listing 6-4.

Listing 6-4. The SEO Component in index.js

```
...
...
import FeaturedPlaces from "../components/Home/FeaturedPlaces"
import SEO from "../components/SEO"
```

```
...
...
export default ({ data }) => (
    <Layout>
        <SEO title="Home" description="Tips, information, blogs and photos
        on Hampi" />
        <StyledHero home="true" img={data.defaultBcg.childImageSharp.fluid}>
            ...
            ...
        </StyledHero>
        <About />
        <Tips />
        <FeaturedPlaces />
    </Layout>
)
```

When we head over to the localhost, we can see the `title` and `meta` tags inside the head tag, as shown in Figure 6-2.



```
<link type="text/css" rel="stylesheet"
href="blob:http://localhost:8000/86de1301-39c0-db4d-
a6b9-917719c7d152">
<link type="text/css" rel="stylesheet"
href="blob:http://localhost:8000/8ffad263-651d-654a-9e0d-
9162a11164bf">
<link type="text/css" rel="stylesheet"
href="blob:http://localhost:8000/52935b47-2d8c-
0642-88e4-6fbe65be094a">
<script charset="utf-8" src="/commons.9464ba3a1a3945331249.hot-
update.js" crossorigin="anonymous"></script>
<script charset="utf-8" src="/commons.264f3340a0f6c366ddd8.hot-
update.js" crossorigin="anonymous"></script>
<meta name="description" content="Tips, information, blogs and
photos on Hampi" data-react-helmet="true">
</head>
<body cz-shortcut-listen="true"> event
```

```
element {                                     inline
}
* {          68ef48a1-1b0c-7a48-81de-c67de72e0391:14
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}
Inherited from html
:root {       68ef48a1-1b0c-7a48-81de-c67de72e0391:1
  --primaryColor: #3fd0d4;
  --mainWhite: #fff;
  --offWhite: #f7f7f7;
  --mainBlack: #222;
  --mainGrey: #ececec;
  --darkGrey: #afafaf;
  --mainTransition: all 0.3s linear;
  --mainSpacing: 4px;
  --lightShadow: 2px 5px 3px 0px rgba(0, 0, 0, 0.5);
```

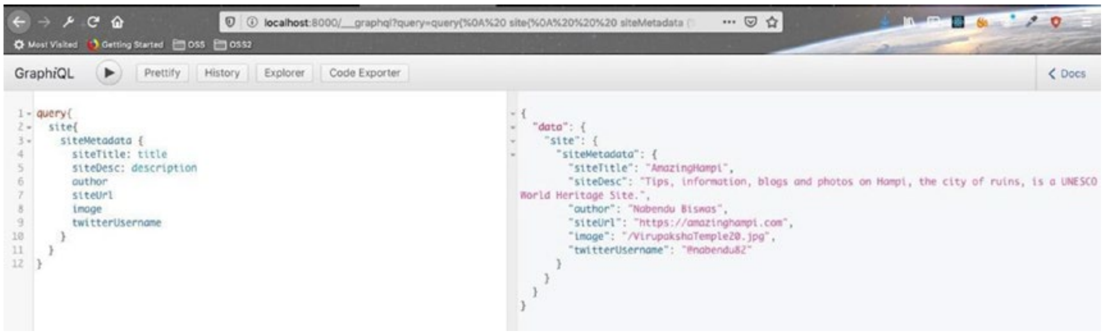
Select a Flex container or item to continue.

- div.about-module--about-center--2w-dC
- div.tips-module--center--2cXrU
- div.items-module--center--2xKcp

- [x] Display line numbers
- [x] Display area names
- [x] Extend lines infinitely

Figure 6-2. *The title and meta tags*

To be able to introduce GraphQL into our `SEO.js` file, we must create the query in the GraphiQL playground, as shown in Figure 6-3.



```
query{
  site{
    siteMetadata {
      siteTitle: title
      siteDesc: description
      author
      siteUrl
      image
      twitterUsername
    }
  }
}
```

```
{
  "data": {
    "site": {
      "siteMetadata": {
        "siteTitle": "AmazingHampi",
        "siteDesc": "Tips, information, blogs and photos on Hampi, the city of ruins, is a UNESCO World Heritage Site.",
        "author": "Nabendu Biswas",
        "siteUrl": "https://amazinghampi.com",
        "image": "/VirupakshaTemple20.jpg",
        "twitterUsername": "@nabendu82"
      }
    }
  }
}
```

Figure 6-3. *The query in the playground*

Let's add the query to the `SEO.js` file and change the title to also include the `siteTitle` that we are getting from our query. We also need to add a new `meta` for `images`. The updated code is marked in bold in Listing 6-5.

Listing 6-5. Updated SEO.js File

```
import React from "react"
import { Helmet } from "react-helmet"
import { useStaticQuery, graphql } from "gatsby"

const getData = graphql`
query{
    site{
        siteMetadata {
            siteTitle: title
            siteDesc: description
            author
            siteUrl
            image
            twitterUsername
        }
    }
}
`;
```

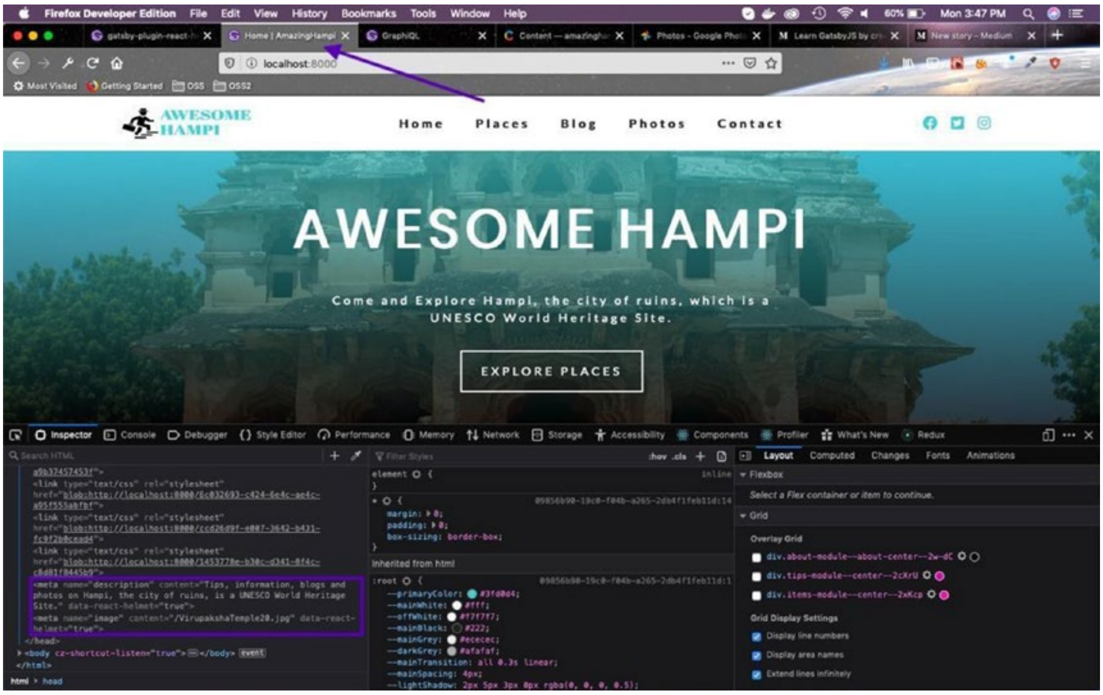
```
const SEO = ({ title, description }) => {
    const { site } = useStaticQuery(getData);
    const { siteTitle, siteDesc, siteUrl, image, twitterUsername } = site.
    siteMetadata;
    return (
        <Helmet htmlAttributes={{ lang: "en" }} title={`${title} |
        ${siteTitle}`}>
            <meta name="description" content={description || siteDesc} />
            <meta name="image" content={image} />
        </Helmet>
        )
}

export default SEO
```

In the updated `SEO.js` file, we made `description` props optional, so if we don't pass anything, we will be using `siteDesc`, which we defined in `gatsby-config.js`. Let's delete the description from `index.js`, as we will use the longer and more accurate `siteDesc`. The update is shown in Listing 6-6.

Listing 6-6. Delete Description from index.js

```
...

export default ({ data }) => (
    <Layout>
        <SEO title="Home" />
        <StyledHero home="true" img={data.defaultBcg.childImageSharp.fluid}>
            ...
                  ...
        </StyledHero>
        <About />
        <Tips />
        <FeaturedPlaces />
    </Layout>
)
```

In our localhost, we can now see these changes (see Figure 6-4).



```
<link type="text/css" rel="stylesheet"
href="blob:http://localhost:8000/6c032693-c424-6e4c-ae4c-a95f555abfbf">
<link type="text/css" rel="stylesheet"
href="blob:http://localhost:8000/ccd26d9f-e007-3642-b431-fc9f2b0cead4">
<link type="text/css" rel="stylesheet"
href="blob:http://localhost:8000/1453778e-b30c-d341-8f4c-c8d81f0445b9">
<meta name="description" content="Tips, information, blogs and
photos on Hampi, the city of ruins, is a UNESCO World Heritage
Site." data-react-helmet="true">
<meta name="image" content="/VirupakshaTemple20.jpg" data-react-
helmet="true">
</head>
<body cz-shortcut-listen="true">…</body>
</html>
```

```
element {
}
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}
Inherited from html
:root {
  --primaryColor: #3fd0d4;
  --mainWhite: #fff;
  --offWhite: #f7f7f7;
  --mainBlack: #222;
  --mainGrey: #ececec;
  --darkGrey: #afafaf;
  --mainTransition: all 0.3s linear;
  --mainSpacing: 4px;
  --lightShadow: 2px 5px 3px 0px rgba(0, 0, 0, 0.5);
```

Figure 6-4. *The title and meta changes*

Next, we will add a Twitter Card to our project. When we share our project on Twitter, the Twitter Card will show a nice image with the description. The updated code is marked in bold in Listing 6-7.

Listing 6-7. Twitter Card in SEO.js

```
...

const SEO = ({ title, description }) => {
    const { site } = useStaticQuery(getData);
    const { siteTitle, siteDesc, siteUrl, image, twitterUsername } = site.
    siteMetadata;
    return (
        <Helmet htmlAttributes={{ lang: "en" }} title={`${title} |
        ${siteTitle}`}>
            <meta name="description" content={description || siteDesc} />
            <meta name="image" content={image} />
            <meta name="twitter:card" content="summary_large_image" />
            <meta name="twitter:creator" content={twitterUsername} />
            <meta name="twitter:title" content={siteTitle} />
            <meta name="twitter:description" content={siteDesc} />
            <meta name="twitter:image" content={`${siteUrl}${image}`} />
        </Helmet>
        )
}

export default SEO
```

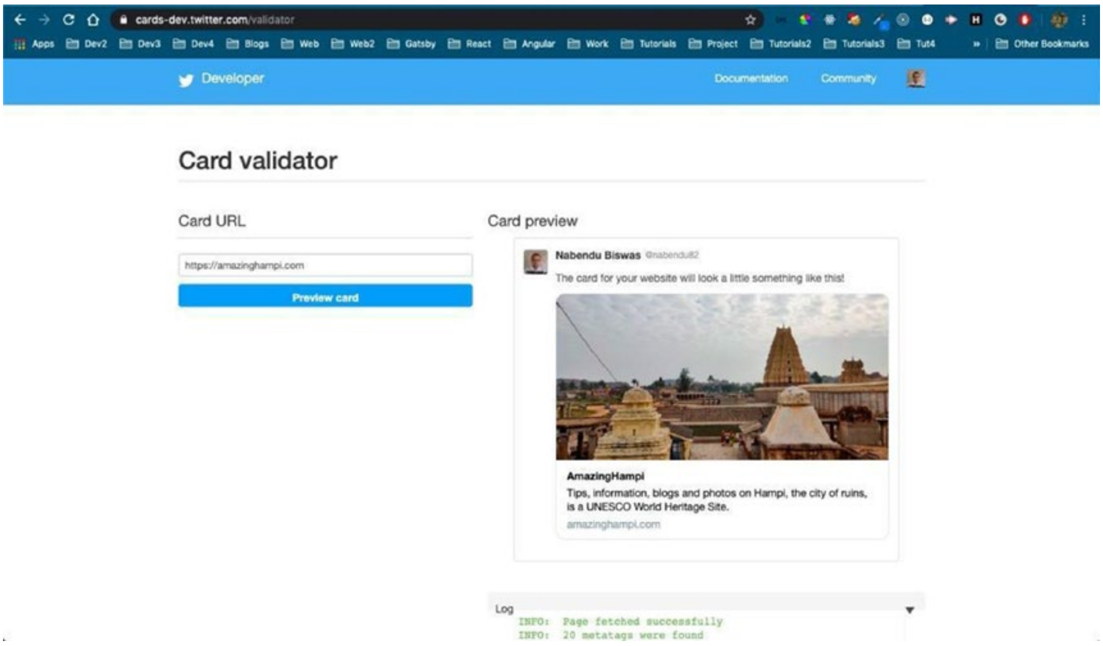
Next, we will add a Facebook Card to our project. When we share our project on Facebook, the Facebook Card will show a nice image with the description. The updated code is marked in bold in Listing 6-8.

Listing 6-8. Facebook Card in SEO.js

```
...

const SEO = ({ title, description }) => {
    const { site } = useStaticQuery(getData);
    const { siteTitle, siteDesc, siteUrl, image, twitterUsername } = site.
    siteMetadata;
    return (
        <Helmet htmlAttributes={{ lang: "en" }} title={`${title} |
        ${siteTitle}`}>
            <meta name="description" content={description || siteDesc} />
            <meta name="image" content={image} />
            <meta name="twitter:card" content="summary_large_image" />
            <meta name="twitter:creator" content={twitterUsername} />
            <meta name="twitter:title" content={siteTitle} />
            <meta name="twitter:description" content={siteDesc} />
            <meta name="twitter:image" content={`${siteUrl}${image}`} />
            <meta property="og:url" content={siteUrl} />
            <meta property="og:type" content="website" />
            <meta property="og:title" content={siteTitle} />
            <meta property="og:description" content={siteDesc} />
            <meta property="og:image" content={`${siteUrl}${image}`} />
            <meta property="og:image:width" content="400" />
            <meta property="og:image:height" content="300" />
        </Helmet>
        )
}

export default SEO
```

To test the cards, we need to push the code to GitHub so that it is deployed in Netlify.

Next, let's test this on the Twitter Card Validator and the Facebook Card Validator, to check how it will look on both social networks. The results are shown in Figures 6-5 and 6-6.



Card validator

Card URL

https://amazinghampi.com

Card preview

Nabendu Biswas @nabendu82

The card for your website will look a little something like this!

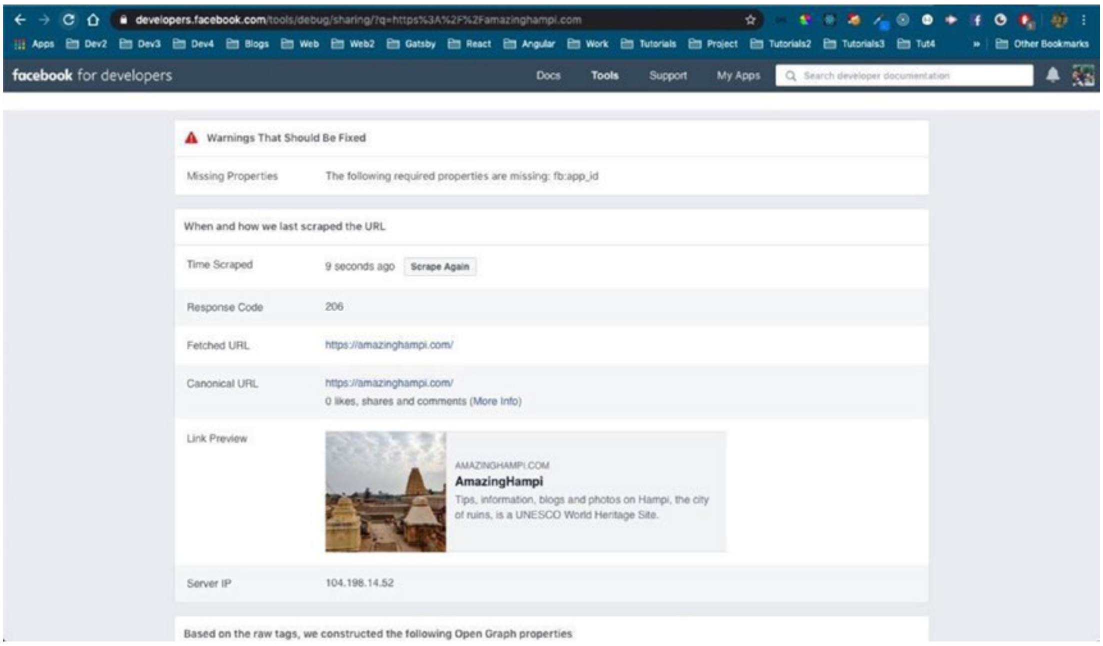
AmazingHampi

Tips, information, blogs and photos on Hampi, the city of ruins, is a UNESCO World Heritage Site.

amazinghampi.com

```
Log
INFO:  Page fetched successfully
INFO:  20 metatags were found
```

Figure 6-5. *Twitter Card validator*


developers.facebook.com/tools/debug/sharing/?q=https%3A%2F%2Famazinghampi.com
Apps Dev2 Dev3 Dev4 Blogs Web Web2 Gatsby React Angular Work Tutorials Project Tutorials2 Tutorials3 Tut4 Other Bookmarks
facebook for developers
Docs Tools Support My Apps
Search developer documentation

Warnings That Should Be Fixed

Missing Properties	The following required properties are missing: fb:app_id

When and how we last scraped the URL

Time Scraped	9 seconds ago Scrape Again
Response Code	206
Fetched URL	https://amazinghampi.com/
Canonical URL	https://amazinghampi.com/ 0 likes, shares and comments (More Info)
Link Preview	AMAZINGHAMPI.COM **AmazingHampi** Tips, information, blogs and photos on Hampi, the city of ruins, is a UNESCO World Heritage Site.
Server IP	104.198.14.52

Based on the raw tags, we constructed the following Open Graph properties

Figure 6-6. *Facebook Card validator*

It looks great on both networks. Let's now add SEO to all the pages in our project.

First, we will add an SEO component to `404.js` and `contact.js`. These components will also have default descriptions like `index.js`, so we are not passing the `description` prop. The updated code is marked in bold in Listings 6-9 and 6-10.

Listing 6-9. The SEO Component in 404.js

```
...
...
import Banner from "../components/Banner"
import SEO from "../components/SEO"

export default function error() {
return (
    <Layout>
        <SEO title="Error" />
        <header className={styles.error}>
                              ...
        </header>
    </Layout>
)
}
```

Listing 6-10. The SEO Component in contact.js

```
...
import Contact from '../components/Contact/Contact'
import SEO from "../components/SEO"

...

export default function contact({ data }) {
    return (
        <Layout>
            <SEO title="Contact" />
            <StyledHero img={data.connectBcg.childImageSharp.fluid}>
            </StyledHero>
            <Contact />
        </Layout>
    )
}
```

The next three pages (`blog.js`, `photos.js`, and `places.js`) will have different descriptions. We will be passing the `description` props. The updated code is marked in bold in Listings 6-11 through 6-13.

Listing 6-11. The SEO Component in blog.js

```
...
import BlogList from '../components/Blog/BlogList'
import SEO from "../components/SEO"

...
export default function blog({ data }) {
    return (
        <Layout>
            <SEO title="Blog" description="Real experiences blogs on Hampi,
            the city of ruins, is a UNESCO World Heritage Site." />
            <StyledHero img={data.blogBcg.childImageSharp.fluid} />
            <BlogList />
        </Layout>
    )
}
```

Listing 6-12. The SEO Component in photos.js

```
...
import PhotoList from '../components/Photos/PhotoList'
import SEO from "../components/SEO"

...

export default function photos({ data }) {
    return (
        <Layout>
            <SEO title="Photos" description="Royalty free image of Hampi,
            the city of ruins, is a UNESCO World Heritage Site." />
            <StyledHero img={data.blogBcg.childImageSharp.fluid} />
            <PhotoList />
        </Layout>
    )
}
```

Listing 6-13. The SEO Component in places.js

```
...
import Places from '../components/Places/Places'
import SEO from "../components/SEO"

...

export default function places({ data }) {
    return (
        <Layout>
            <SEO title="Places" description="Places to visit in Hampi, the
            city of ruins, is a UNESCO World Heritage Site." />
            <StyledHero img={data.defaultBcg.childImageSharp.fluid}>
            </StyledHero>
            <Places />
        </Layout>
    )
}
```

*

We have three templates that auto-generate pages for us. We will be using the page name as the title in all three. The updated code is marked in bold in Listings 6-14 through 6-16.

Listing 6-14. The SEO Component in blog-template.js

```
...
import StyledHero from "../components/StyledHero"
import SEO from "../components/SEO"

const Blog = ({ data }) => {
    const { title, published, author, description: {childMarkdownRemark},
    image} = data.post;
    return <Layout>
                <SEO title={title} />
                <h1 className={styles.center}>{title}</h1>
                <StyledHero img={image.fluid} />
                <section className={styles.blog}>
                    <div className={styles.center}>
                        ...
                    </div>
                </section>
        </Layout>
}

...

export default Blog
```

Listing 6-15. The SEO Component in place-template.js

```
...
import AniLink from "gatsby-plugin-transition-link/AniLink"
import SEO from "../components/SEO"

const Template = ({ data }) => {
    const { name, timeRequired, timings, entryFees, description: {
    description }, images } = data.place;
    const [mainImage, ...placeImages] = images

    return (
        <Layout>
            <SEO title={name} />
            <StyledHero img={mainImage.fluid} />
            <section className={styles.template}>
                ...
            </section>
        </Layout>
    )
}

...
export default Template
```

Listing 6-16. The SEO Component in photos-template.js

```
...
import Img from "gatsby-image"
import SEO from "../components/SEO"

const Photos = ({ data }) => {
    const { name, description, images } = data.photo;
    let mainImage = images[1].fluid;

    return (
        <Layout>
            <SEO title={name} description={`Royalty free image of ${name}`}/>
            <section className={styles.blog}>
```

```
                <h1 className={styles.center}>{name}</h1>
                ...
            </section>
        </Layout>
    )
}

...

export default Photos
```

Other Plugins

We will add two more plugins for SEO—`gatsby-plugin-robots-txt` and `gatsby-plugin-sitemap`. The details for installing both can be found in Chapter 2.

After the required npm installs, add the code in Listing 6-17 to the `gatsby-config.js` file. The updated code is marked in bold.

Listing 6-17. Plugins in gatsby-config.js

```
  plugins: [
    {
      resolve: `gatsby-source-filesystem`,
      options: {
        name: `images`,
        path: `${__dirname}/src/images/`,
      },
    },
    {
      resolve: `gatsby-source-contentful`,
      options: {
        spaceId: process.env.CONTENTFUL_SPACE_ID,
        accessToken: process.env.CONTENTFUL_ACCESS_TOKEN,
      },
    },
    {
      resolve: 'gatsby-plugin-robots-txt',
      options: {
        host: 'https://amazinghampi.com',
        sitemap: 'https://amazinghampi.com/sitemap.xml',
        policy: [{ userAgent: '*', allow: '/' }]
      }
    },
    `gatsby-plugin-sitemap`,
    `gatsby-plugin-styled-components`,
    `gatsby-transformer-sharp`,
    `gatsby-plugin-sharp`,
    `gatsby-plugin-transition-link`,
    `gatsby-transformer-remark`,
    `gatsby-plugin-react-helmet`
 ]
```

I will also add Google Analytics to the site, so that I can keep count and do other analyses on my visitors. The installation process is similar to what we did in Chapter 2.

After following the required steps on the Google Analytics page and `npm` installing the plugin, the changes shown in Listing 6-18 need to be made to the `gatsby-config.js` file. The updated code is marked in bold.

Listing 6-18. Google Analytics in gatsby-config.js

```
  plugins: [
            ...
    {
      resolve: 'gatsby-plugin-robots-txt',
      options: {
        host: 'https://amazinghampi.com',
        sitemap: 'https://amazinghampi.com/sitemap.xml',
        policy: [{ userAgent: '*', allow: '/' }]
      }
    },
    {
      resolve: `gatsby-plugin-google-analytics`,
      options: {
        trackingId: "UA-XXXXXXXXX-2",
      }
    },
    `gatsby-plugin-sitemap`,
              ...
]
```

After pushing and successfully deploying to Netlify, the site is live at https://amazinghampi.com/[3] and the project is complete! See Figure 6-7.

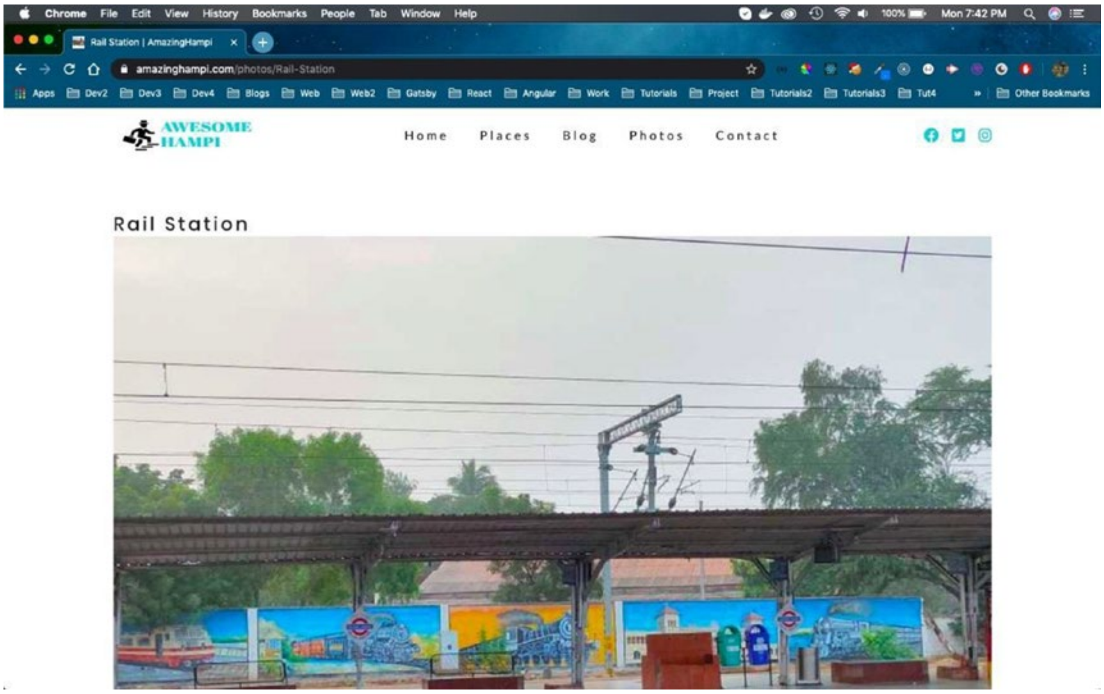


Figure 6-7. *The live amazinghampi.com site*

Feel free to use this code to create amazing sites by cloning or forking from my GitHub at https://github.com/nabendu82/gatsbyTourism.[4]

[3]https://amazinghampi.com/

[4]https://github.com/nabendu82/gatsbyTourism

Adding Advertisements to the Site

I decided to include advertisements on the site built in this series. I had just learned how to include ads on a Gatsby-powered site and added this information to my blog site at `https://nabendu.blog/`[5].

Using Media.net Ads

I have approval to use Media.net ads. You can learn more about that process in Chapter 2. After you log in to the Media.net console, you need to click the Add New button, as shown in Figure 6-8.

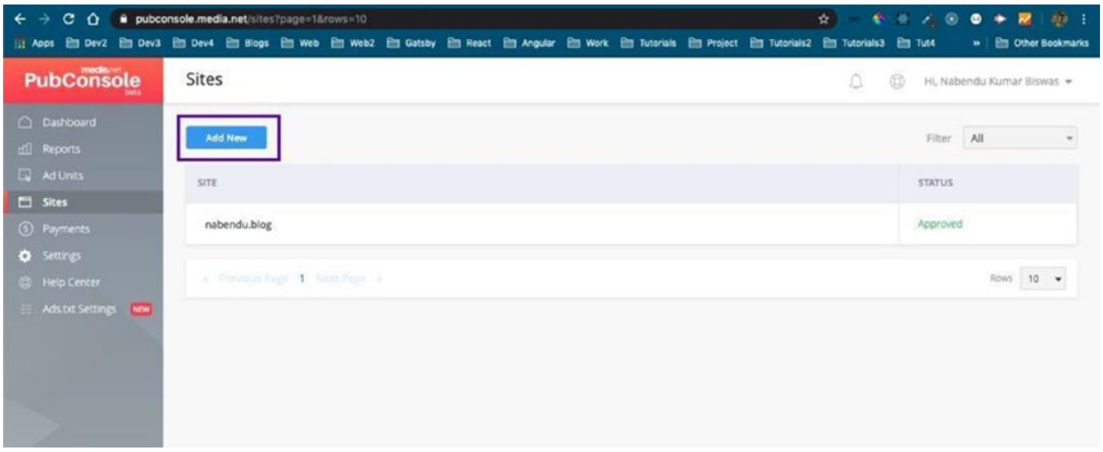


Figure 6-8. *The Media.net site*

It will show the popup in Figure 6-9, where you have to enter the domain name, average daily traffic, and current ad revenue.

[5]`https://nabendu.blog/`

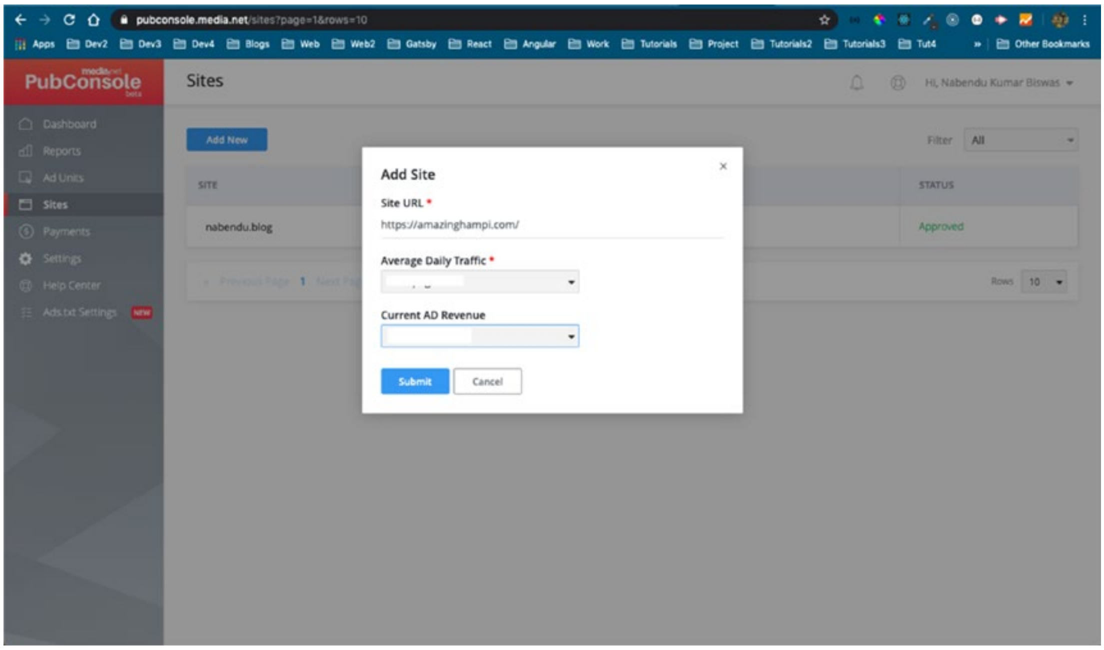
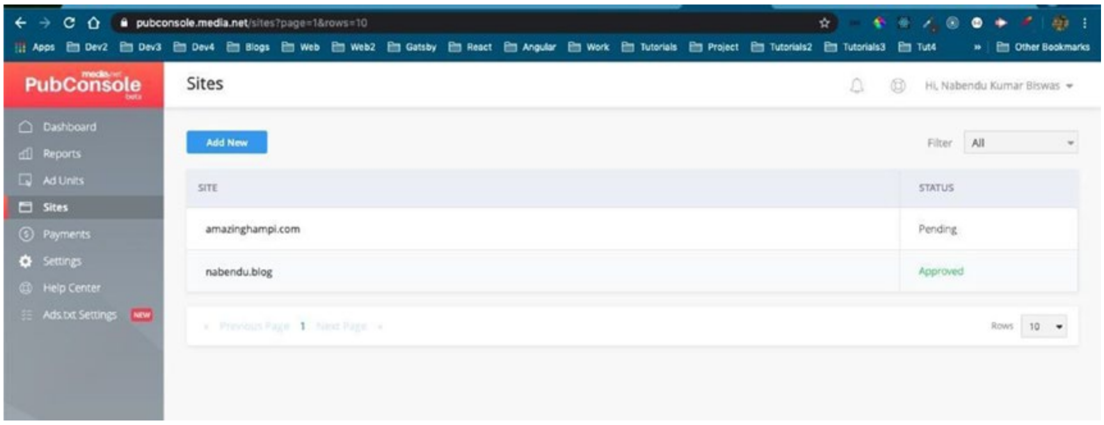


Figure 6-9. *The Add Site dialog box*

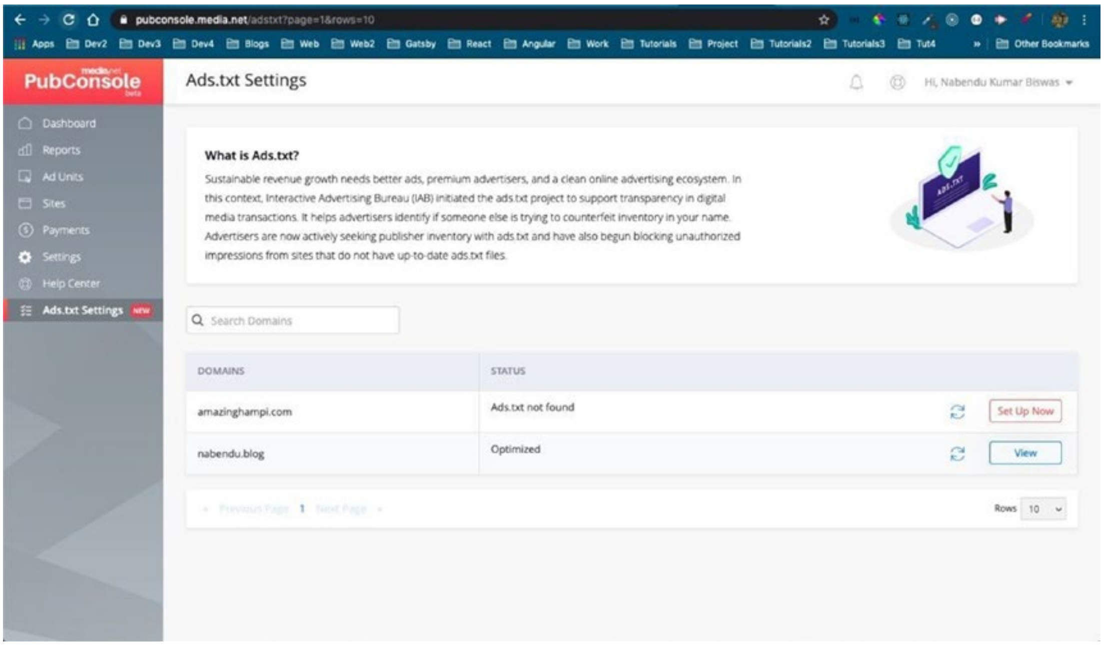
After you click the Submit button, it will show the status as Pending, as shown in Figure 6-10.



SITE	STATUS
amazinghampi.com	Pending
nabendu.blog	Approved

Figure 6-10. *Your status will be pending for a while*

Next, move to the Ads.txt Settings tab. Click the Set Up Now button on the newly added site, as shown in Figure 6-11.


Ads.txt Settings

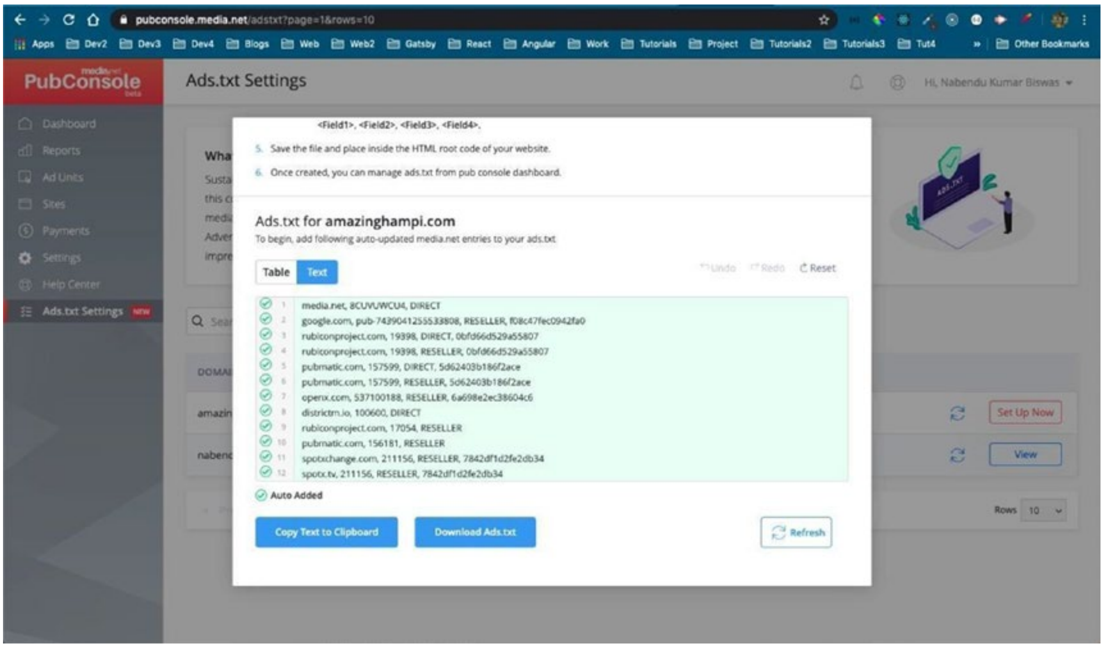
What is Ads.txt?

Sustainable revenue growth needs better ads, premium advertisers, and a clean online advertising ecosystem. In this context, Interactive Advertising Bureau (IAB) initiated the ads.txt project to support transparency in digital media transactions. It helps advertisers identify if someone else is trying to counterfeit inventory in your name. Advertisers are now actively seeking publisher inventory with ads.txt and have also begun blocking unauthorized impressions from sites that do not have up-to-date ads.txt files.

DOMAINS	STATUS
amazinghampi.com	Ads.txt not found
nabendu.blog	Optimized

Figure 6-11. The Ads.txt settings

You will get a popup to download the `Ads.txt` file, as shown in Figure 6-12.


<Field1>, <Field2>, <Field3>, <Field4>.

5. Save the file and place inside the HTML root code of your website.
6. Once created, you can manage ads.txt from pub console dashboard.

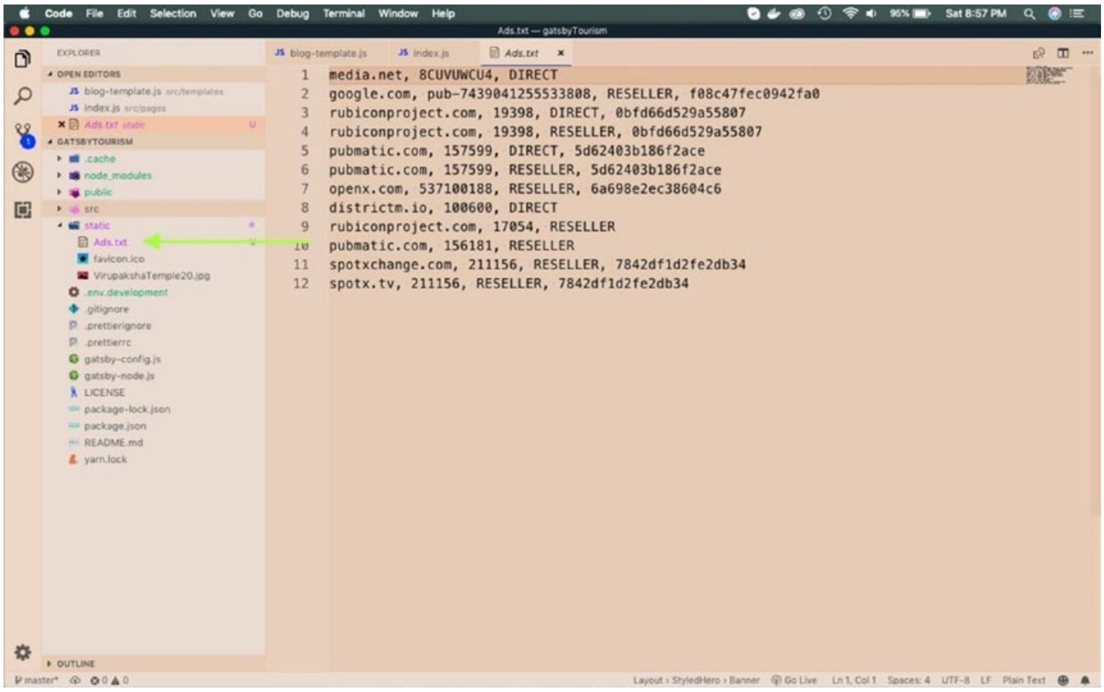
Ads.txt for **amazinghampi.com**

To begin, add following auto-updated media.net entries to your ads.txt

```
media.net, 8CUVUWCU4, DIRECT
google.com, pub-7439041255533808, RESELLER, f08c47fec0942fa0
rubiconproject.com, 19398, DIRECT, 0bfd66d529a55807
rubiconproject.com, 19398, RESELLER, 0bfd66d529a55807
pubmatic.com, 157599, DIRECT, 5d62403b186f2ace
pubmatic.com, 157599, RESELLER, 5d62403b186f2ace
openx.com, 537100188, RESELLER, 6a698e2ec38604c6
districtm.io, 100600, DIRECT
rubiconproject.com, 17054, RESELLER
pubmatic.com, 156181, RESELLER
spotxchange.com, 211156, RESELLER, 7842df1d2fe2db34
spotx.tv, 211156, RESELLER, 7842df1d2fe2db34
```

Figure 6-12. Download the Ads.txt file

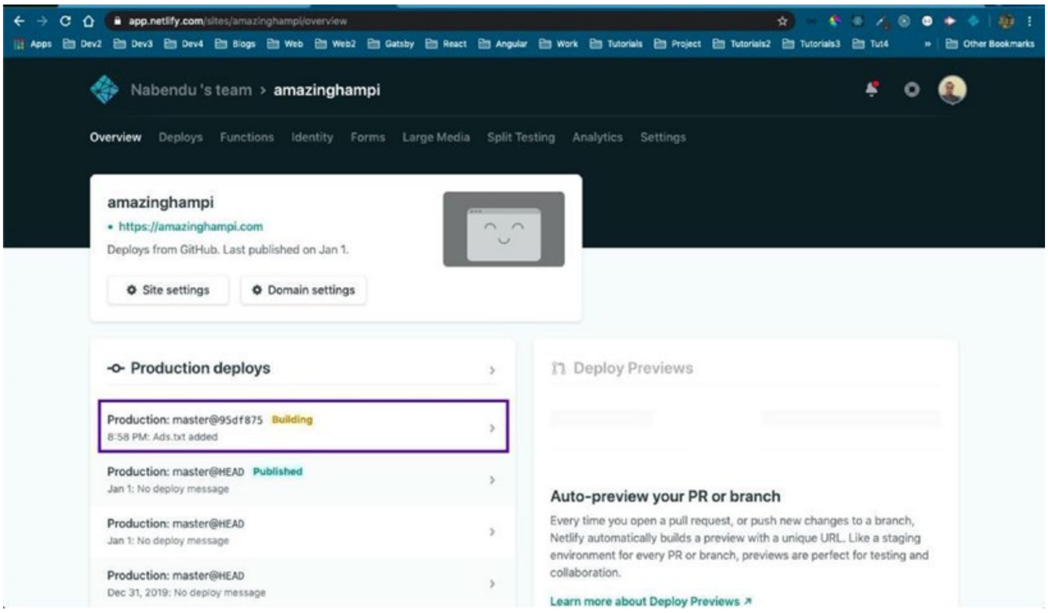
Place the `Ads.txt` file in the `static` folder of the project, as shown in Figure 6-13.



```
media.net, 8CUVUWCU4, DIRECT
google.com, pub-7439041255533808, RESELLER, f08c47fec0942fa0
rubiconproject.com, 19398, DIRECT, 0bfd66d529a55807
rubiconproject.com, 19398, RESELLER, 0bfd66d529a55807
pubmatic.com, 157599, DIRECT, 5d62403b186f2ace
pubmatic.com, 157599, RESELLER, 5d62403b186f2ace
openx.com, 537100188, RESELLER, 6a698e2ec38604c6
districtm.io, 100600, DIRECT
rubiconproject.com, 17054, RESELLER
pubmatic.com, 156181, RESELLER
spotxchange.com, 211156, RESELLER, 7842df1d2fe2db34
spotx.tv, 211156, RESELLER, 7842df1d2fe2db34
```

Figure 6-13. *Place that file in the static folder*

After that, pushing the changes to GitHub will start the automatic deployment of the site, as shown in Figure 6-14.



amazinghampi
• https://amazinghampi.com
Deploys from GitHub. Last published on Jan 1.

Production: master@95df875 Building
8:58 PM: Ads.txt added

Production: master@HEAD Published
Jan 1: No deploy message

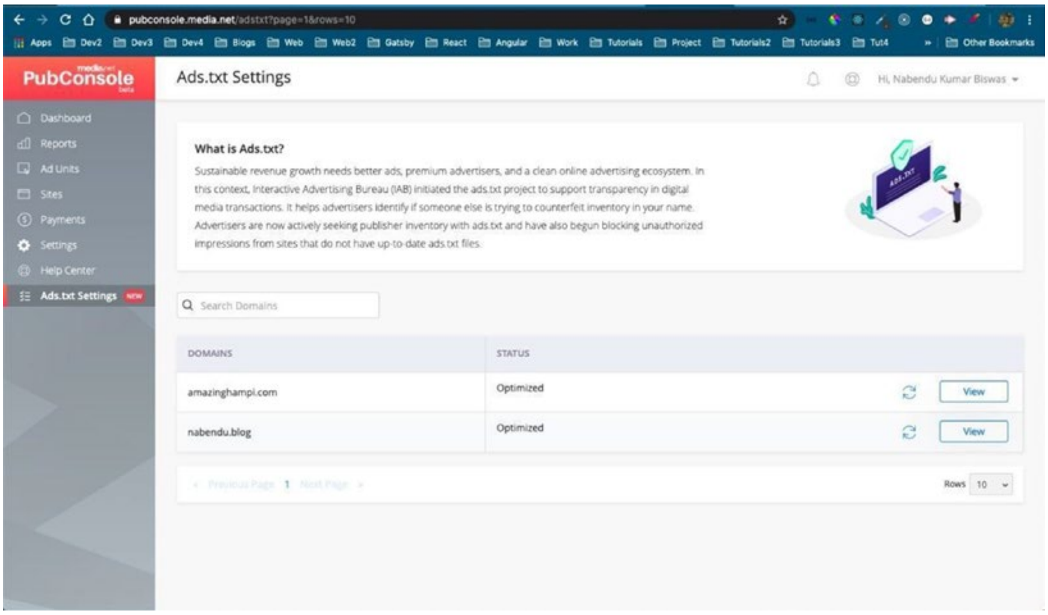
Production: master@HEAD
Jan 1: No deploy message

Production: master@HEAD
Dec 31, 2019: No deploy message

Auto-preview your PR or branch
Every time you open a pull request, or push new changes to a branch, Netlify automatically builds a preview with a unique URL. Like a staging environment for every PR or branch, previews are perfect for testing and collaboration.

Figure 6-14. *Automatic deployment begins*

Once the deployment is complete, you will see that it has Optimized status in the Ads.txt Settings tab. This is shown in Figure 6-15.

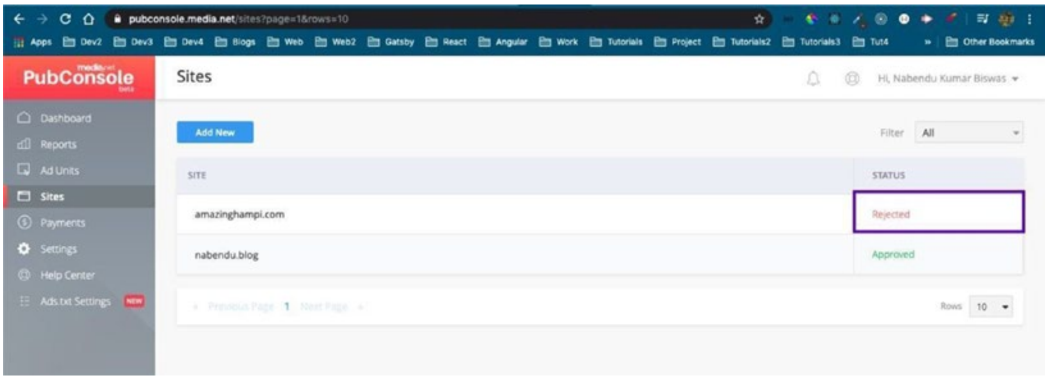


What is Ads.txt?
Sustainable revenue growth needs better ads, premium advertisers, and a clean online advertising ecosystem. In this context, Interactive Advertising Bureau (IAB) initiated the ads.txt project to support transparency in digital media transactions. It helps advertisers identify if someone else is trying to counterfeit inventory in your name. Advertisers are now actively seeking publisher inventory with ads.txt and have also begun blocking unauthorized impressions from sites that do not have up-to-date ads.txt files.

DOMAINS	STATUS	
amazinghampi.com	Optimized	View
nabendu.blog	Optimized	View

Figure 6-15. *The domain is optimized*

Media.net rejected my site, so I was not able to show ads from this network. This message is shown in Figure 6-16.

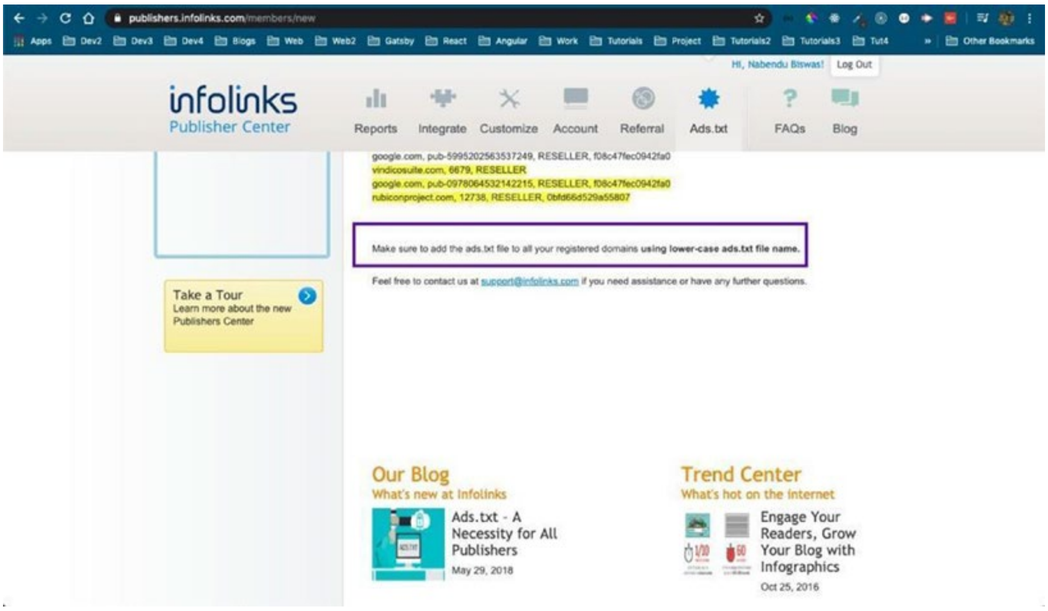


SITE	STATUS
amazinghampi.com	Rejected
nabendu.blog	Approved

Figure 6-16. *The amazinghampi site was rejected*

Using Infolinks Ads

I thought it would be smart to show another ad network, in this case Infolinks. After logging in to this site, I provided my site name. They also asked me to place an `ads.txt` on the site. See Figure 6-17.


publishers.infolinks.com/members/new
Hi, Nabendu Biswas!
Log Out
infolinks Publisher Center
Reports
Integrate
Customize
Account
Referral
Ads.txt
FAQs
Blog
Take a Tour
Learn more about the new Publishers Center
Our Blog
What's new at Infolinks
Ads.txt - A Necessity for All Publishers
May 29, 2018
Trend Center
What's hot on the internet
Engage Your Readers, Grow Your Blog with Infographics
Oct 25, 2016

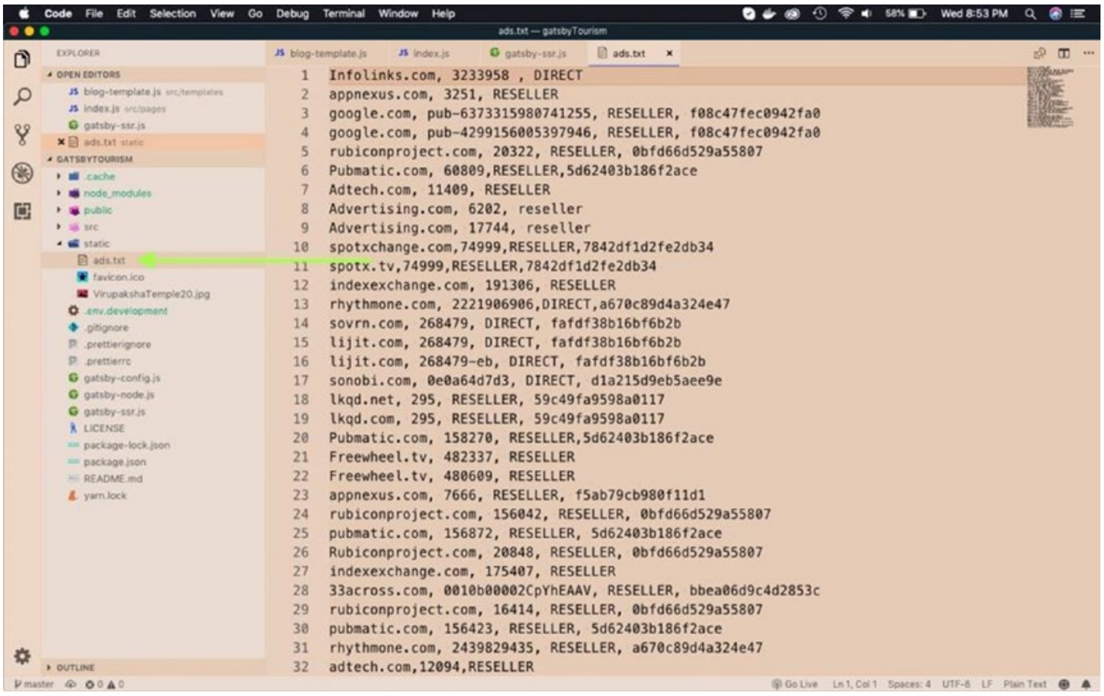
```
google.com, pub-5995202563537249, RESELLER, f08c47fec0942fa0
vindicosuite.com, 6679, RESELLER
google.com, pub-0978064532142215, RESELLER, f08c47fec0942fa0
rubiconproject.com, 12738, RESELLER, 0bfd66d529a55807
```

Make sure to add the ads.txt file to all your registered domains **using lower-case ads.txt file name.**

Feel free to contact us at support@infolinks.com if you need assistance or have any further questions.

Figure 6-17. *The new ads.txt*

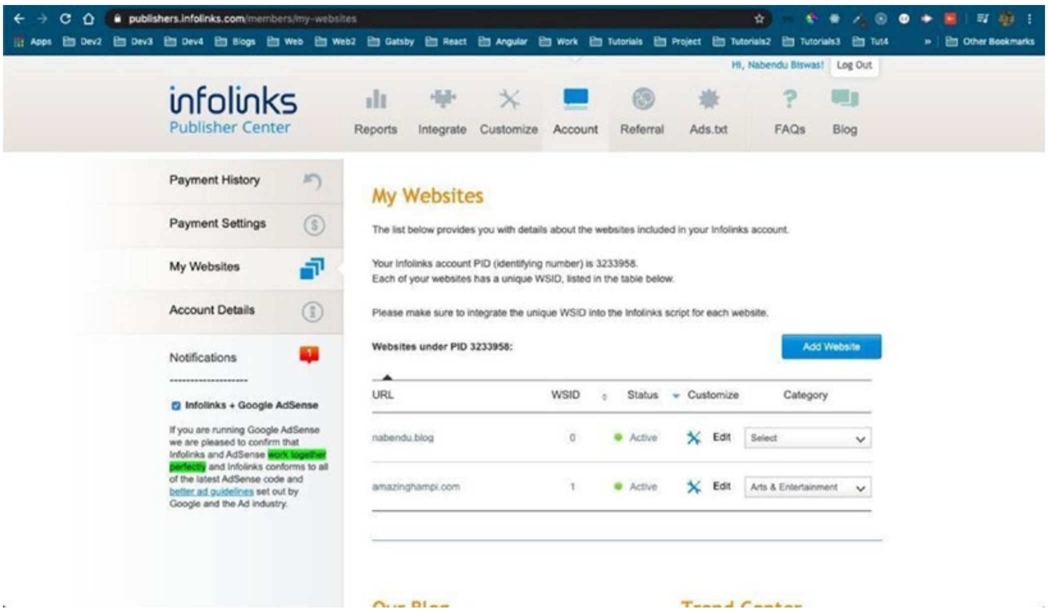
I added the new `ads.txt` file and pushed it to Netlify. It is shown in Figure 6-18.



```
Infolinks.com, 3233958 , DIRECT
appnexus.com, 3251, RESELLER
google.com, pub-6373315980741255, RESELLER, f08c47fec0942fa0
google.com, pub-4299156005397946, RESELLER, f08c47fec0942fa0
rubiconproject.com, 20322, RESELLER, 0bfd66d529a55807
Pubmatic.com, 60809,RESELLER,5d62403b186f2ace
Adtech.com, 11409, RESELLER
Advertising.com, 6202, reseller
Advertising.com, 17744, reseller
spotxchange.com,74999,RESELLER,7842df1d2fe2db34
spotx.tv,74999,RESELLER,7842df1d2fe2db34
indexexchange.com, 191306, RESELLER
rhythmone.com, 2221906906,DIRECT,a670c89d4a324e47
sovrn.com, 268479, DIRECT, fafdf38b16bf6b2b
lijit.com, 268479, DIRECT, fafdf38b16bf6b2b
lijit.com, 268479-eb, DIRECT, fafdf38b16bf6b2b
sonobi.com, 0e0a64d7d3, DIRECT, d1a215d9eb5aee9e
lkqd.net, 295, RESELLER, 59c49fa9598a0117
lkqd.com, 295, RESELLER, 59c49fa9598a0117
Pubmatic.com, 158270, RESELLER,5d62403b186f2ace
Freewheel.tv, 482337, RESELLER
Freewheel.tv, 480609, RESELLER
appnexus.com, 7666, RESELLER, f5ab79cb980f11d1
rubiconproject.com, 156042, RESELLER, 0bfd66d529a55807
pubmatic.com, 156872, RESELLER, 5d62403b186f2ace
Rubiconproject.com, 20848, RESELLER, 0bfd66d529a55807
indexexchange.com, 175407, RESELLER
33across.com, 0010b00002CpYhEAAV, RESELLER, bbea06d9c4d2853c
rubiconproject.com, 16414, RESELLER, 0bfd66d529a55807
pubmatic.com, 156423, RESELLER, 5d62403b186f2ace
rhythmone.com, 2439829435, RESELLER, a670c89d4a324e47
adtech.com,12094,RESELLER
```

Figure 6-18. *The ads.txt file*

After waiting a day, they approved my site, as shown in Figure 6-19.



If you are running Google AdSense we are pleased to confirm that Infolinks and AdSense work together perfectly and Infolinks conforms to all of the latest AdSense code and better ad guidelines set out by Google and the Ad industry.

My Websites

The list below provides you with details about the websites included in your Infolinks account.

Your Infolinks account PID (identifying number) is 3233958.
Each of your websites has a unique WSID, listed in the table below.

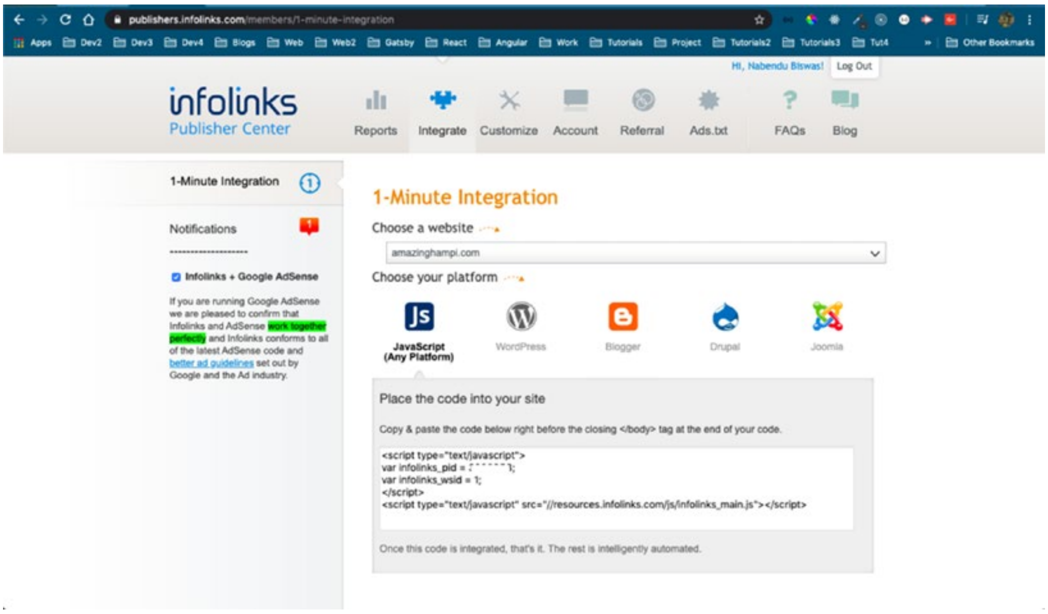
Please make sure to integrate the unique WSID into the Infolinks script for each website.

Websites under PID 3233958:

URL	WSID	Status	Customize	Category
nabendu.blog	0	● Active	Edit	Select
amazinghampi.com	1	● Active	Edit	Arts & Entertainment

Figure 6-19. *The site was approved*

The integration process is very simple. Go to the Integrate tab, choose the site, and copy the code. It is shown in Figure 6-20.



If you are running Google AdSense we are pleased to confirm that Infolinks and AdSense work together perfectly and Infolinks conforms to all of the latest AdSense code and better ad guidelines set out by Google and the Ad industry.

1-Minute Integration

Choose a website: amazinghampi.com

Choose your platform

Place the code into your site

Copy & paste the code below right before the closing </body> tag at the end of your code.

```
<script type="text/javascript">
var infolinks_pid = [redacted];
var infolinks_wsid = 1;
</script>
<script type="text/javascript" src="//resources.infolinks.com/js/infolinks_main.js"></script>
```

Once this code is integrated, that's it. The rest is intelligently automated.

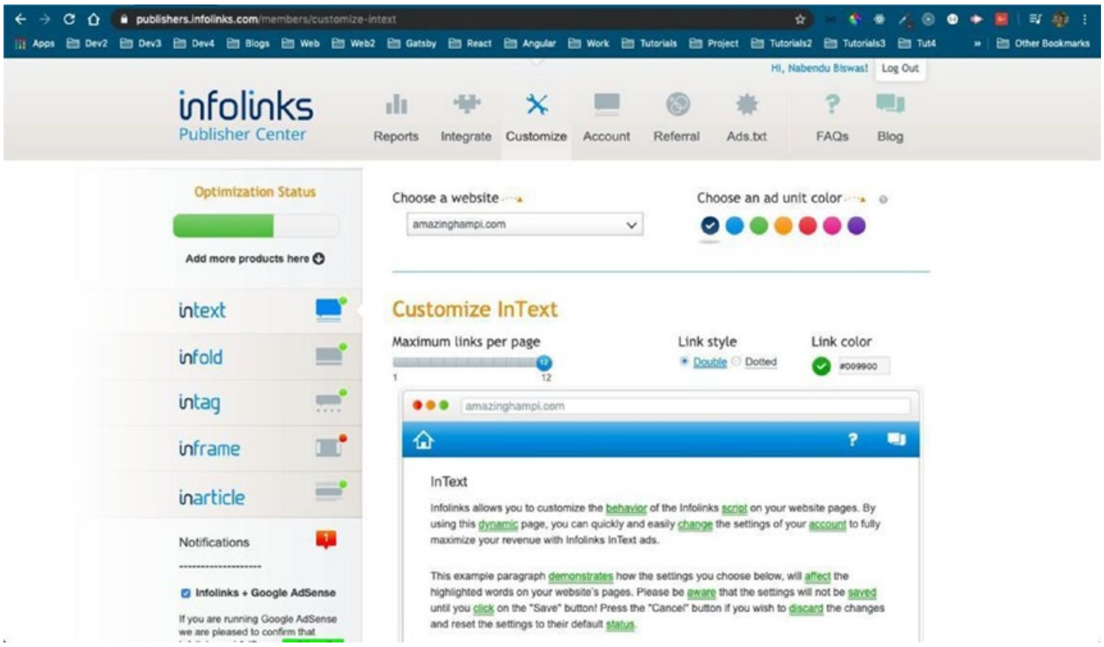
Figure 6-20. *The Integrate tab*

The way to include ads on a Gatsby site is a bit different; we also need to put this code in the `gatsby-ssr.js` file, as shown in Listing 6-19.

Listing 6-19. Ad Code in gatsby-ssr.js

```
const React = require("react");

exports.onRenderBody = function({ setHeadComponents, setPostBodyComponents }) {

    setHeadComponents([

    ]);

    setPostBodyComponents([
                <script
                    dangerouslySetInnerHTML={{
                        __html:`
                        var infolinks_pid = XXXXXXX;
                        var infolinks_wsid = 1;
                        `

                    }}
                />,
                <script type="text/javascript" src="//resources.infolinks.
                com/js/infolinks_main.js"></script>
    ]);
};
```

Unlike with Media.net, you don't need to worry about placing your ads in Infolinks. You do it using the Customize tab (see Figure 6-21).



Choose a website: amazinghampi.com

Choose an ad unit color

Customize InText

Maximum links per page: 12

Link style: [✓ Double] Dotted

Link color: #009900

InText

Infolinks allows you to customize the behavior of the Infolinks script on your website pages. By using this dynamic page, you can quickly and easily change the settings of your account to fully maximize your revenue with Infolinks InText ads.

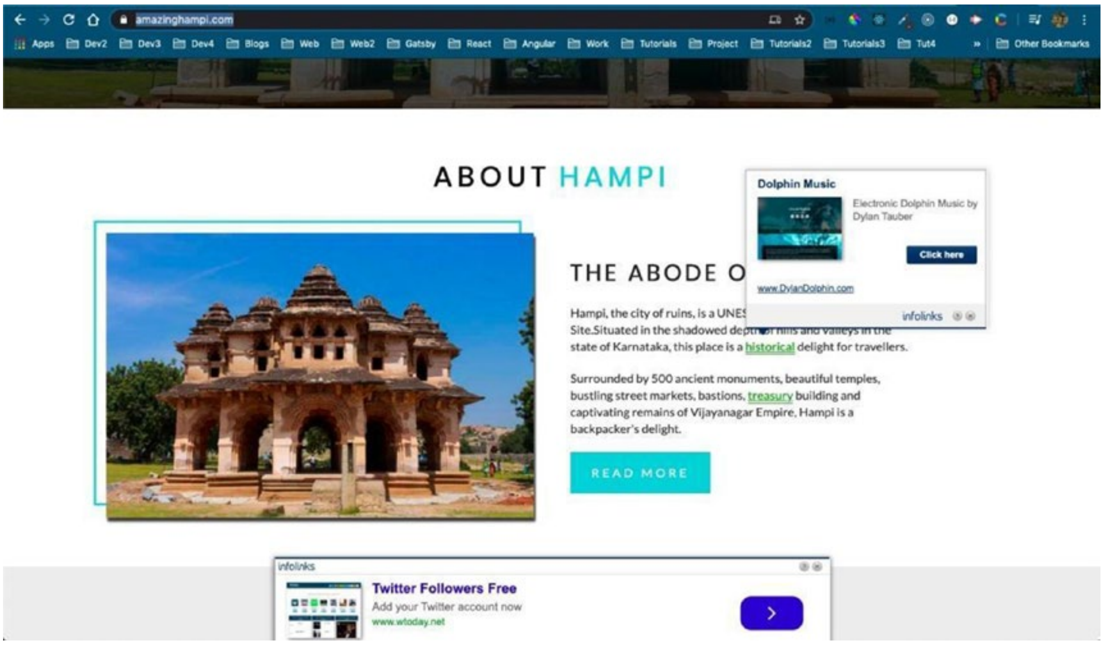
This example paragraph demonstrates how the settings you choose below, will affect the highlighted words on your website's pages. Please be aware that the settings will not be saved until you click on the "Save" button! Press the "Cancel" button if you wish to discard the changes and reset the settings to their default status.

If you are running Google AdSense we are pleased to confirm that

Figure 6-21. *Use the Customize tab*

I pushed my code to Netlify and it was deployed. Now the ads are live on `https://amazinghampi.com/`[6]. You get a floating ad and text ads (see Figure 6-22).

[6] `https://amazinghampi.com/`



Hampi, the city of ruins, is a UNES
Site.Situated in the shadowed dep
state of Karnataka, this place is a historical delight for travellers.

Surrounded by 500 ancient monuments, beautiful temples, bustling street markets, bastions, treasury building and captivating remains of Vijayanagar Empire, Hampi is a backpacker's delight.

Figure 6-22. *Ads live on the site*

I hope you adequately learned how to add advertisements to your Gatsby site in this section.

Summary

This completes Chapter 6 and the last part of creating the tourism site using Contentful. We covered the following topics in this chapter:

- Adding various Gatsby plugins to add functionality to our Gatsby site
- Adding advertising to our Gatsby site, which is more complex than adding it to a normal HTML, CSS, or JS site

In the next chapter, we will create a video chat site in Gatsby using Twilio.

CHAPTER 7

Creating a Video Chat Site

On April 2, 2020 one of my favorite sites, `dev.to`, announced a Twilio hackathon. The link is found here[1]. In this hackathon, participants had to make anything with the Twilio API and submit it before April 30, 2020. I decided to make a simple video app with Gatsby and the Twilio API. One of the main categories of this hackathon was Covid-19 communications, so it will be simple enough for anyone to use, including elderly people.

I got help from a YouTube tutorial in order to build this[2] web app. You need a Twilio account for this project. We will go through the whole process, from setting up the account to writing Twilio functions for the video chat app, in this chapter.

The Setup

In a new folder called `SimpleVideoApp`, create a new Gatsby project with the default starter, using the `gatsby new` command. The commands are shown in Listing 7-1.

Listing 7-1. Creating a New Gatsby Project

```
mkdir SimpleVideoApp
cd SimpleVideoApp
gatsby new .
```

Once the installation is done, it's time to install `twilio` and `twilio-video`. Use the command in Listing 7-2 from the terminal to do this.

[1]`https://dev.to/devteam/announcing-the-twilio-hackathon-on-dev-2lh8`
[2]`https://www.youtube.com/embed/KO2SnxY6c_0`

N. Biswas, *Foundation Gatsby Projects*, https://doi.org/10.1007/978-1-4842-6558-1_7

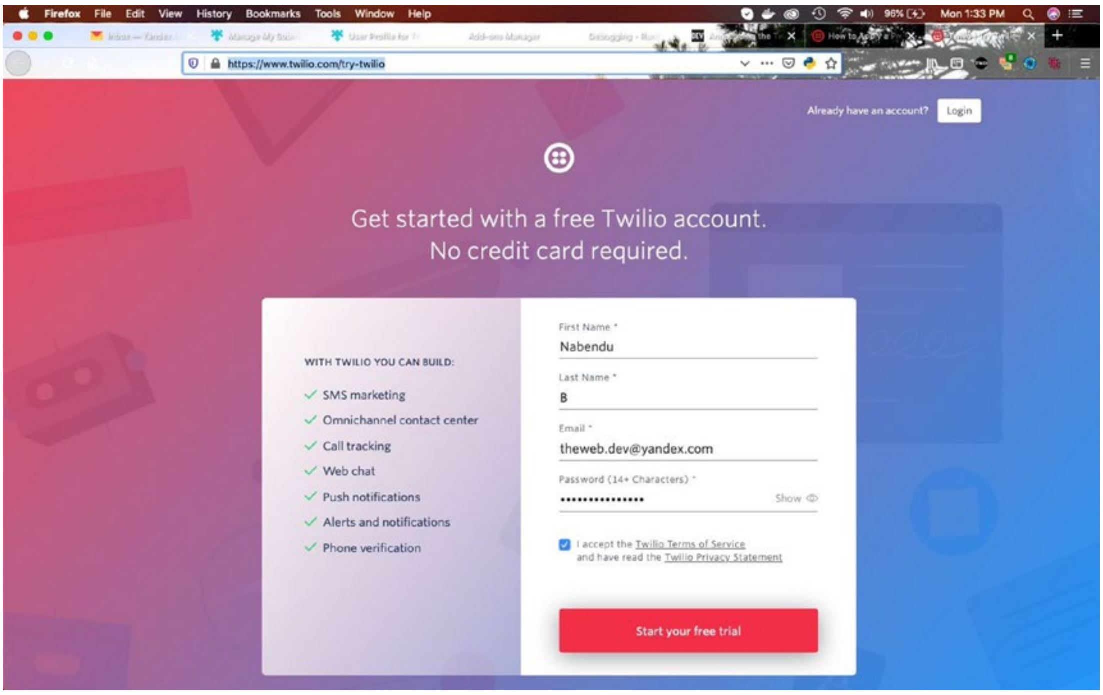
Listing 7-2. Twilio Install

```
npm i twilio
npm i twilio-video
```

It's time to create a Twilio account.

Creating a Twilio Account

Go to `https://www.twilio.com/try-twilio` to create an account. Enter the details on this page and click the Start Your Free Trial button (see Figure 7-1).


https://www.twilio.com/try-twilio

Already have an account? Login

Get started with a free Twilio account.
No credit card required.

WITH TWILIO YOU CAN BUILD:

- ✓ SMS marketing
- ✓ Omnichannel contact center
- ✓ Call tracking
- ✓ Web chat
- ✓ Push notifications
- ✓ Alerts and notifications
- ✓ Phone verification

First Name *
Nabendu

Last Name *
B

Email *
theweb.dev@yandex.com

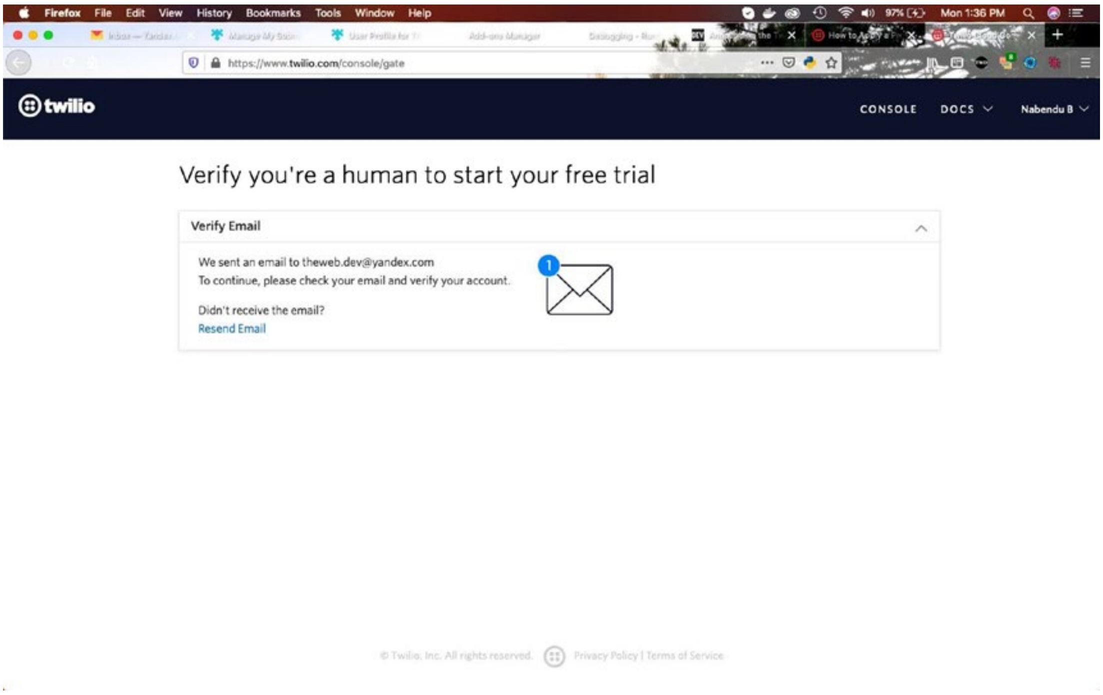
Password (14+ Characters) *
••••••••••••••• Show

✓ I accept the Twilio Terms of Service and have read the Twilio Privacy Statement

Start your free trial

Figure 7-1. *Twilio account*

You will get the usual verification email on the next page (see Figure 7-2).


Firefox File Edit View History Bookmarks Tools Window Help
Mon 1:36 PM
https://www.twilio.com/console/gate
twilio
CONSOLE
DOCS
Nabendu B

Verify you're a human to start your free trial

Verify Email

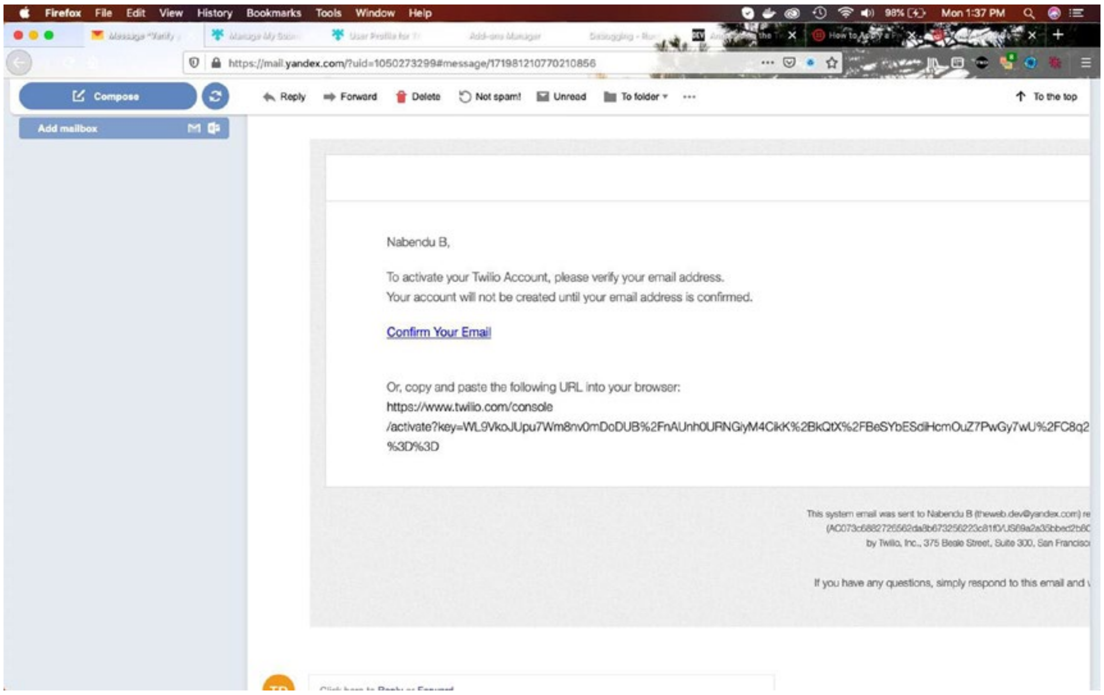
We sent an email to theweb.dev@yandex.com
To continue, please check your email and verify your account.

Didn't receive the email?
Resend Email

© Twilio, Inc. All rights reserved. Privacy Policy | Terms of Service

Figure 7-2. *Verification email*

Upon checking my mailbox, I found the mail. Click the Confirm Your Email link to continue (see Figure 7-3).



Nabendu B,

To activate your Twilio Account, please verify your email address.
Your account will not be created until your email address is confirmed.

Confirm Your Email

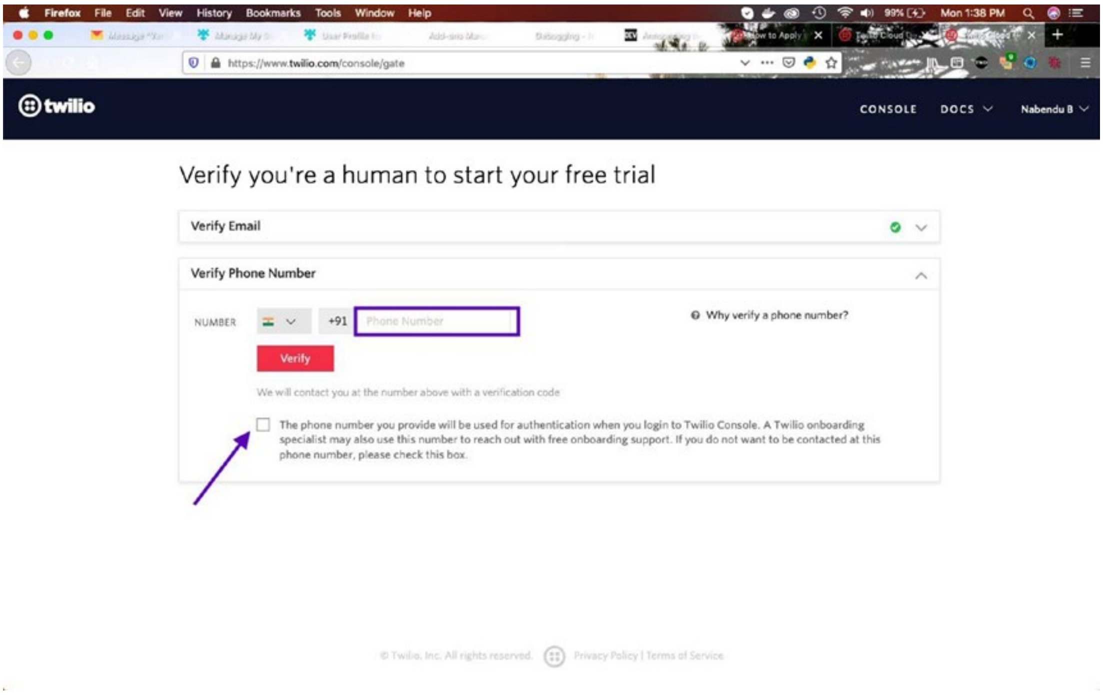
Or, copy and paste the following URL into your browser:
https://www.twilio.com/console
/activate?key=WL9VkoJUpu7Wm8nv0mDoDUB%2FnAUnh0URNGiyM4CikK%2BkQtX%2FBeSYbESdiHcmOuZ7PwGy7wU%2FC8q2
%3D%3D

This system email was sent to Nabendu B (theweb.dev@yandex.com) re
(AC073c6882726562da8b673256223c81f0/US69a2a35bbed2b8C
by Twilio, Inc., 375 Beale Street, Suite 300, San Francisc

If you have any questions, simply respond to this email and

Figure 7-3. *Confirm your email*

After that, you have to do mobile number verification. Provide a valid mobile number and click the check box; then click the Verify button (see Figure 7-4).


Firefox File Edit View History Bookmarks Tools Window Help
https://www.twilio.com/console/gate
twilio
CONSOLE
DOCS
Nabendu B

Verify you're a human to start your free trial

Verify Email

Verify Phone Number

NUMBER +91 Phone Number

Why verify a phone number?

Verify

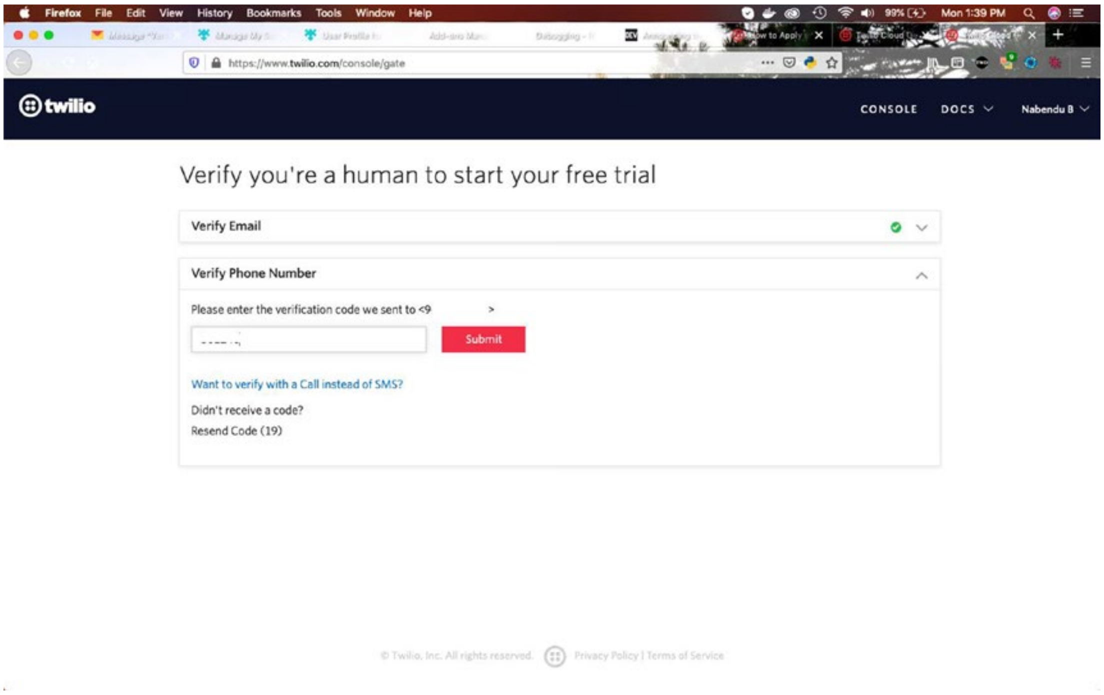
We will contact you at the number above with a verification code

The phone number you provide will be used for authentication when you login to Twilio Console. A Twilio onboarding specialist may also use this number to reach out with free onboarding support. If you do not want to be contacted at this phone number, please check this box.

© Twilio, Inc. All rights reserved. Privacy Policy | Terms of Service

Figure 7-4. *Mobile verification*

We will get a verification code on our mobile phone, which you need to enter here and then click Submit (see Figure 7-5).


Verify you're a human to start your free trial

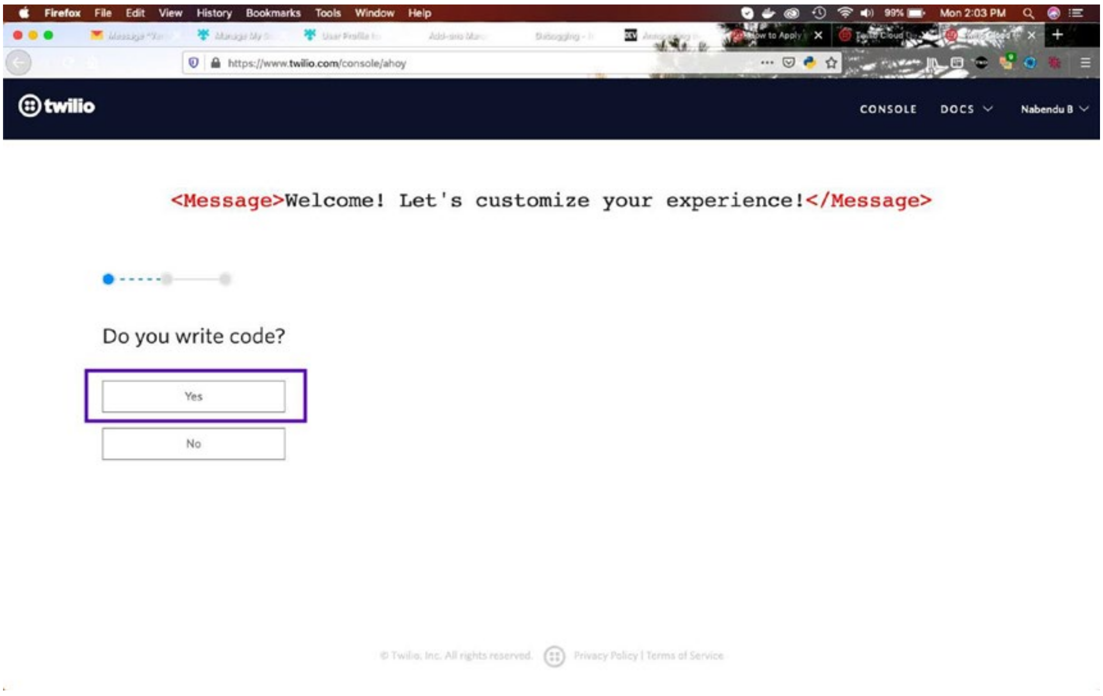
Verify Email

Verify Phone Number

Please enter the verification code we sent to <9 >

Figure 7-5. *Verification code*

After that, you'll see the screen in Figure 7-6, in which you have to state whether you code. I chose Yes.


Firefox File Edit View History Bookmarks Tools Window Help
https://www.twilio.com/console/ahoy
twilio
CONSOLE
DOCS
Nabendu B

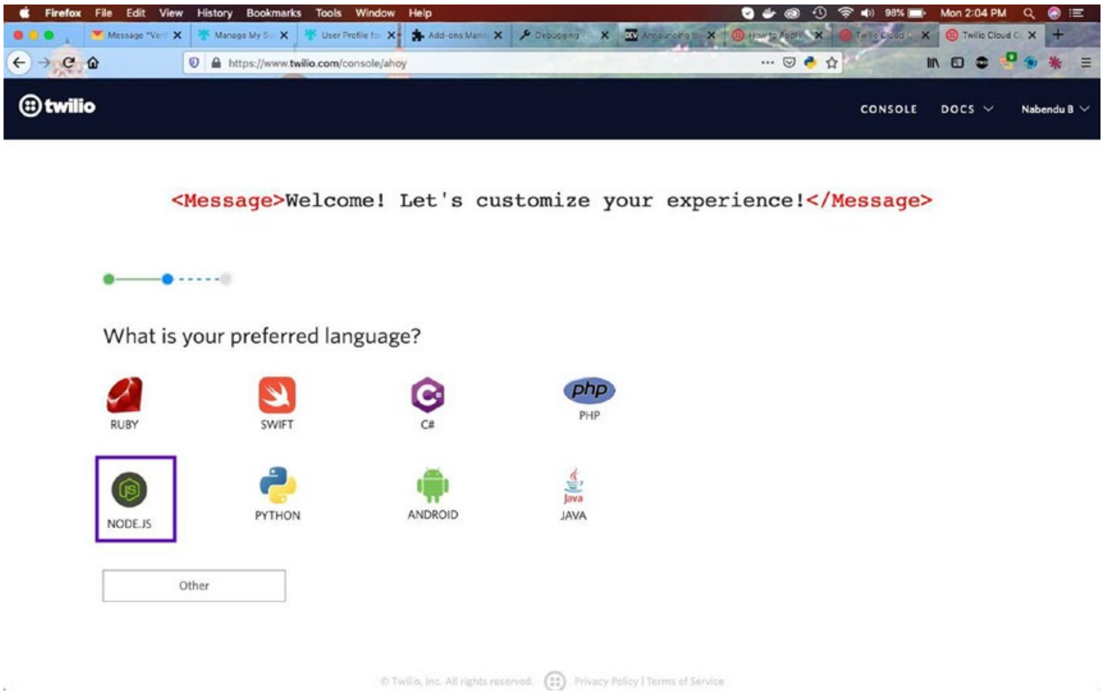
<Message>Welcome! Let's customize your experience!</Message>

Do you write code?

Yes
No
© Twilio, Inc. All rights reserved. Privacy Policy | Terms of Service

Figure 7-6. *Choose yes when asked if you write code*

In the next screen we have to choose the framework. We need to choose NodeJS here (see Figure 7-7).

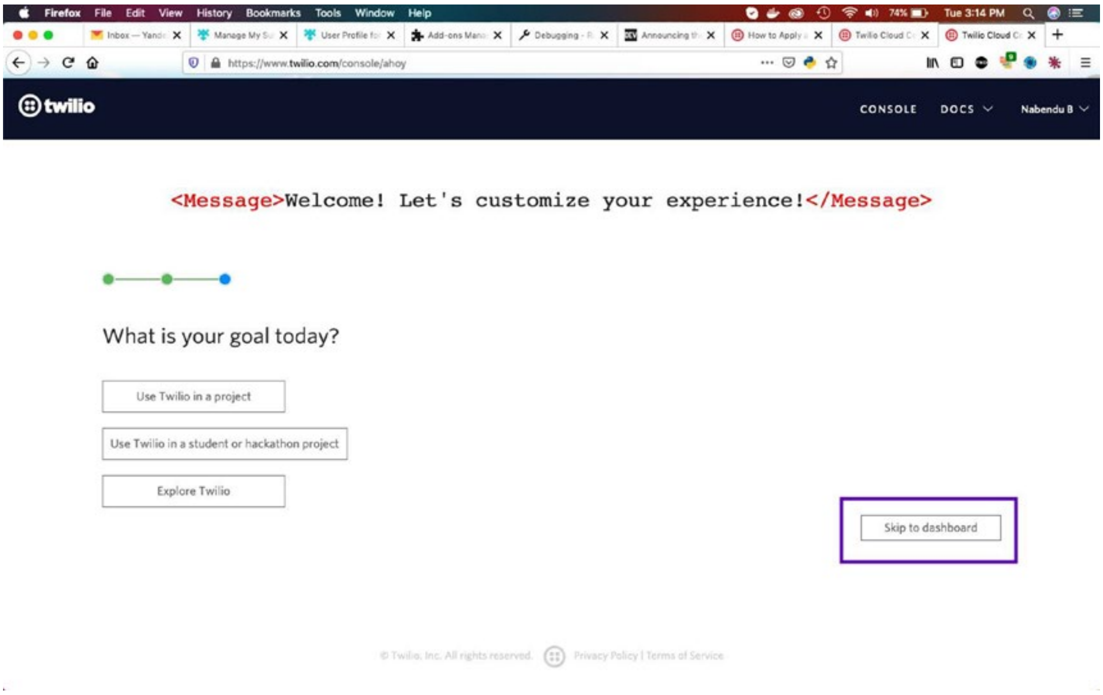


<Message>Welcome! Let's customize your experience!</Message>

What is your preferred language?

Figure 7-7. *Choose NodeJS*

Next, click the Skip to Dashboard button, as shown in Figure 7-8.

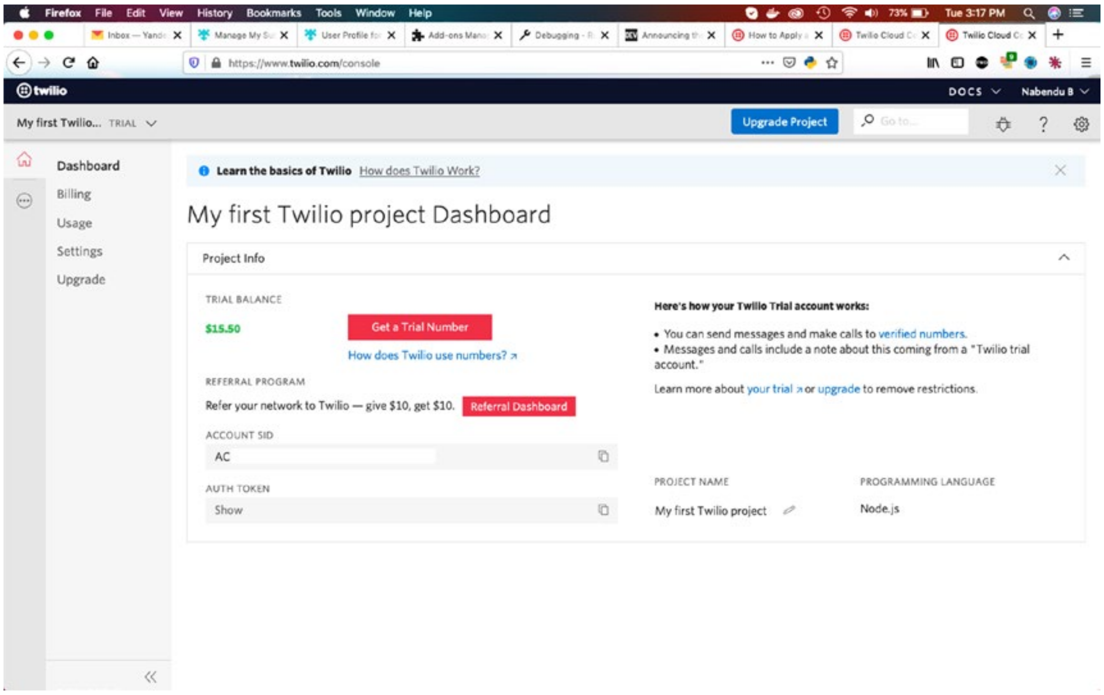


<Message>Welcome! Let's customize your experience!</Message>

What is your goal today?

Figure 7-8. *Skip to the dashboard*

This will take you to the dashboard (see Figure 7-9).



Learn the basics of Twilio How does Twilio Work?

My first Twilio project Dashboard

Project Info

TRIAL BALANCE
$15.50

REFERRAL PROGRAM
Refer your network to Twilio — give $10, get $10.

ACCOUNT SID
AC

AUTH TOKEN
Show

Here's how your Twilio Trial account works:

- You can send messages and make calls to verified numbers.
- Messages and calls include a note about this coming from a "Twilio trial account."

Learn more about your trial or upgrade to remove restrictions.

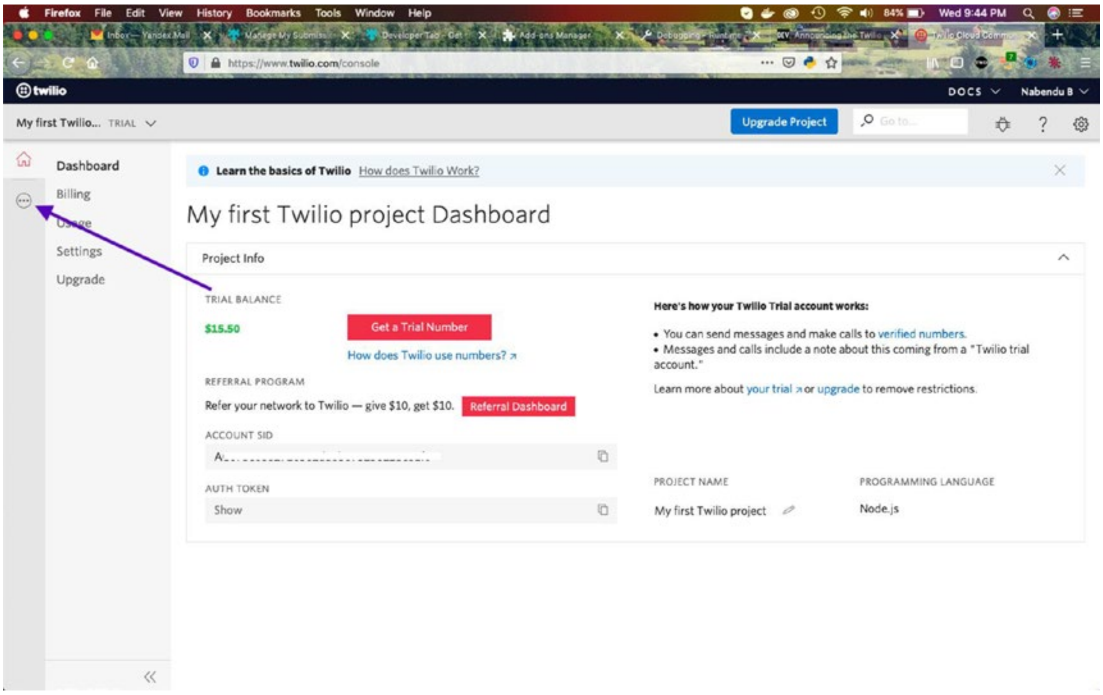
PROJECT NAME
My first Twilio project

PROGRAMMING LANGUAGE
Node.js

Figure 7-9. *The Twilio dashboard*

Working in the Dashboard

There are a lot of settings in the dashboard, including creating a Twilio function. We logged into the Twilio dashboard earlier. Click the three dots on the left menu, as shown in Figure 7-10.



Learn the basics of Twilio How does Twilio Work?

My first Twilio project Dashboard

Project Info

TRIAL BALANCE
$15.50

How does Twilio use numbers?

REFERRAL PROGRAM
Refer your network to Twilio — give $10, get $10.

ACCOUNT SID

AUTH TOKEN

Here's how your Twilio Trial account works:
- You can send messages and make calls to verified numbers.
- Messages and calls include a note about this coming from a "Twilio trial account."

Learn more about your trial or upgrade to remove restrictions.

PROJECT NAME
My first Twilio project

PROGRAMMING LANGUAGE
Node.js

Figure 7-10. *Click the three dots*

Next, click Programmable Video from the menu (see Figure 7-11).

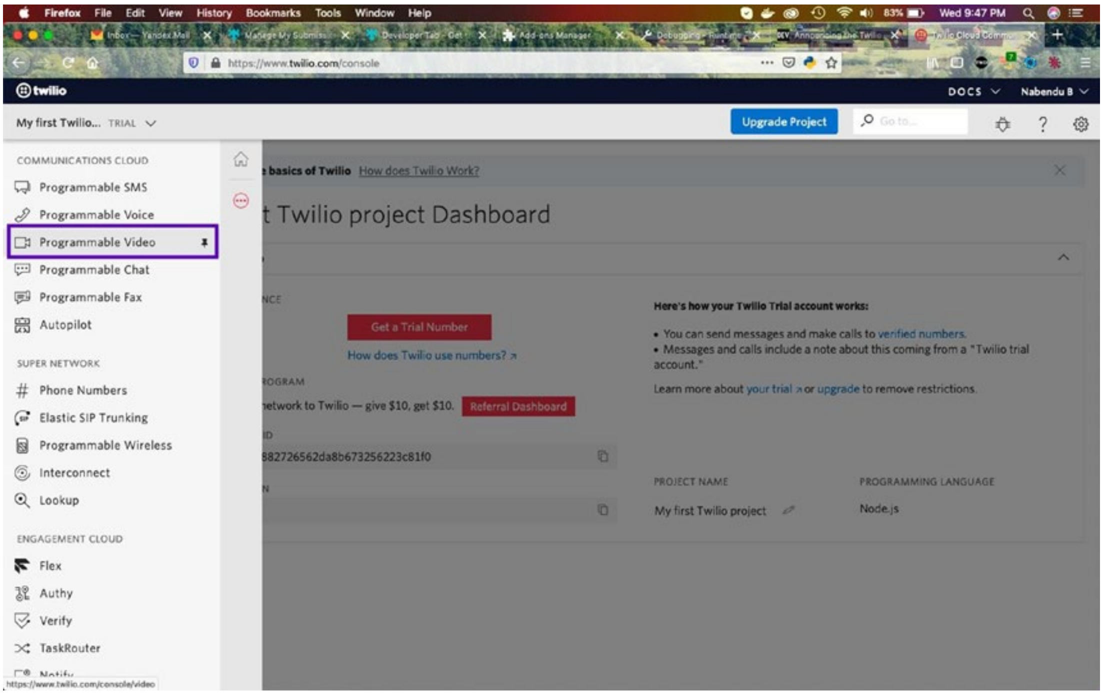


Figure 7-11. *Choose Programmable Video*

On the next screen, click the Show API Credentials link (see Figure 7-12).

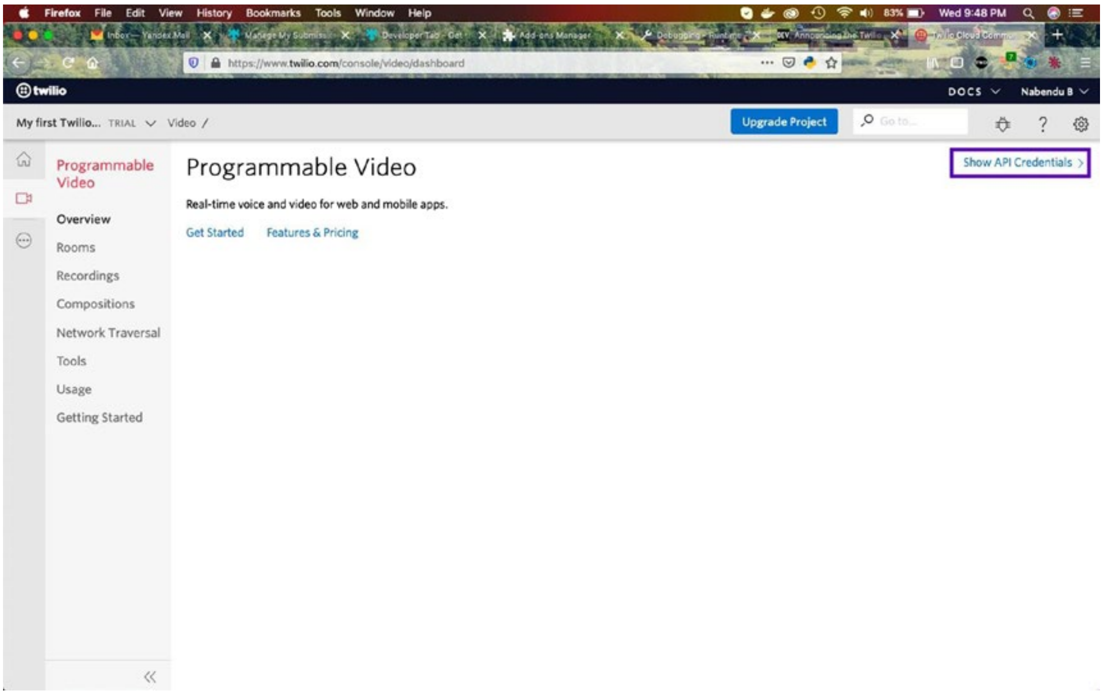
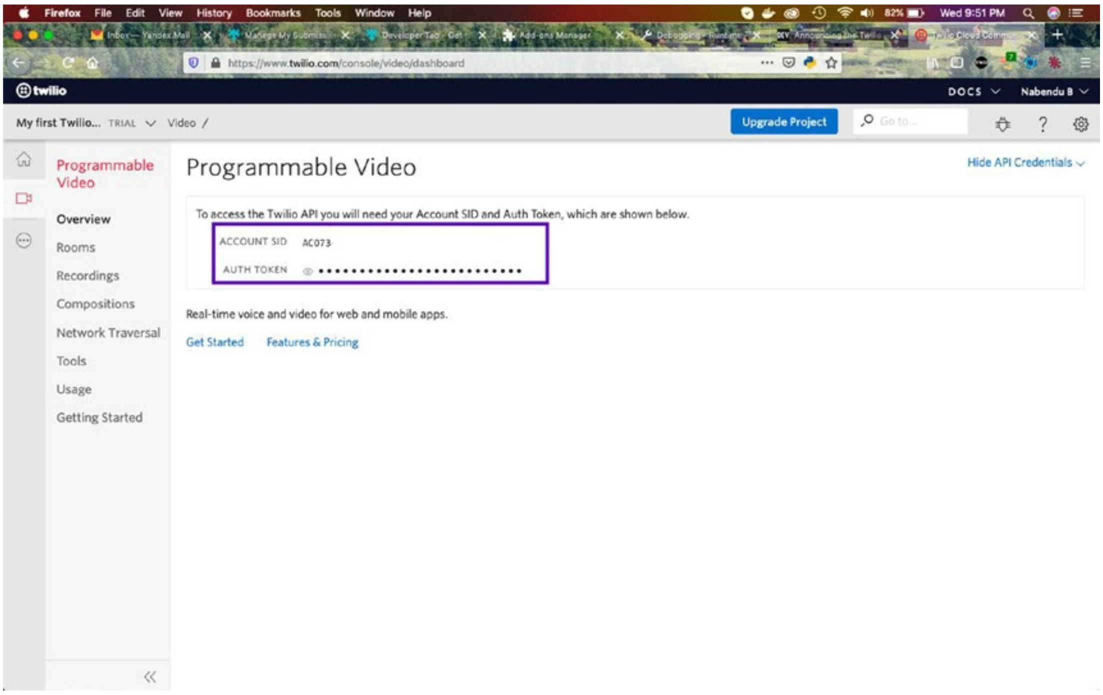


Figure 7-12. *Click Show API Credentials*

On the next page, note your Account SID and Auth Token (see Figure 7-13).



Programmable Video

To access the Twilio API you will need your Account SID and Auth Token, which are shown below.

ACCOUNT SID AC073

AUTH TOKEN ••••••••••••••••••••••••••

Real-time voice and video for web and mobile apps.

Figure 7-13. *Write down these credentials somewhere safe*

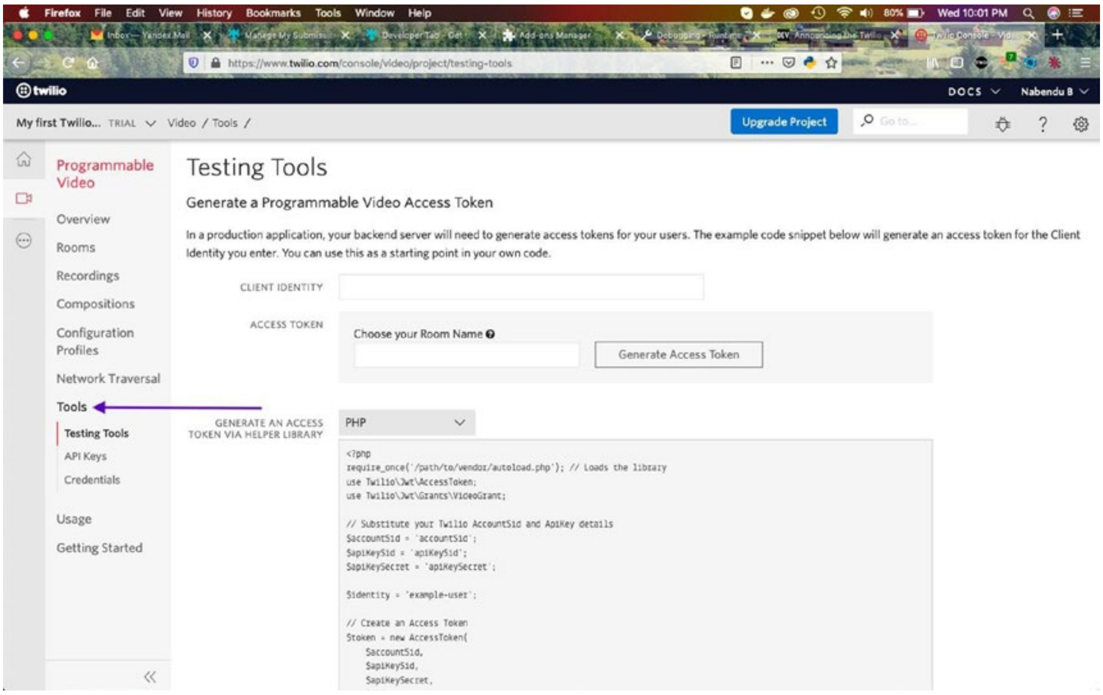
Updating the API Key Settings

It's time to put these secrets in an `.env` file. Create an `.env` file in the root directory and put the account SID and auth token into the `TWILIO_ACC_SID` and `TWILIO_AUTH_TOKEN` variables, respectively. The content is shown in Listing 7-3.

Listing 7-3. The Environment File

```
TWILIO_ACC_SID=XXXXXXXXXXXX
TWILIO_AUTH_TOKEN=XXXXXXXXXX
```

Then choose Tools from the left menu (see Figure 7-14).



Testing Tools

Generate a Programmable Video Access Token

In a production application, your backend server will need to generate access tokens for your users. The example code snippet below will generate an access token for the Client Identity you enter. You can use this as a starting point in your own code.

CLIENT IDENTITY

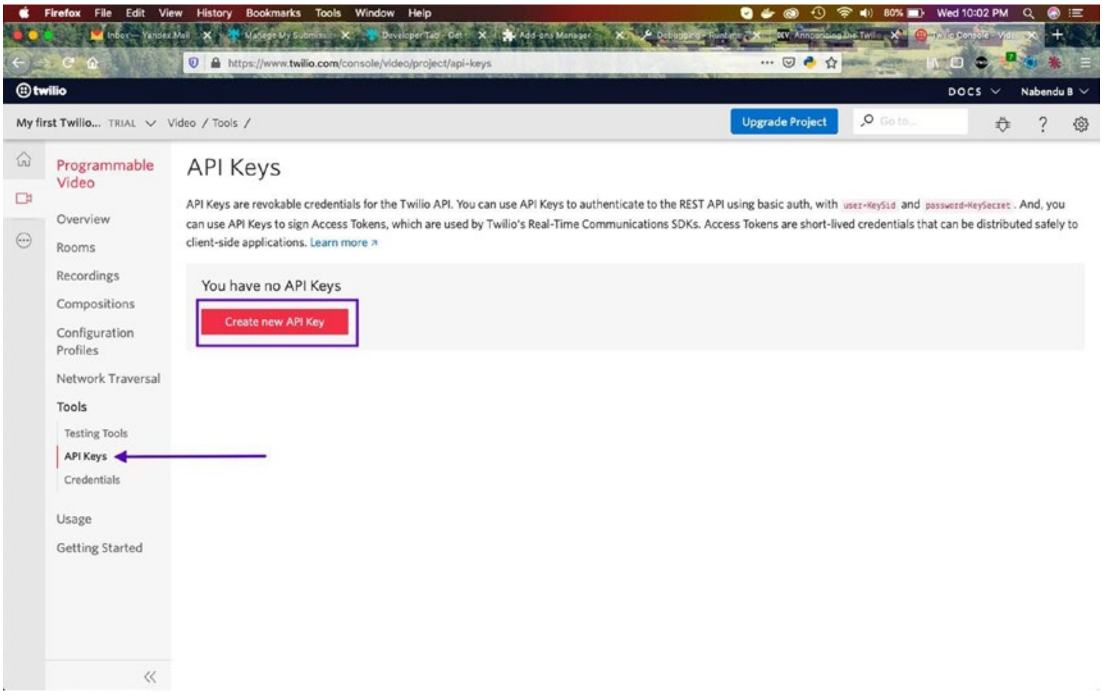
ACCESS TOKEN — Choose your Room Name — Generate Access Token

GENERATE AN ACCESS TOKEN VIA HELPER LIBRARY — PHP

```
<?php
require_once('/path/to/vendor/autoload.php'); // Loads the library
use Twilio\Jwt\AccessToken;
use Twilio\Jwt\Grants\VideoGrant;

// Substitute your Twilio AccountSid and ApiKey details
$accountSid = 'accountSid';
$apiKeySid = 'apiKeySid';
$apiKeySecret = 'apiKeySecret';

$identity = 'example-user';

// Create an Access Token
$token = new AccessToken(
    $accountSid,
    $apiKeySid,
    $apiKeySecret,
```

Figure 7-14. *Tools*

After that, click the API Keys submenu link. Next, we have to click the Create New API Key button (see Figure 7-15).



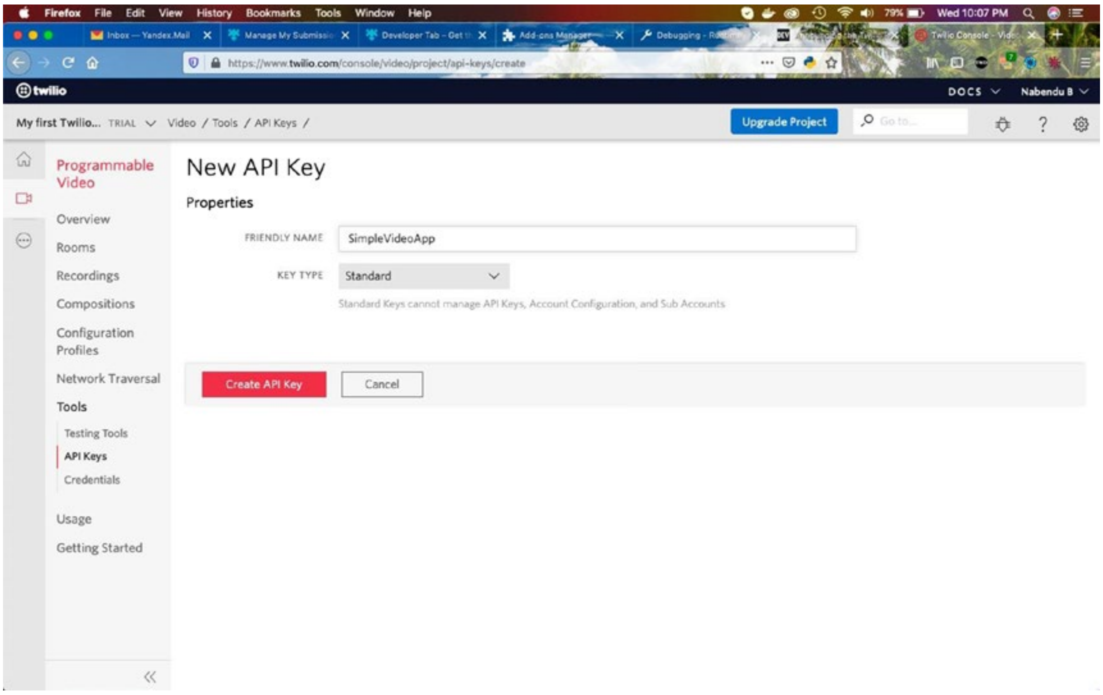
API Keys

API Keys are revokable credentials for the Twilio API. You can use API Keys to authenticate to the REST API using basic auth, with `user=KeySid` and `password=KeySecret`. And, you can use API Keys to sign Access Tokens, which are used by Twilio's Real-Time Communications SDKs. Access Tokens are short-lived credentials that can be distributed safely to client-side applications. Learn more ↗

You have no API Keys

Figure 7-15. *Create a new API key*

On the next screen, we have to give the new API key a name and then click the Create API Key button (see Figure 7-16).



New API Key

Properties

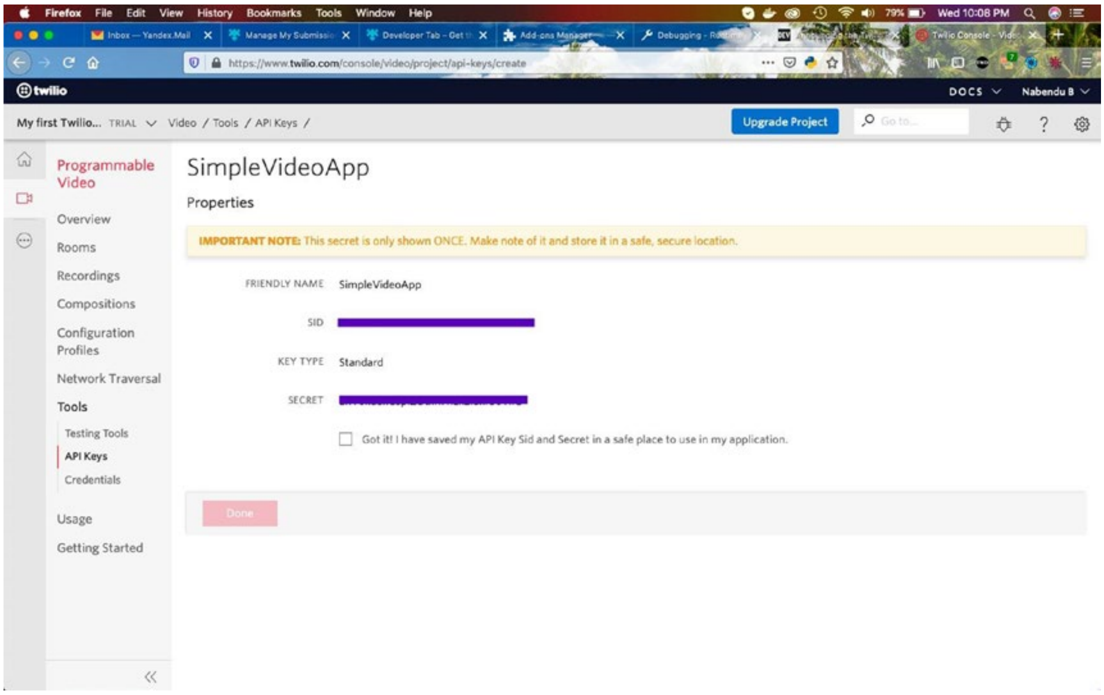
FRIENDLY NAME: SimpleVideoApp

KEY TYPE: Standard

Standard Keys cannot manage API Keys, Account Configuration, and Sub Accounts

Figure 7-16. *Provide a name here*

On the next screen, we will be shown the `SID` and the `SECRET`. We need to note them both (see Figure 7-17).



SimpleVideoApp

Properties

IMPORTANT NOTE: This secret is only shown ONCE. Make note of it and store it in a safe, secure location.

FRIENDLY NAME SimpleVideoApp

SID [redacted]

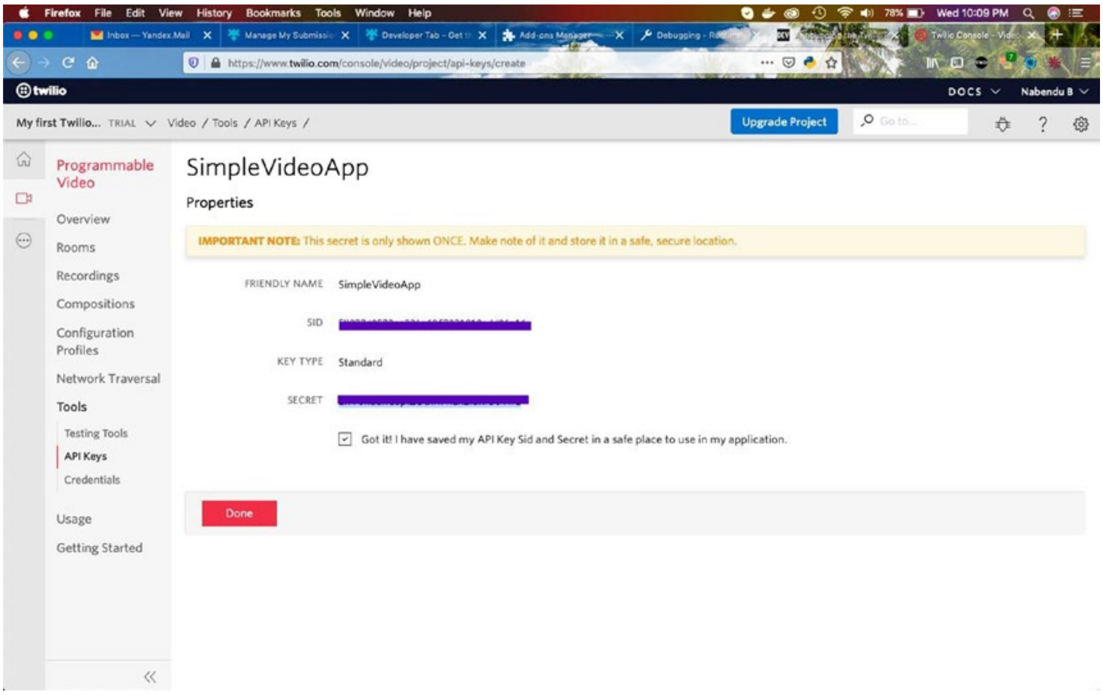
KEY TYPE Standard

SECRET [redacted]

☐ Got it! I have saved my API Key Sid and Secret in a safe place to use in my application.

Figure 7-17. *Note the SID and secret*

Then click the check box and the Done button (see Figure 7-18).



SimpleVideoApp

Properties

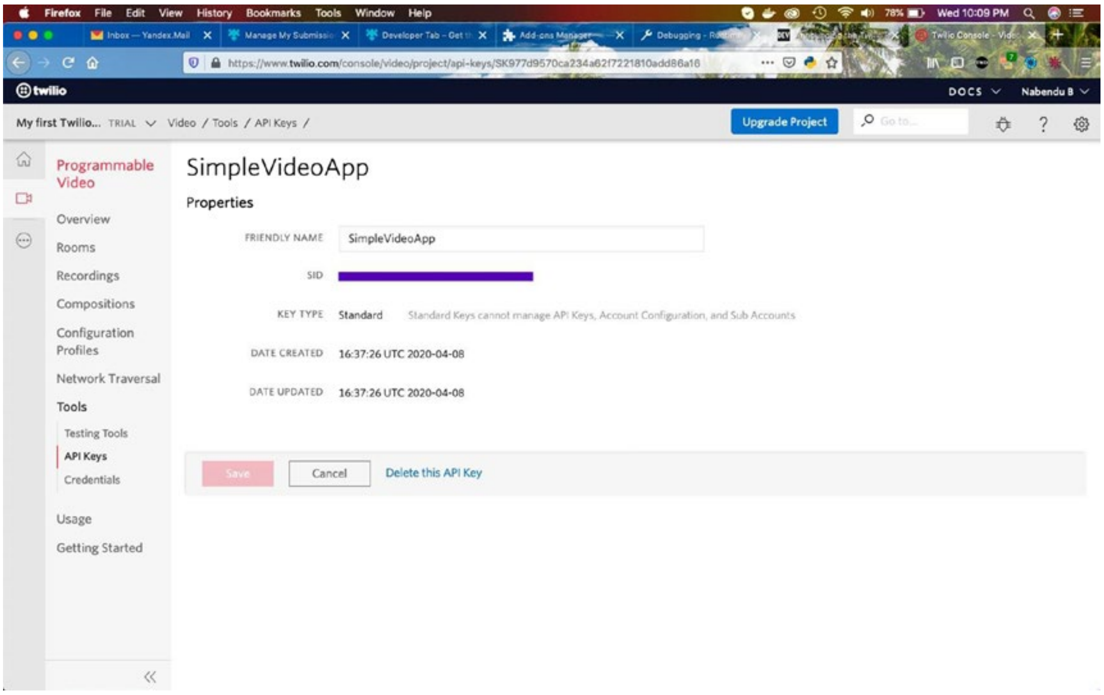
IMPORTANT NOTE: This secret is only shown ONCE. Make note of it and store it in a safe, secure location.

FRIENDLY NAME	SimpleVideoApp
SID	[redacted]
KEY TYPE	Standard
SECRET	[redacted]

☑ Got it! I have saved my API Key Sid and Secret in a safe place to use in my application.

Figure 7-18. *Check the box to indicate you saved the information*

On the next screen, the secret won't be visible. This means our API keys have been saved (see Figure 7-19).


SimpleVideoApp

Properties

FRIENDLY NAME SimpleVideoApp

SID

KEY TYPE Standard Standard Keys cannot manage API Keys, Account Configuration, and Sub Accounts

DATE CREATED 16:37:26 UTC 2020-04-08

DATE UPDATED 16:37:26 UTC 2020-04-08

Figure 7-19. *The API keys have been saved*

Again, go to the `.env` file and save the two variables in Listing 7-4.

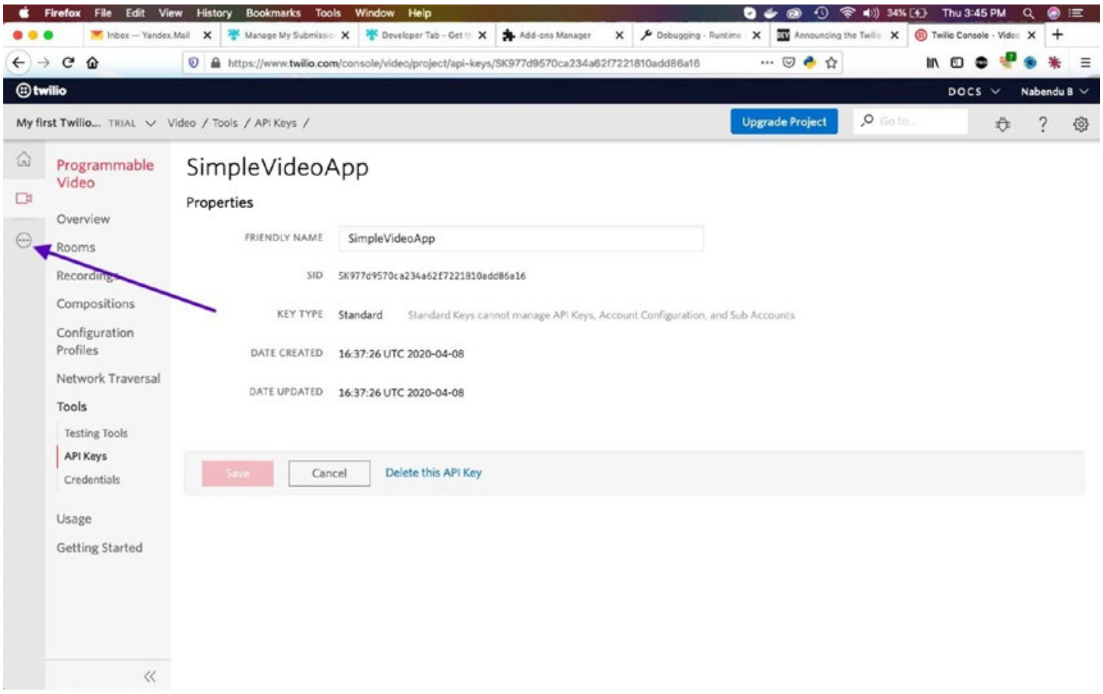
Listing 7-4. Environment File with Additional Keys

```
TWILIO_ACC_SID=XXXXXXXXXXXX
TWILIO_AUTH_TOKEN=XXXXXXXXXX
TWILIO_API_SID=XXXXXXXXXXXX
TWILIO_API_SECRET=XXXXXXXXXX
```

The settings of the API keys took a lot of time, so we will start with Twilio functions in the next section.

Creating Twilio Functions

We will finally start creating Twilio functions. First, click the three dots on the left side of the Twilio dashboard (see Figure 7-20).



SimpleVideoApp

Properties

FRIENDLY NAME SimpleVideoApp

SID SK977d9570ca234a62f7221810add86a16

KEY TYPE Standard Standard Keys cannot manage API Keys, Account Configuration, and Sub Accounts

DATE CREATED 16:37:26 UTC 2020-04-08

DATE UPDATED 16:37:26 UTC 2020-04-08

Figure 7-20. *The Twilio dashboard*

Next, if you scroll down a bit, you will find the Functions menu; click it (see Figure 7-21).

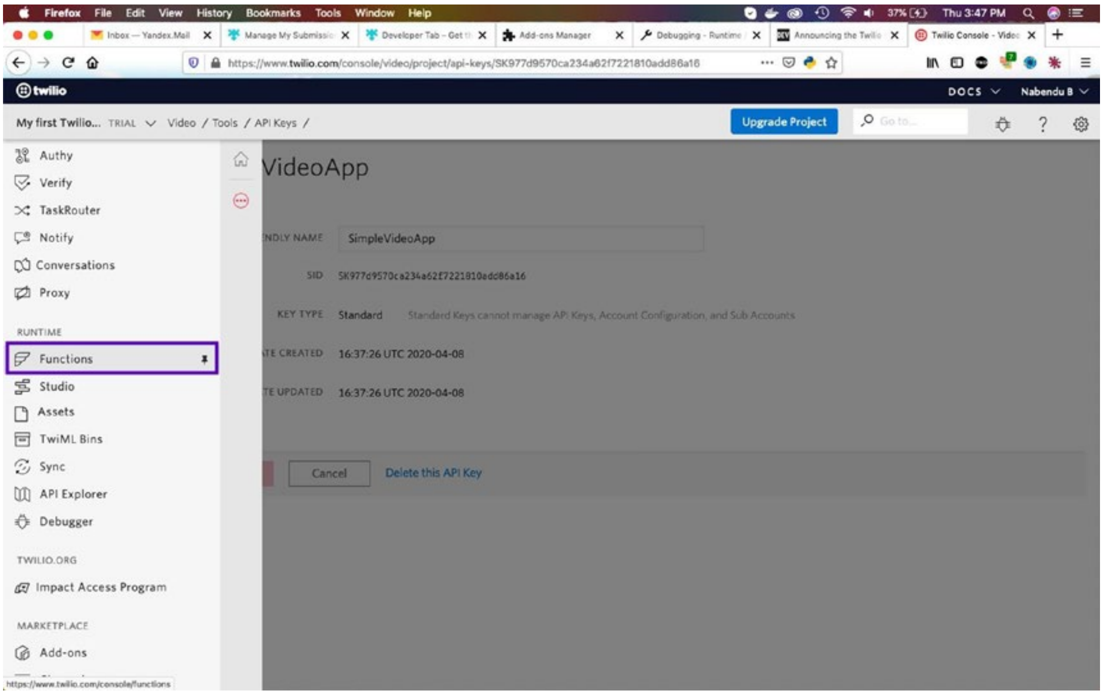
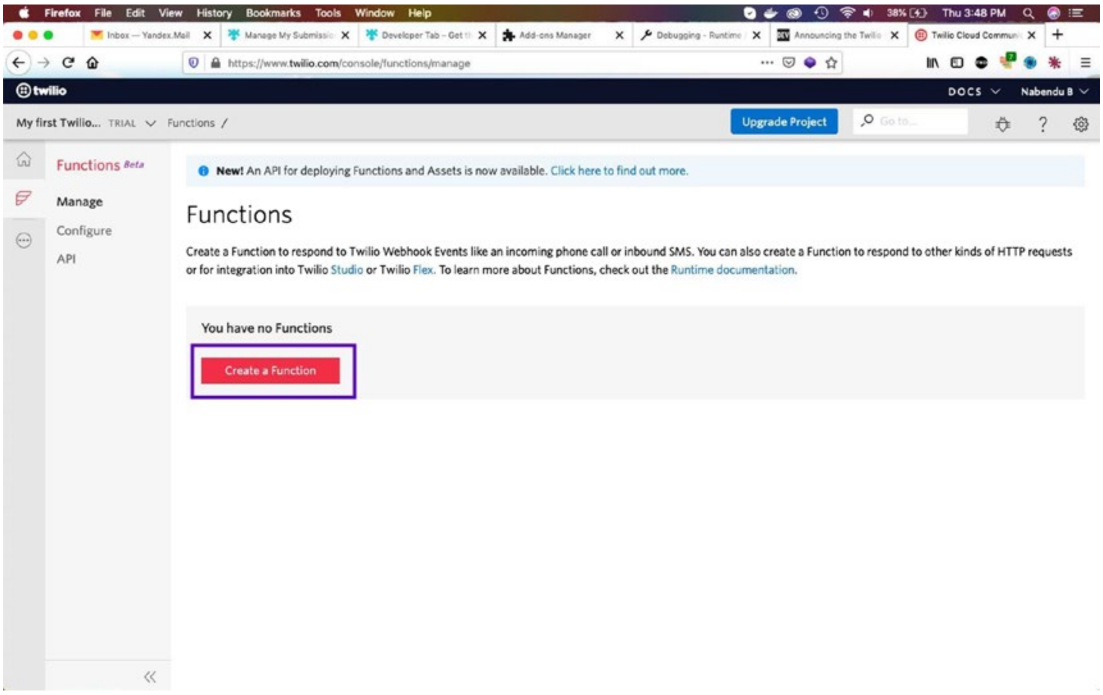


Figure 7-21. *Functions*

On the next screen, click the Create a Function button (see Figure 7-22).



New! An API for deploying Functions and Assets is now available. Click here to find out more.

Functions

Create a Function to respond to Twilio Webhook Events like an incoming phone call or inbound SMS. You can also create a Function to respond to other kinds of HTTP requests or for integration into Twilio Studio or Twilio Flex. To learn more about Functions, check out the Runtime documentation.

You have no Functions

Figure 7-22. *Create a function*

A popup will appear in which you need to click Blank and then click the Create button (see Figure 7-23).

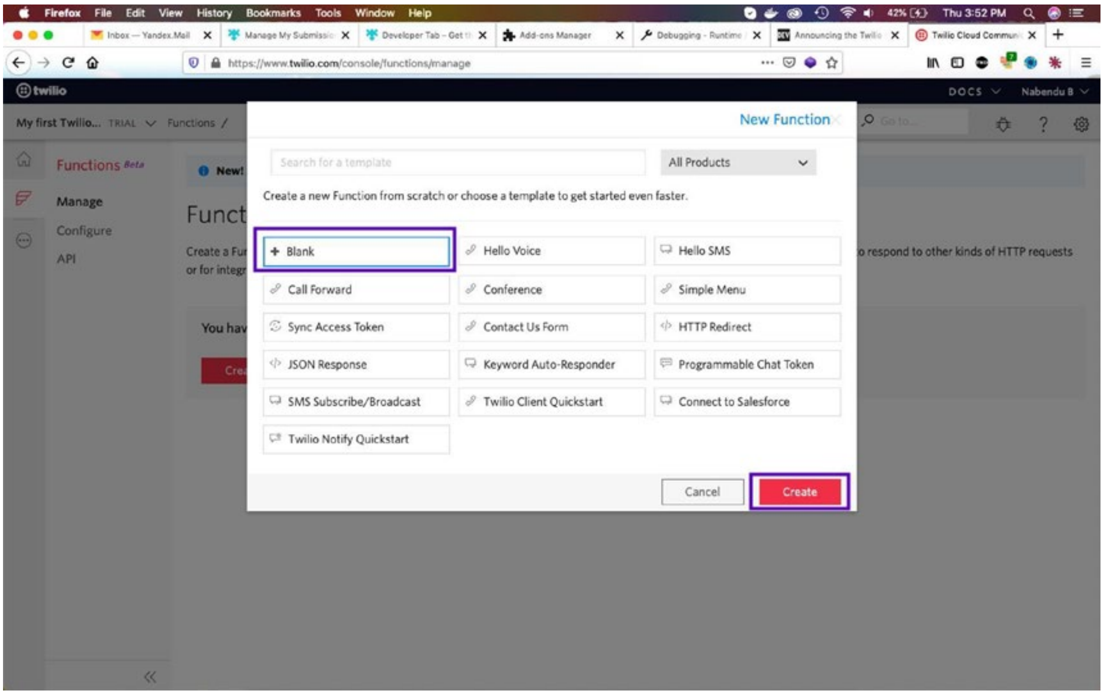
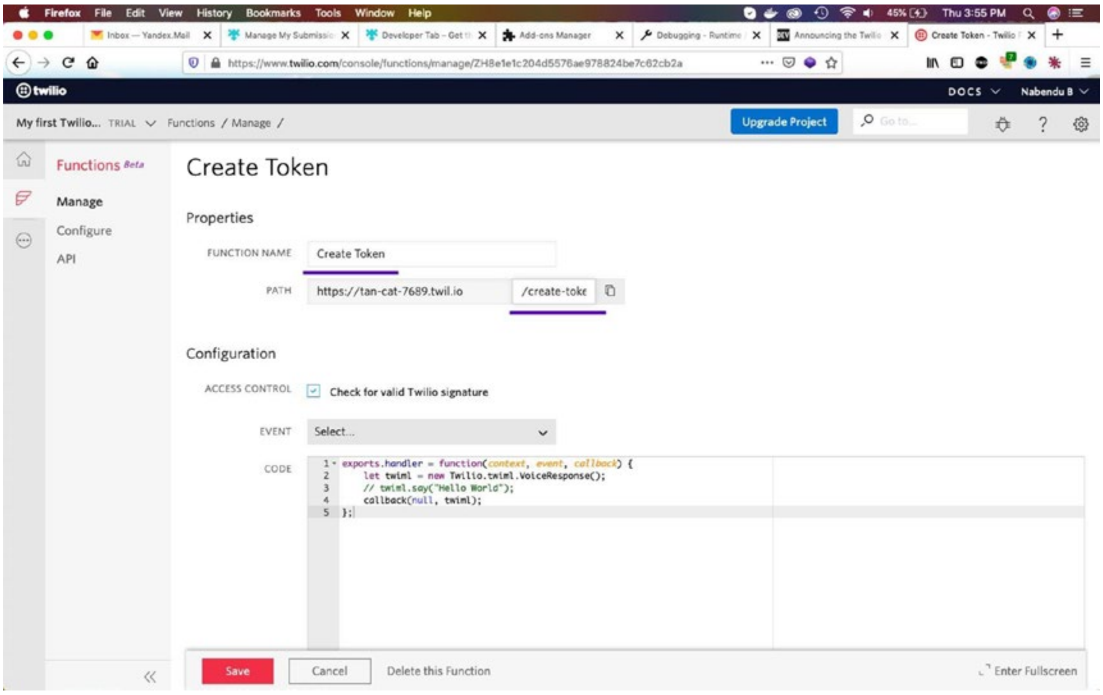


Figure 7-23. *Create a blank function*

On the next page, we need to give our function a name and a path. I named them Create Token and `/create-token`, respectively (see Figure 7-24).



Create Token

Properties

FUNCTION NAME: Create Token

PATH: https://tan-cat-7689.twil.io /create-toke

Configuration

ACCESS CONTROL: [✓] Check for valid Twilio signature

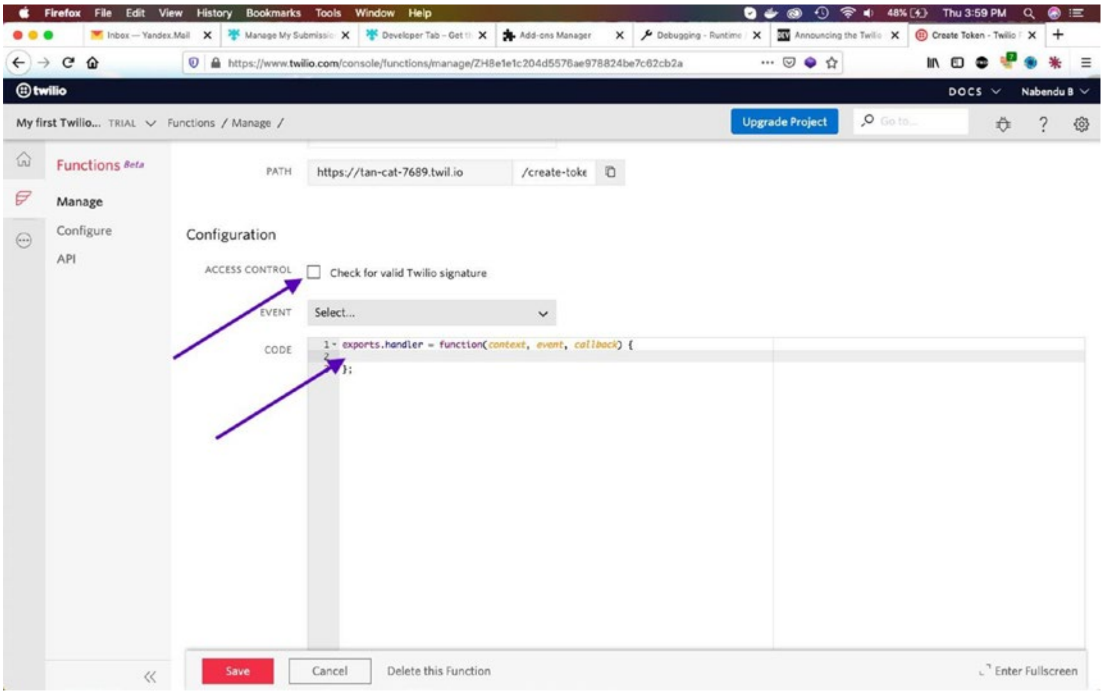
EVENT: Select...

CODE:

```
exports.handler = function(context, event, callback) {
    let twiml = new Twilio.twiml.VoiceResponse();
    // twiml.say("Hello World");
    callback(null, twiml);
};
```

Figure 7-24. *Create a token*

Next, we need to remove the check box and then remove everything inside the function (see Figure 7-25).



PATH https://tan-cat-7689.twil.io /create-toke

ACCESS CONTROL ☐ Check for valid Twilio signature

EVENT Select...

CODE

```
exports.handler = function(context, event, callback) {

};
```

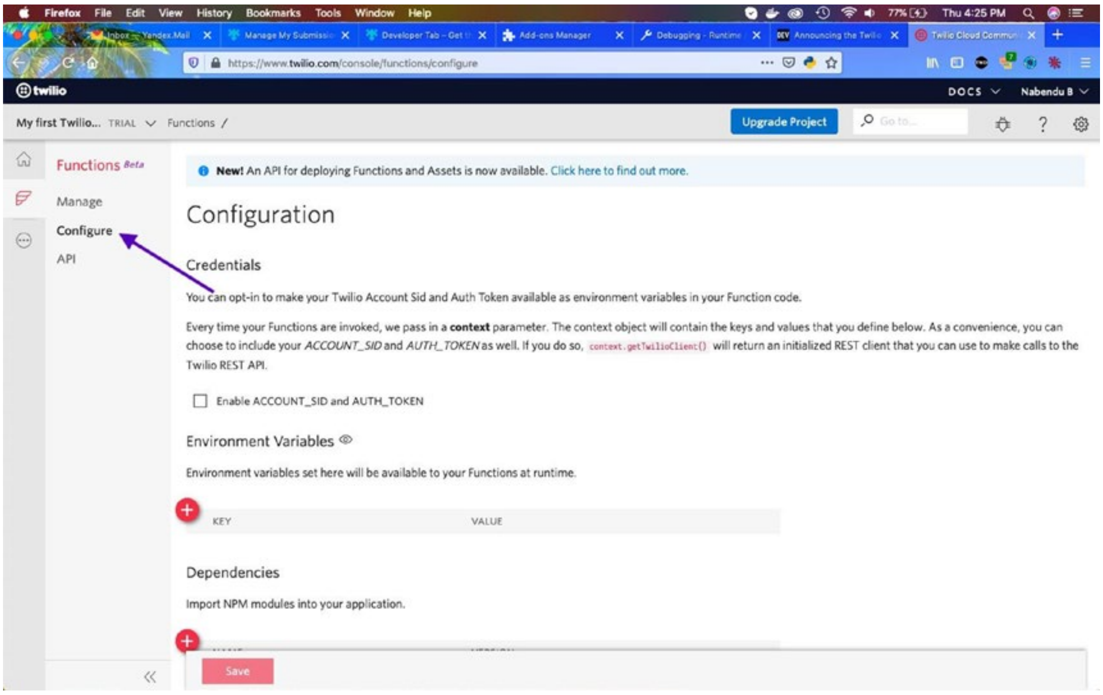
Figure 7-25. *Remove everything inside the function*

Next, we will write some code in the function. This code will use built-in Twilio variables and get a new access token from our stored variables. The code is shown in Listing 7-5.

Listing 7-5. Twilio Functions

```
exports.handler = function(context, event, callback) {
    let accessToken = Twilio.jwt.AccessToken;
    let videoGrant = accessToken.VideoGrant;
    let token = new accessToken(process.env.ACCOUNT_SID, process.env.API_
    KEY, process.env.API_SECRET);
};
```

After writing these three lines of code, click the Save button. After that, click the Configure link in the left menu. It will open the page shown in Figure 7-26.



New! An API for deploying Functions and Assets is now available. Click here to find out more.

Configuration

Credentials

You can opt-in to make your Twilio Account Sid and Auth Token available as environment variables in your Function code.

Every time your Functions are invoked, we pass in a **context** parameter. The context object will contain the keys and values that you define below. As a convenience, you can choose to include your *ACCOUNT_SID* and *AUTH_TOKEN* as well. If you do so, `context.getTwilioClient()` will return an initialized REST client that you can use to make calls to the Twilio REST API.

☐ Enable ACCOUNT_SID and AUTH_TOKEN

Environment Variables

Environment variables set here will be available to your Functions at runtime.

KEY	VALUE

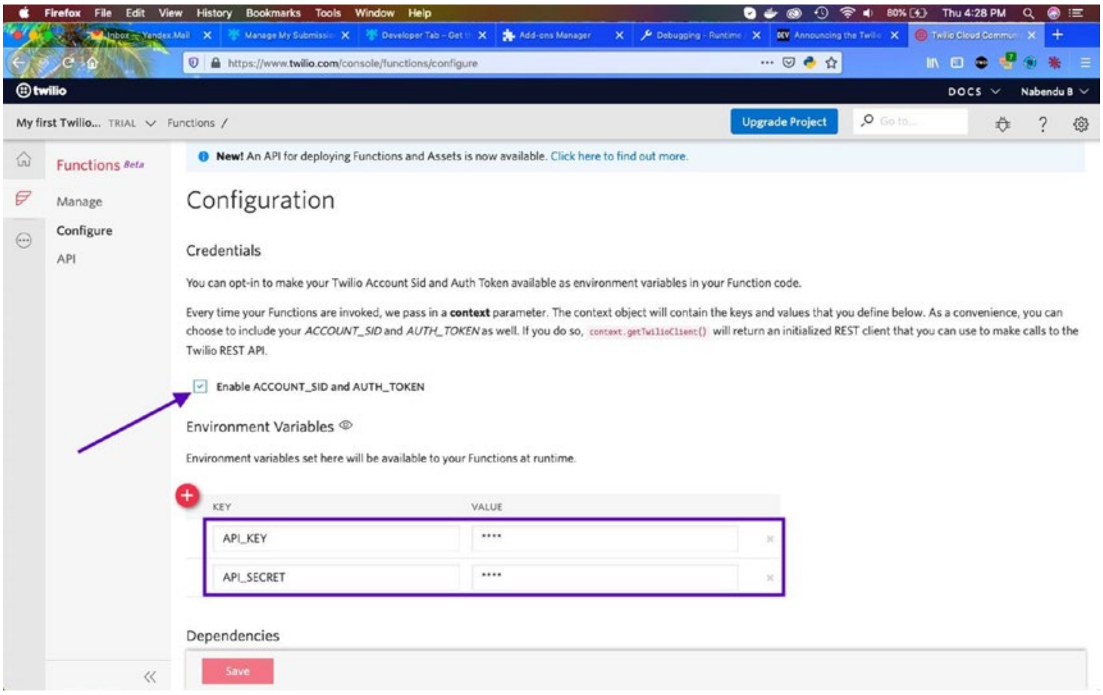
Dependencies

Import NPM modules into your application.

Figure 7-26. Configuration page

We need to check the Enable `ACCOUNT_SID` and `AUTH_TOKEN` check box. After that, click the + button next to KEY twice and enter `API_KEY` and `API_SECRET` in the Key column.

You get their values from `TWILIO_API_SID` and `TWILIO_API_SECRET`, which will be saved in the previous `.env` file (see Figure 7-27).


Functions Beta
Manage
Configure
API
New! An API for deploying Functions and Assets is now available. Click here to find out more.
Upgrade Project
Save

Configuration

Credentials

You can opt-in to make your Twilio Account Sid and Auth Token available as environment variables in your Function code.

Every time your Functions are invoked, we pass in a **context** parameter. The context object will contain the keys and values that you define below. As a convenience, you can choose to include your *ACCOUNT_SID* and *AUTH_TOKEN* as well. If you do so, `context.getTwilioClient()` will return an initialized REST client that you can use to make calls to the Twilio REST API.

☑ Enable ACCOUNT_SID and AUTH_TOKEN

Environment Variables

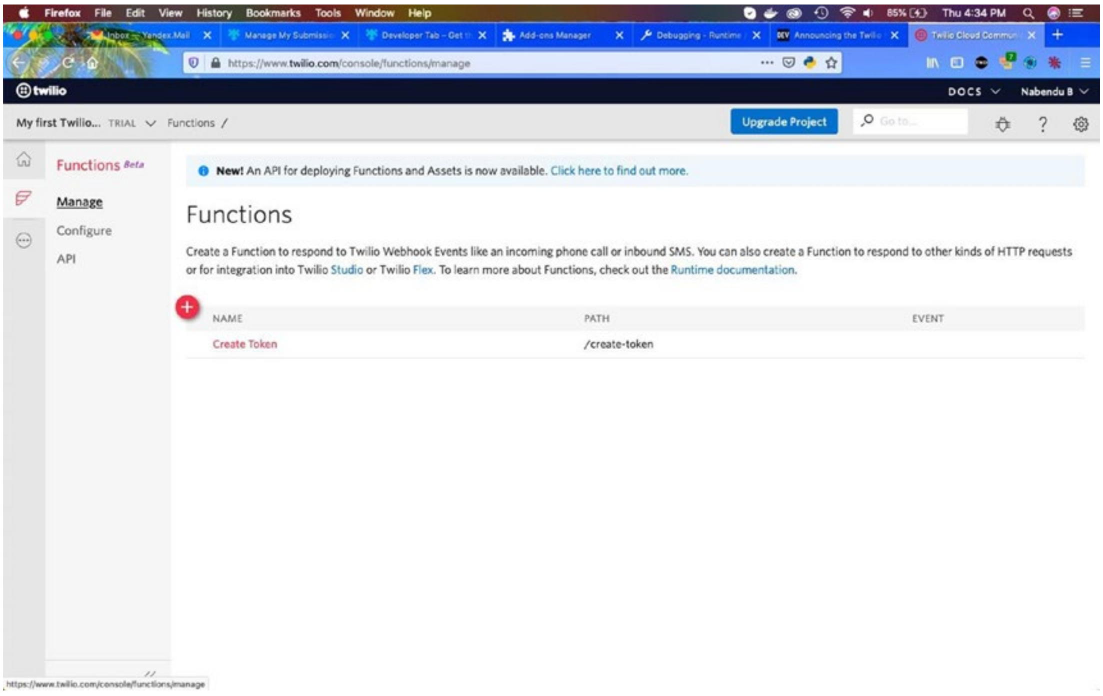
Environment variables set here will be available to your Functions at runtime.

KEY	VALUE
API_KEY	****
API_SECRET	****

Dependencies

Figure 7-27. *Enable the API keys*

After clicking the Save button, click the Manage link (see Figure 7-28).



New! An API for deploying Functions and Assets is now available. Click here to find out more.

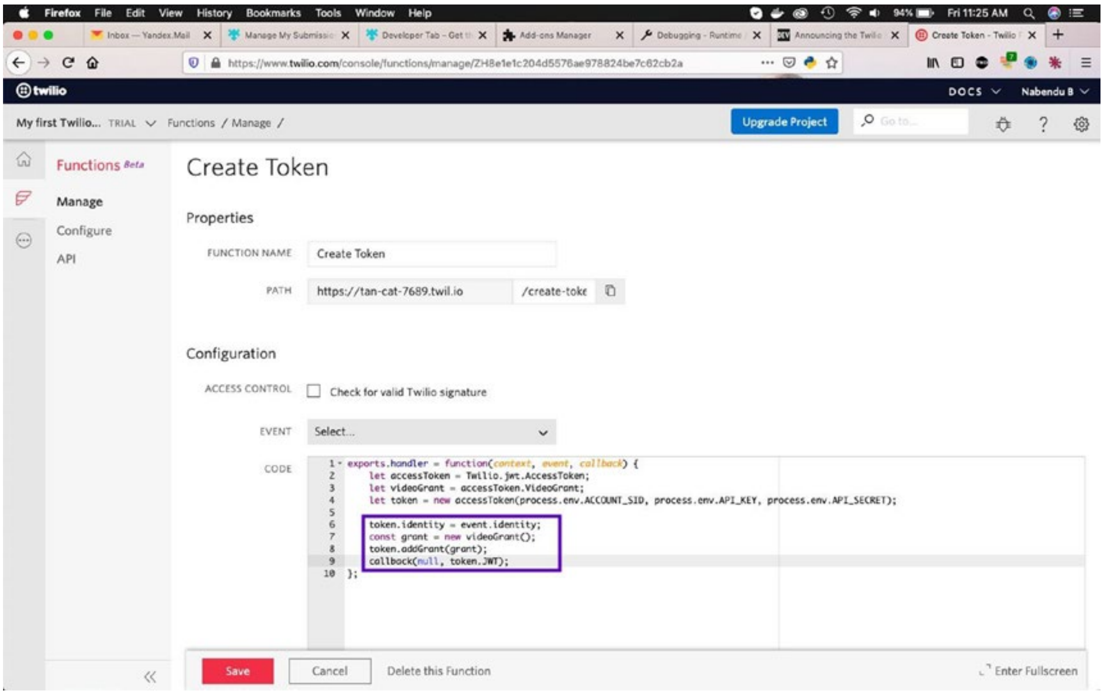
Functions

Create a Function to respond to Twilio Webhook Events like an incoming phone call or inbound SMS. You can also create a Function to respond to other kinds of HTTP requests or for integration into Twilio Studio or Twilio Flex. To learn more about Functions, check out the Runtime documentation.

NAME	PATH	EVENT
Create Token	/create-token	

Figure 7-28. *Choose the Manage option*

We need to click the function name (such as Create Token) to go to the edit page. Next, we add the four lines in Figure 7-29 to the function. We are doing a callback with the token with JWT.



Create Token

Properties

FUNCTION NAME: Create Token

PATH: https://tan-cat-7689.twil.io /create-toke

Configuration

ACCESS CONTROL: Check for valid Twilio signature

EVENT: Select...

CODE

```
exports.handler = function(context, event, callback) {
    let accessToken = Twilio.jwt.AccessToken;
    let videoGrant = accessToken.VideoGrant;
    let token = new accessToken(process.env.ACCOUNT_SID, process.env.API_KEY, process.env.API_SECRET);

    token.identity = event.identity;
    const grant = new videoGrant();
    token.addGrant(grant);
    callback(null, token.JWT);
};
```

Figure 7-29. *Configuring the function*

Adding the Code

We will do some coding now, but first start the project by moving to the directory and running the `gatsby develop` command.

Basic Setup

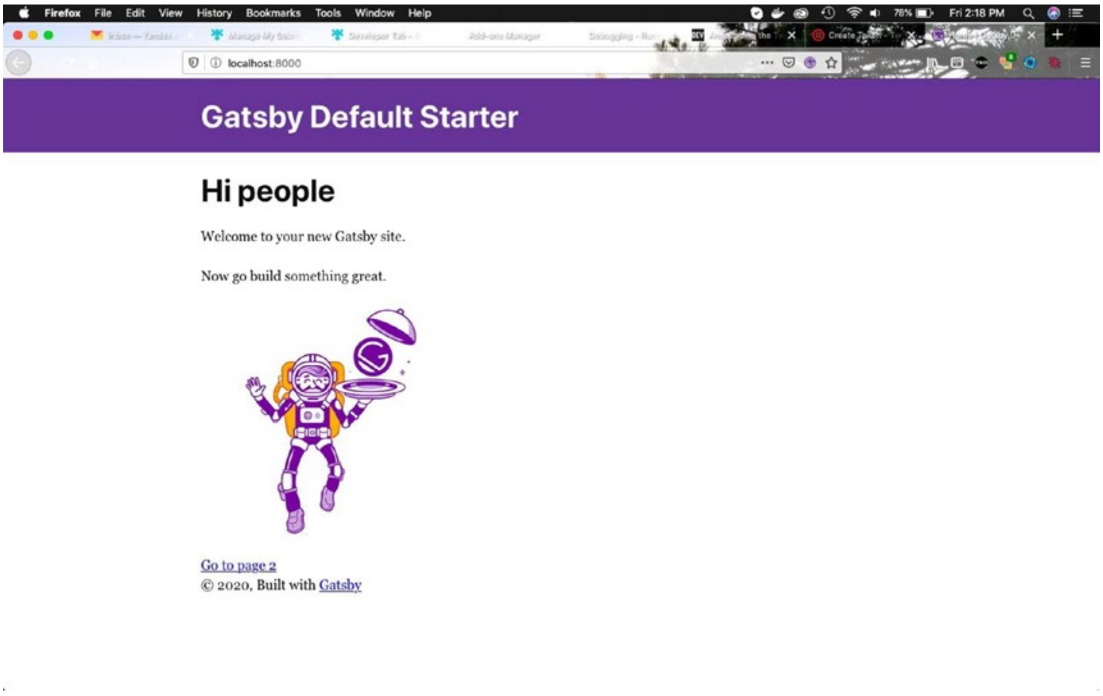
The command to start the server is shown in Listing 7-6.

Listing 7-6. Server Start Command

```
cd SimpleVideoApp
gatsby develop
```

If we go to `http://localhost:8000/`,[3] we will get the default page (see Figure 7-30).

[3]`http://localhost:8000/`



Gatsby Default Starter

Hi people

Welcome to your new Gatsby site.

Now go build something great.

Go to page 2
© 2020, Built with Gatsby

Figure 7-30. *Default starter*

Time to change the default starter, so open the `index.js` file and add the code in Listing 7-7.

Listing 7-7. The index.js File

```
import React from "react"
import Layout from "../components/layout"
import SEO from "../components/seo"

const IndexPage = () => {
  return (
    <Layout>
        <SEO title="Home" />
   </Layout>
  )
}

export default IndexPage
```

We will remove all the unnecessary things. Remove the page-2.js file, as we don’t need it.

It’s also time to change some things in this Gatsby starter. Open the layout.js file in the components folder and change the footer text. The updated code is marked in bold in Listing 7-8.

Listing 7-8. The layout.js File

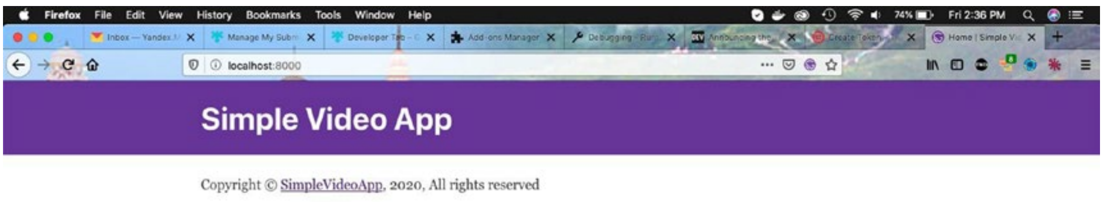
```
...
...
return (

      <Header siteTitle={data.site.siteMetadata.title} />
      <div
        style={{
          margin: `0 auto`,
          maxWidth: 960,
          padding: `0 1.0875rem 1.45rem`,
        }}
      >
        <main>{children}</main>
        <footer>
          Copyright © <a href="https://thewebdev.tech”>SimpleVideoApp</a>,
          {new Date().getFullYear()},
            All rights reserved
        </footer>
      </div>

  )
...
...
```

Also, let’s change the site’s title. Head over to the gatsby-config.js file and change it. The updated code is marked in bold in Listing 7-9.

Listing 7-9. Site Metadata Change in gatsby-config.js

```
module.exports = {
  siteMetadata: {
    title: `Simple Video App`,
    description: `A simple video app, created using gatsby and twilio,
    for twilio hackathon on dev.to`,
    author: `Nabendu Biswas`,
  },
  plugins: [
                        ...
                    ...
  ],
}
```

When we go to `http://localhost:8000/`,[4] we will see the updated app (see Figure 7-31).



Simple Video App

Copyright © SimpleVideoApp, 2020, All rights reserved

Figure 7-31. *Updated app*

Create a Login Form

Let's create a simple functional login form. Create a file called `login-form.js` inside the `components` folder and put the code from Listing 7-10 in it. It is a simple form with a text field that allows you to enter the name and a button to submit the form.

[4]`http://localhost:8000/`

Listing 7-10. The login-form.js File

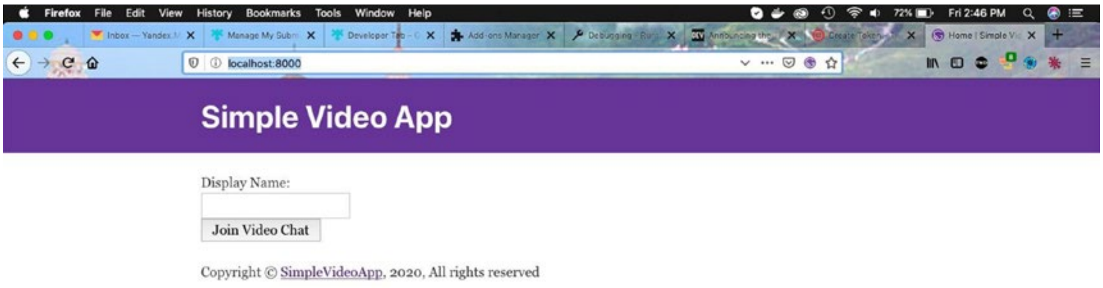
```
import React, { useState } from "react"

const LoginForm = () => {
    const [name, setName] = useState("")

    return (
            <form>
                    <label htmlFor="name">
                      Display Name: <br />
                        <input
                        type="text"
                        id="name"
                        name="name"
                        value={name}
                        onChange={e => setName(e.target.value)}
                        />
                    </label>
                 <br />
                <button type="submit">Join Video Chat</button>
            </form>
    )
}

export default LoginForm
```

Next, let's show this component in the `index.js` file. Import it and use it. The updated code is marked in bold in Listing 7-11.

Listing 7-11. LoginForm in index.js

```
import React from "react"
import Layout from "../components/layout"
import SEO from "../components/seo"
import LoginForm from "../components/login-form"

const IndexPage = () => {
  return (
    <Layout>
              <SEO title="Home" />
                <LoginForm />
   </Layout>
  )
}

export default IndexPage
```

The web app now shows our not-so-beautiful login form (see Figure 7-32).



Display Name:

Copyright © SimpleVideoApp, 2020, All rights reserved

Figure 7-32. *Simple login form*

Let's style this web app a bit, as I like web apps to look nice. First update all the primary styles in `layout.css`. Remove all the content and replace it with Listing 7-12.

Listing 7-12. The New layout.css File

```
@import url("https://fonts.googleapis.com/css?family=Quicksand&display=swap");

* {
    box-sizing: border-box;
    margin: 0;
}

:root {
    --primaryColor:#243e36;
    --mainGrey: #F9F9FA;
    --mainWhite: #fff;
    --mainBlack: #0A0A0A;
    --darkGrey: #8e8e8e;
    --mainTransition: all 0.3s linear;
    --mainSpacing: 4px;
}

body {
    font-family: "Quicksand", sans-serif;
    background: var(--mainGrey);
    color: var(--mainBlack);
    font-size: 18px;
    overflow-x: hidden;
}

footer {
  margin-top: auto;
  padding: 2rem;
  text-align: center;
  color: var(--mainBlack);
}
```

Next, let's update the `header.js` file to use a new color scheme. I had updated the background and also removed the line `margin: 0 auto` from `maxWidth: 960`. The updated code is marked in bold in Listing 7-13.

Listing 7-13. The Updated header.js File

```
import { Link } from "gatsby"
import PropTypes from "prop-types"
import React from "react"

const Header = ({ siteTitle }) => (
  <header
    style={{
      background: `#243e36`,
      marginBottom: `1.45rem`,
    }}
  >
```

```
    <div
      style={{
        maxWidth: 960,
        padding: `1.45rem 1.0875rem`,
      }}
    >
      <h1 style={{ margin: 0 }}>
            ...
            ...
      </h1>
    </div>
  </header>
)
...
...

export default Header
```

Next, create a file called `login.module.css` in the `components` folder and put the code in Listing 7-14 into it.

Listing 7-14. The login.module.css File

```
.contact {
    padding: 4rem 0;
}
.center {
    width: 80vw;
    margin: 0 auto;
}
@media screen and (min-width: 992px) {
    .center {
        width: 50vw;
        margin: 0 auto;
    }
}
```

```
.contact label {
    text-transform: capitalize;
    display: block;
    margin-bottom: 0.5rem;
}
.contact h3 {
    text-transform: uppercase;
    font-size: 2rem;
    text-align: center;
    letter-spacing: 7px;
    color: var(--mainBlack);
    margin-bottom: 2rem;
}
.formControl,
.submit {
    width: 100%;
    font-size: 1rem;
    margin-bottom: 1rem;
    padding: 0.375rem 0.75rem;
    border: 1px solid var(--darkGrey);
    border-radius: 0.25rem;
}
.submit {
    background-color: var(--primaryColor);
    border-color: var(--primaryColor);
    text-transform: uppercase;
    color: var(--mainWhite);
    transition: var(--mainTransition);
    cursor: pointer;
}
.submit:hover {
    background: var(--darkGrey);
    color: var(--mainBlack);
    border-color: var(--darkGrey);
}
```

Update the `login-form.js` file to include these styles. The updated code is marked in bold in Listing 7-15.

Listing 7-15. Styles in login-form.js

```
import React, { useState } from "react"
import styles from "./login.module.css"

const LoginForm = () => {
    const [name, setName] = useState("")

    return (
        <section className={styles.contact}>
            <h3>Login</h3>
            <div className={styles.center}>
            <form>
                <div>
                    <label htmlFor="name">Display Name</label>
                    <input
                        type="text"
                        id="name"
                        name="name"
                        value={name}
                        className={styles.formControl}
                        onChange={e => setName(e.target.value)}
                    />
                </div>
                <button type="submit" className={styles.submit}>Join Video
                Chat</button>
            </form>
            </div>
        </section>
    )
}

export default LoginForm
```

The web app now looks perfect, as shown in Figure 7-33.

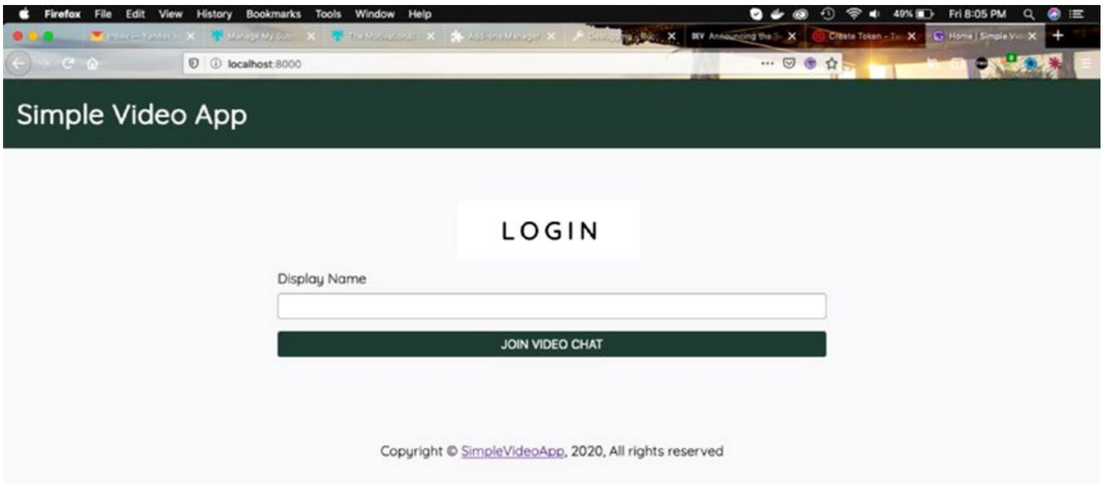


Figure 7-33. *New and improved web app*

Connect the App to Twilio

We need to do an API call to the Twilio endpoint, which contains our function. For this we will install Axios first.

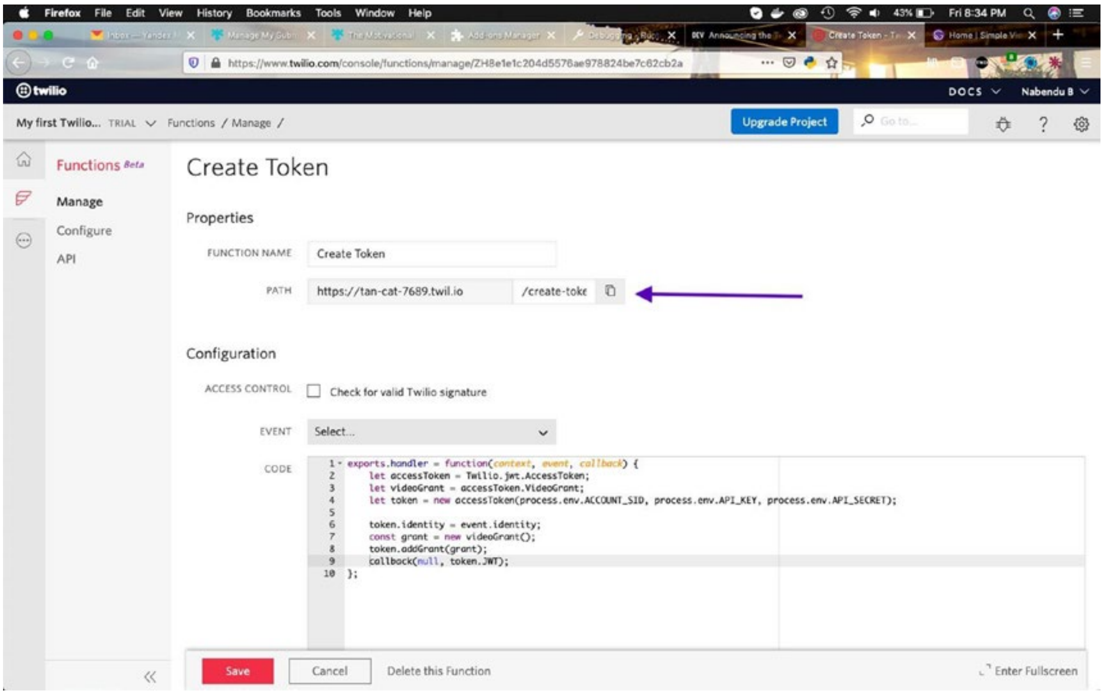
Stop your `gatsby develop` and install Axios using the `npm i axios` command. Don't forget to restart your development server by re-running the `gatsby develop` command.

We will now use Axios to send the form data to our Twilio endpoint. Open the `login-form.js` file and create a form called `onSubmit`. It will call a function called `handleSubmit` and use the Twilio endpoint URL, and the data will be the `name`. The updated code is marked in bold in Listing 7-16.

Listing 7-16. Axios in login-form.js

```
import React, { useState } from "react"
import styles from "./login.module.css"
import axios from "axios"

const LoginForm = () => {
    const [name, setName] = useState("")
    const handleSubmit = async event => {
        event.preventDefault()
        const result = await axios({
            method: "POST",
            url: "https://tan-cat-7689.twil.io/create-token",
            data: {
                identity: name,
            },
        })
        console.log(result);
    }

    return (
        <section className={styles.contact}>
            <h3>Login</h3>
            <div className={styles.center}>
            <form onSubmit={handleSubmit}>
                <div>
                                ...
                                ...
                </div>
                <button type="submit" className={styles.submit}>Join Video
                Chat</button>
            </form>
            </div>
        </section>
    )
}

export default LoginForm
```

We get the Twilio endpoint in the code from the function page in Twilio (see Figure 7-34).



FUNCTION NAME: Create Token

PATH: https://tan-cat-7689.twil.io /create-toke

ACCESS CONTROL: Check for valid Twilio signature

EVENT: Select...

CODE:

```
exports.handler = function(context, event, callback) {
    let accessToken = Twilio.jwt.AccessToken;
    let videoGrant = accessToken.VideoGrant;
    let token = new accessToken(process.env.ACCOUNT_SID, process.env.API_KEY, process.env.API_SECRET);

    token.identity = event.identity;
    const grant = new videoGrant();
    token.addGrant(grant);
    callback(null, token.JWT);
};
```

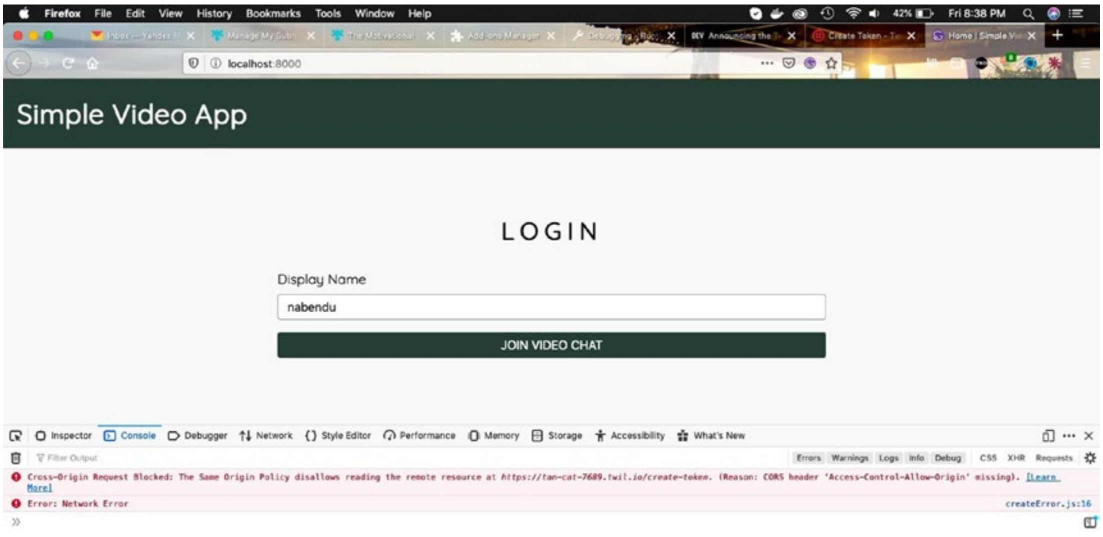
Figure 7-34. *The function page*

We also updated the code as per Listing 7-17.

Listing 7-17. Updated Twilio Function

```
exports.handler = function(context, event, callback) {
    let accessToken = Twilio.jwt.AccessToken;
    let videoGrant = accessToken.VideoGrant;
    let token = new accessToken(process.env.ACCOUNT_SID, process.env.API_
    KEY, process.env.API_SECRET);

    token.identity = event.identity;
    const grant = new videoGrant();
    token.addGrant(grant);
        callback(null, token.JWT);
};
```

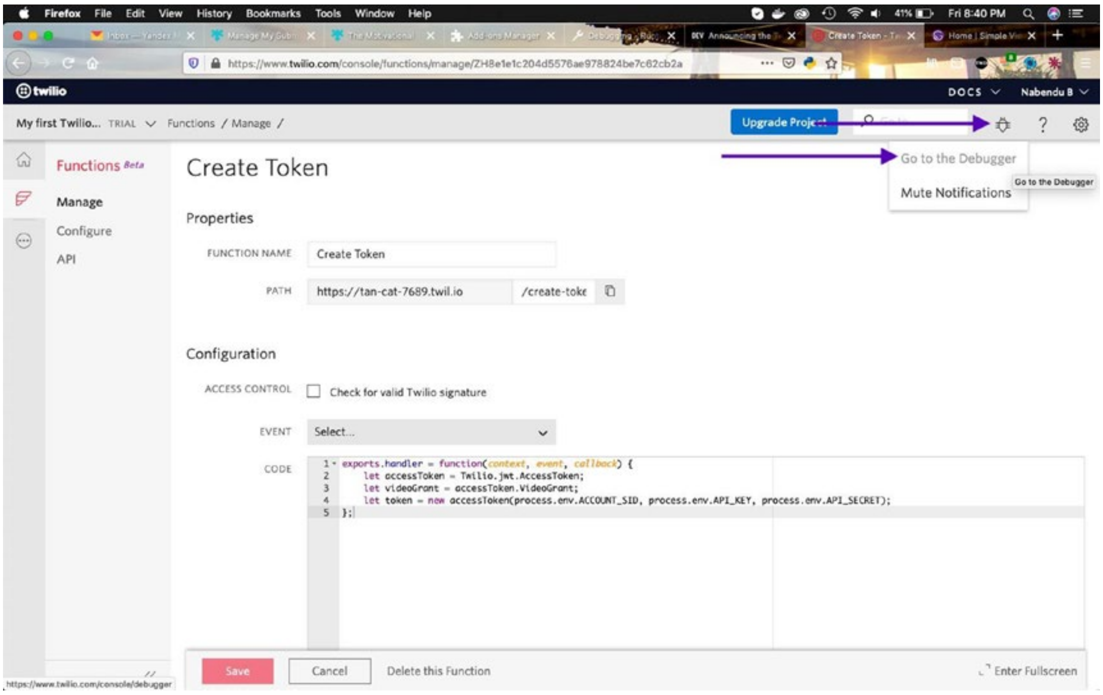
Go back to the web app and open the console. After that, provide a name and log in. We are getting a CORS error and are not able to log in (see Figure 7-35).



```
Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at https://tan-cat-7689.twil.io/create-token. (Reason: CORS header 'Access-Control-Allow-Origin' missing). [Learn More]
Error: Network Error                                                                  createError.js:16
```

Figure 7-35. *Logging in*

There is a very good way to check the actual error from inside the Twilio dashboard. Click the bug icon on the top-right side. After that, click the Go to the Debugger link (see Figure 7-36).



Create Token

Properties

FUNCTION NAME: Create Token

PATH: https://tan-cat-7689.twil.io /create-toke

Configuration

ACCESS CONTROL: Check for valid Twilio signature

EVENT: Select...

CODE:

```
exports.handler = function(context, event, callback) {
    let accessToken = Twilio.jwt.AccessToken;
    let videoGrant = accessToken.VideoGrant;
    let token = new accessToken(process.env.ACCOUNT_SID, process.env.API_KEY, process.env.API_SECRET);
};
```

Figure 7-36. *You can check an error from the Twilio dashboard*

The screen in Figure 7-37 will appear and will show the error. We need to click the highlighted error in this screen.

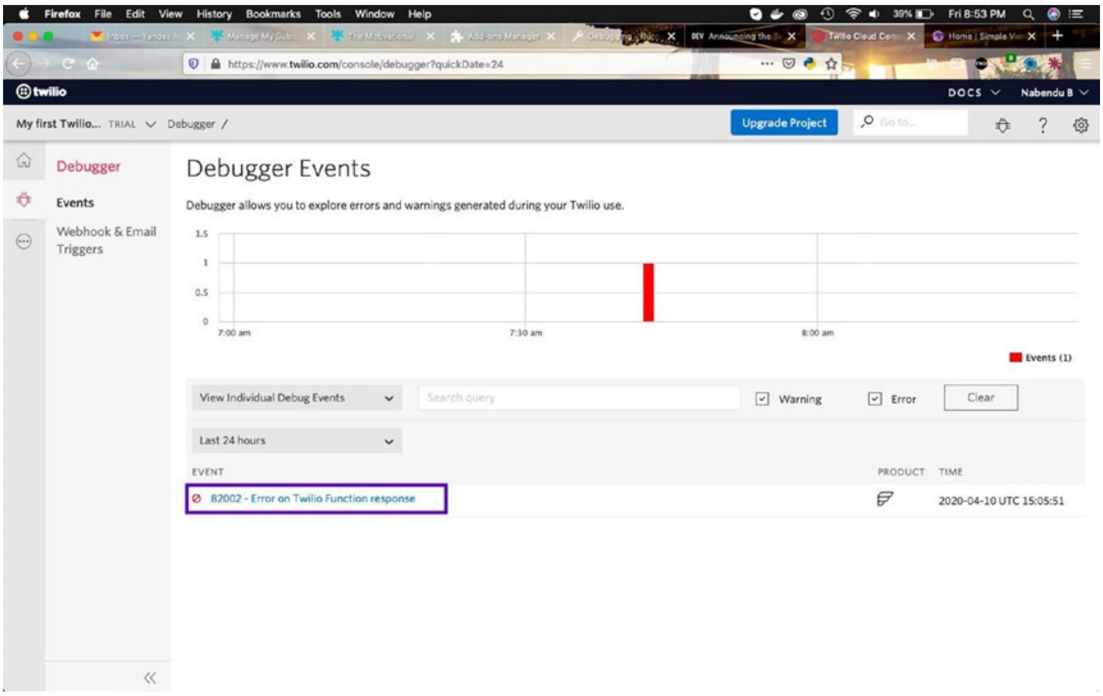
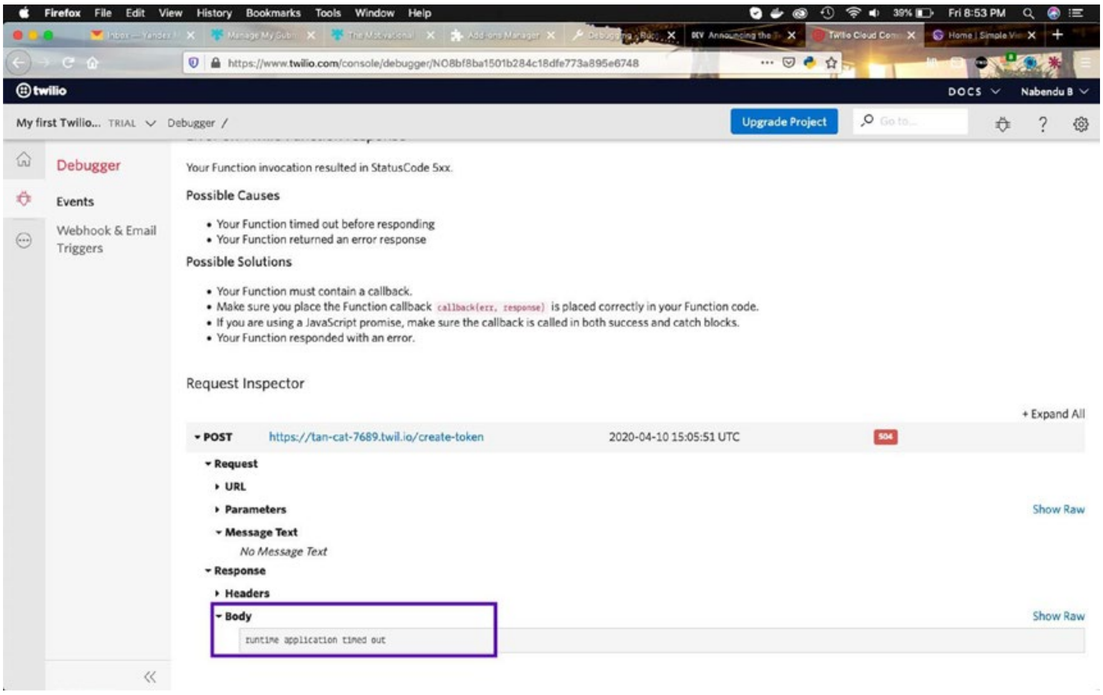

Firefox File Edit View History Bookmarks Tools Window Help
Fri 8:53 PM
https://www.twilio.com/console/debugger?quickDate=24
twilio
DOCS
Nabendu B
My first Twilio... TRIAL
Debugger /
Upgrade Project
Go to...
Debugger
Events
Webhook & Email Triggers
Debugger Events
Debugger allows you to explore errors and warnings generated during your Twilio use.
1.5
1
0.5
0
7:00 am
7:30 am
8:00 am
Events (1)
View Individual Debug Events
Search query
Warning
Error
Clear
Last 24 hours
EVENT
PRODUCT
TIME
82002 - Error on Twilio Function response
2020-04-10 UTC 15:05:51

Figure 7-37. *Click the highlighted error*

It will show us the real error (see Figure 7-38).


Firefox File Edit View History Bookmarks Tools Window Help
Fri 8:53 PM
https://www.twilio.com/console/debugger/NO8bf8ba1501b284c18dfe773a895e6748
twilio
DOCS
Nabendu B
My first Twilio... TRIAL
Debugger /
Upgrade Project
Go to...
Debugger
Events
Webhook & Email Triggers

Your Function invocation resulted in StatusCode 5xx.

Possible Causes

- Your Function timed out before responding
- Your Function returned an error response

Possible Solutions

- Your Function must contain a callback.
- Make sure you place the Function callback `callback(err, response)` is placed correctly in your Function code.
- If you are using a JavaScript promise, make sure the callback is called in both success and catch blocks.
- Your Function responded with an error.

Request Inspector

+ Expand All

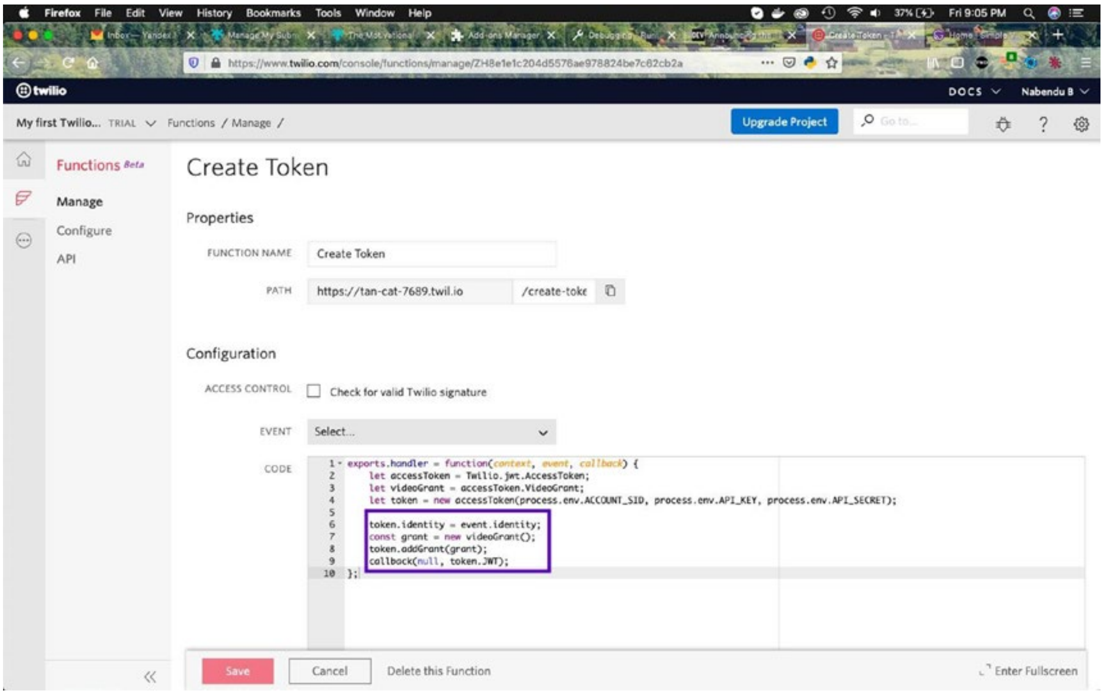
POST https://tan-cat-7689.twil.io/create-token 2020-04-10 15:05:51 UTC 504

Request
URL
Parameters Show Raw
Message Text
No Message Text
Response
Headers
Body Show Raw

```
runtime application timed out
```

Figure 7-38. *The real error*

After reviewing my function again, I realized that the four lines had not been saved from earlier. I added those lines again and clicked the Save button (see Figure 7-39).



Create Token

Properties

FUNCTION NAME: Create Token

PATH: https://tan-cat-7689.twil.io /create-toke

Configuration

ACCESS CONTROL: ☐ Check for valid Twilio signature

EVENT: Select...

CODE

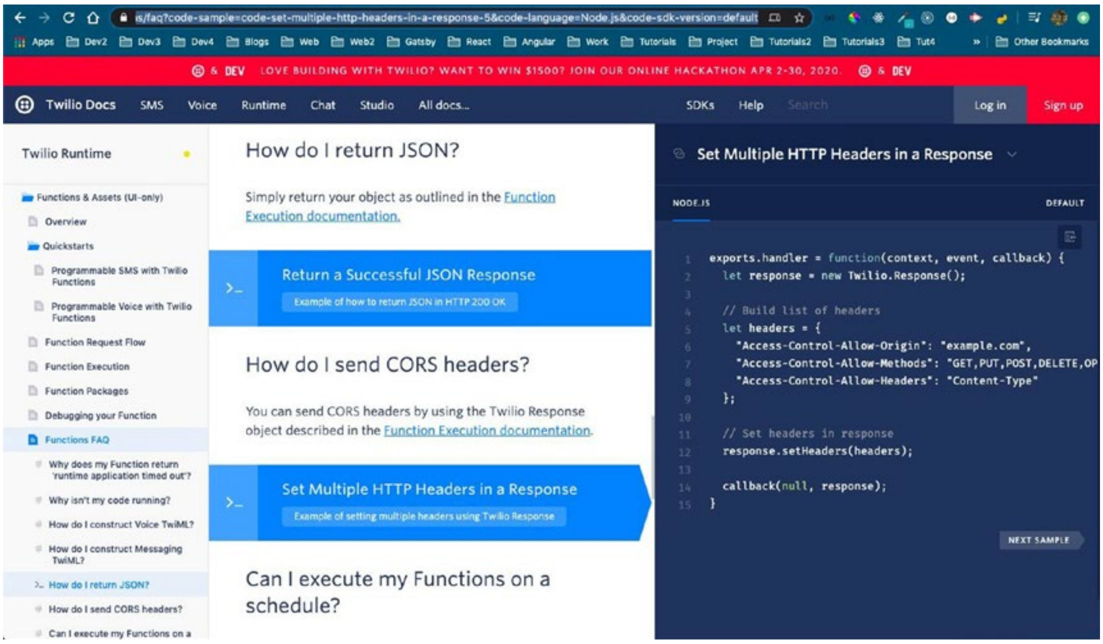
```
exports.handler = function(context, event, callback) {
    let accessToken = Twilio.jwt.AccessToken;
    let videoGrant = accessToken.VideoGrant;
    let token = new accessToken(process.env.ACCOUNT_SID, process.env.API_KEY, process.env.API_SECRET);

    token.identity = event.identity;
    const grant = new videoGrant();
    token.addGrant(grant);
    callback(null, token.JWT);
};
```

Figure 7-39. *Added lines, added again*

After logging in again, I get the same error. By watching the YouTube[5] video, I realize that we need to use the function as in the Twilio docs link[6] (see Figure 7-40).

[5]`https://www.youtube.com/watch?v=KO2SnxY6c_0`

[6]`https://www.twilio.com/docs/runtime/functions/faq?code-sample=code-set-multiple-http-headers-in-a-response-5&codelanguage=Node.js&code-sdk-version=default`


s/faq?code-sample=code-set-multiple-http-headers-in-a-response-5&code-language=Node.js&code-sdk-version=default
Apps Dev2 Dev3 Dev4 Blogs Web Web2 Gatsby React Angular Work Tutorials Project Tutorials2 Tutorials3 Tut4 Other Bookmarks
& DEV LOVE BUILDING WITH TWILIO? WANT TO WIN $1500? JOIN OUR ONLINE HACKATHON APR 2-30, 2020. & DEV
Twilio Docs SMS Voice Runtime Chat Studio All docs... SDKs Help Search Log in Sign up
Twilio Runtime
Functions & Assets (UI-only)
Overview
Quickstarts
Programmable SMS with Twilio Functions
Programmable Voice with Twilio Functions
Function Request Flow
Function Execution
Function Packages
Debugging your Function
Functions FAQ
Why does my Function return 'runtime application timed out'?
Why isn't my code running?
How do I construct Voice TwiML?
How do I construct Messaging TwiML?
How do I return JSON?
How do I send CORS headers?
Can I execute my Functions on a
NODE.JS
DEFAULT
NEXT SAMPLE

How do I return JSON?

Simply return your object as outlined in the Function Execution documentation.

Return a Successful JSON Response
Example of how to return JSON in HTTP 200 OK

How do I send CORS headers?

You can send CORS headers by using the Twilio Response object described in the Function Execution documentation.

Set Multiple HTTP Headers in a Response
Example of setting multiple headers using Twilio Response

Can I execute my Functions on a schedule?

Set Multiple HTTP Headers in a Response

```
exports.handler = function(context, event, callback) {
  let response = new Twilio.Response();

  // Build list of headers
  let headers = {
    "Access-Control-Allow-Origin": "example.com",
    "Access-Control-Allow-Methods": "GET,PUT,POST,DELETE,OP
    "Access-Control-Allow-Headers": "Content-Type"
  };

  // Set headers in response
  response.setHeaders(headers);

  callback(null, response);
}
```

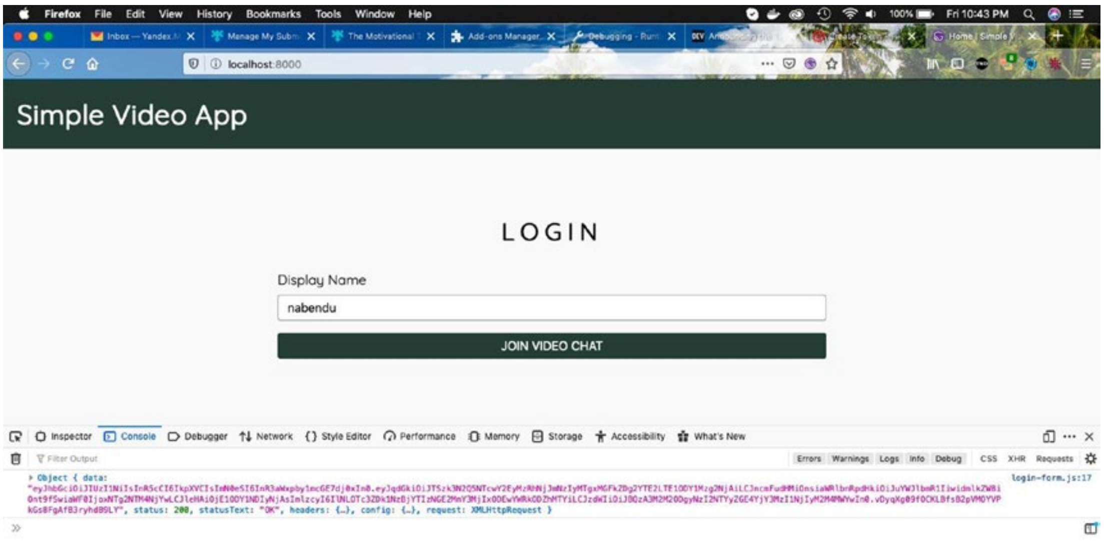
Figure 7-40. *Twilio docs*

Updating your function accordingly. Also, don't forget to click the Save button. The updated code is marked in bold in Listing 7-18.

Listing 7-18. Twilio Function Updated Again

```
exports.handler = function(context, event, callback) {
    let accessToken = Twilio.jwt.AccessToken;
    let videoGrant = accessToken.VideoGrant;
 let token = new accessToken(process.env.ACCOUNT_SID, process.env.API_KEY,
 process.env.API_SECRET);
    token.identity = event.identity;
    const grant = new videoGrant();
    token.addGrant(grant);

    let response = new Twilio.Response();

// Build list of headers
let headers = {
    "Access-Control-Allow-Origin": "*",
    "Access-Control-Allow-Methods": "GET,PUT,POST",
    "Access-Control-Allow-Headers": "Content-Type"
          };

              // Set headers in response
              response.setHeaders(headers);
              response.setBody(JSON.stringify(token.toJwt()));

             callback(null, response);
};
```

When we submit again, we will get the JWT back successfully (see Figure 7-41).



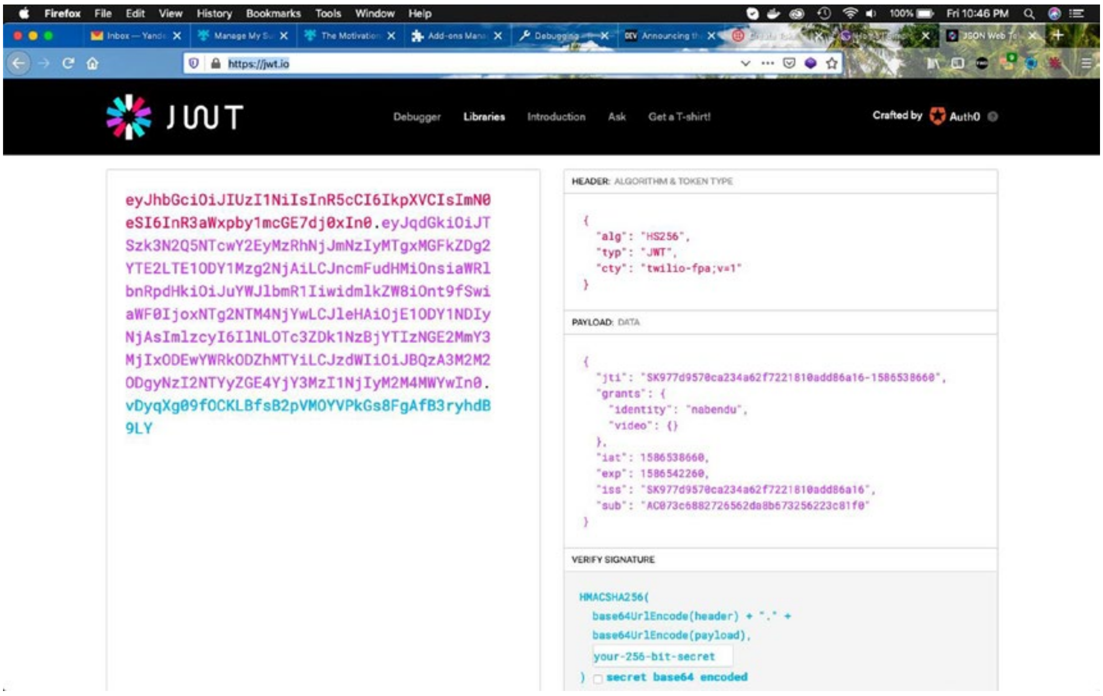
Display Name: nabendu

```
▸ Object { data:                                                                                                   login-form.js:17
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCIsImN0eSI6InR3aWxpby1mcGE7dj0xIn0.eyJqdGkiOiJTSzk3N2Q5NTcwY2EyMzRhNjJmNzIyMTgxMGFkZDg2YTE2LTE1ODY1Mzg2NjAiLCJncmFudHMiOnsiaWRlbnRpdHkiOiJuYWJlbmR1IiwidmlkZW8i
Ont9fSwiaWF0IjoxNTg2NTM4NjYwLCJleHAiOjE1ODY1NDIyNjAsImlzcyI6IlNLOTc3ZDk1NzBjYTIzNGE2MmY3MjIxODEwYWRkODZhMTYiLCJzdWIiOiJBQzA3M2M2ODgyNzI2NTYyZGE4YjY3MzI1NjIyM2M4MWYwIn0.vDyqXg09fOCKLBfsB2pVMOYVP
kGs8FgAfB3ryhdB9LY", status: 200, statusText: "OK", headers: {…}, config: {…}, request: XMLHttpRequest }
```

Figure 7-41. *JWT back*

We can also check the validity of it by going to the site at `https://jwt.io/`[7] and pasting the returned JWT there. And, yes, it is valid (see Figure 7-42).

[7] `https://jwt.io/`



```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCIsImN0
eSI6InR3aWxpby1mcGE7dj0xIn0.eyJqdGkiOiJT
Szk3N2Q5NTcwY2EyMzRhNjJmNzIyMTgxMGFkZDg2
YTE2LTE1ODY1Mzg2NjAiLCJncmFudHMiOnsiaWRl
bnRpdHkiOiJuYWJlbmR1IiwidmlkZW8iOnt9fSwi
aWF0IjoxNTg2NTM4NjYwLCJleHAiOjE1ODY1NDIy
NjAsImlzcyI6IlNLOTc3ZDk1NzBjYTIzNGE2MmY3
MjIxODEwYWRkODZhMTYiLCJzdWIiOiJBQzA3M2M2
ODgyNzI2NTYyZGE4YjY3MzI1NjIyM2M4MWYwIn0.
vDyqXg09fOCKLBfsB2pVMOYVPkGs8FgAfB3ryhdB
9LY
```

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT",
  "cty": "twilio-fpa;v=1"
}
```

PAYLOAD: DATA

```
{
  "jti": "SK977d9570ca234a62f7221810add86a16-1586538660",
  "grants": {
    "identity": "nabendu",
    "video": {}
  },
  "iat": 1586538660,
  "exp": 1586542260,
  "iss": "SK977d9570ca234a62f7221810add86a16",
  "sub": "AC073c6882726562da8b673256223c81f0"
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

Figure 7-42. *Valid JWT*

Implementing the Video

It's time to store the result so that we can use it in the next section. Update the `login-form.js` file with a props `storeToken` and then save the JWT in it. The updated code is marked in bold in Listing 7-19.

Listing 7-19. JWT in login-form.js

```
const LoginForm = ({ storeToken }) => {
    const [name, setName] = useState("")

    const handleSubmit = async event => {
        event.preventDefault()
        const result = await axios({
            method: "POST",
            url: "https://tan-cat-7689.twil.io/create-token",
            data: {
                identity: name,
            },
        })
        console.log(result);
        const jwt = result.data;
        storeToken(jwt);
}
```

Next, we will update `index.js` with a new state token and pass the props `storeToken` in `LoginForm`.

We are using ternary logic to display the form if the user is not logged in or for the time being has token text. The updated code is marked in bold in Listing 7-20.

Listing 7-20. storeToken in index.js

```
import React, {useState} from "react"
import Layout from "../components/layout"
import SEO from "../components/seo"
import LoginForm from "../components/login-form";

const IndexPage = () => {
  const [token, setToken] = useState(false);

  return (
    <Layout>
      <SEO title="Home" />
      {!token ? <LoginForm storeToken={setToken} /> : <p>Has Token</p>}
    </Layout>
  )
}

export default IndexPage
```

It's time to test our code in `http://localhost:8000/`[8]. Upon opening it, we get the login form (see Figure 7-43).

[8] `http://localhost:8000/`

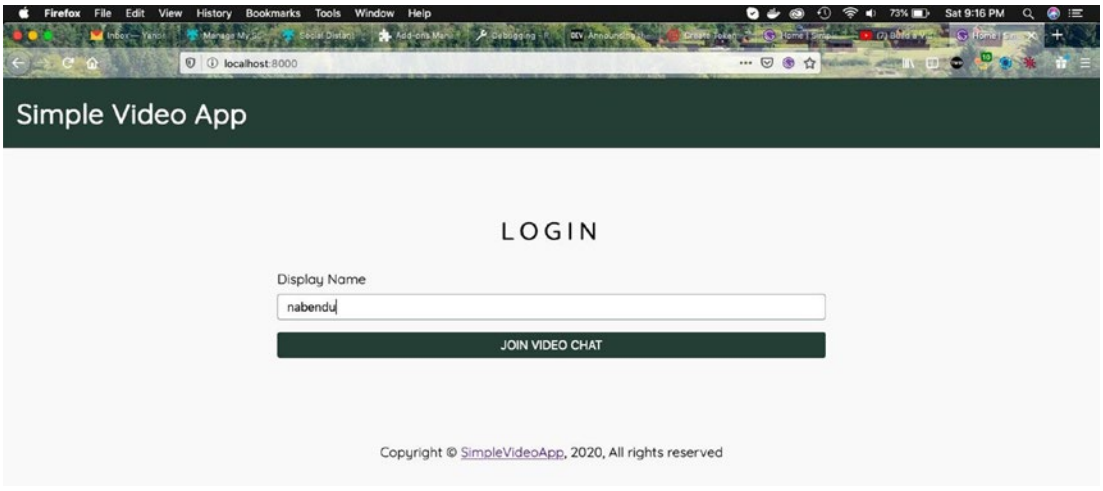


Figure 7-43. *Login form*

When we provide a name and click Join Video Chat, we are taken to the screen in Figure 7-44.

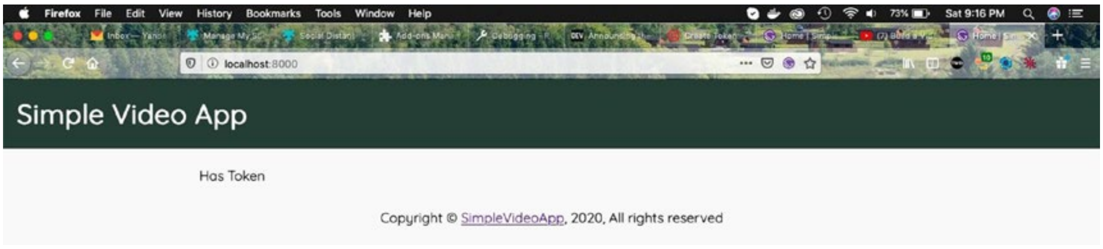


Figure 7-44. *It was successful*

Create the Video Component

Next, we will create the Video component. Create a new file called `video.js` inside the `components` folder. We are using a `useEffect` hook, which will fire when the token changes. The token comes as a prop and is used to connect to a room, using the `twilio-video` built-in method. The code is shown in Listing 7-21.

Listing 7-21. The video.js File

```
import React, { useEffect } from 'react'
import TwilioVideo from "twilio-video"

const Video = ({ token }) => {
    useEffect(() => {
        TwilioVideo.connect(token, { video: true, audio: true, name: "SVA"
        }).then(
            result => {
                        console.log("Successfully joined room", result)
            })
    }, [token])

    return (
        <div>
             Video
        </div>
    )
}

export default Video;
```

Next, let's use this component in `index.js` when the user is authenticated. The updated code is shown in Listing 7-22.

Listing 7-22. Video in index.js

```
import React, {useState} from "react"
import Layout from "../components/layout"
import SEO from "../components/seo"
import LoginForm from "../components/login-form";
import Video from "../components/video";

const IndexPage = () => {
  const [token, setToken] = useState(false);
  return (
    <Layout>
      <SEO title="Home" />
      {!token ? <LoginForm storeToken={setToken} /> : <Video
      token={token}  />}
    </Layout>
  )
}

export default IndexPage
```

We will again log in to our web app, by giving a display name and clicking the Join **Video Chat** button (see Figure 7-45).

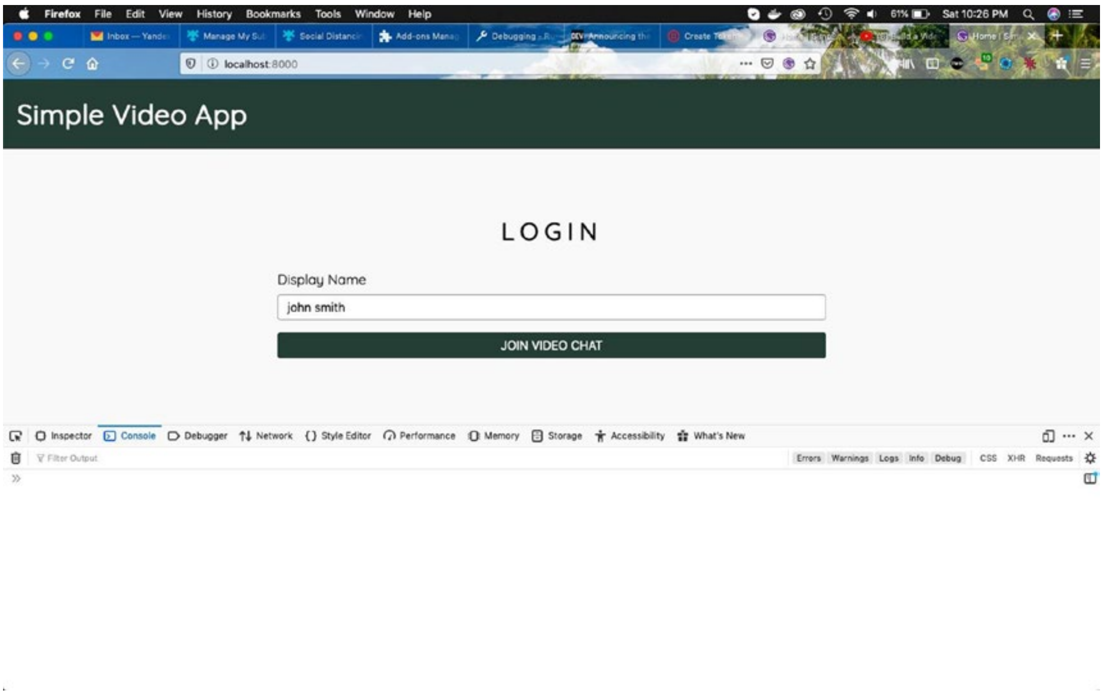
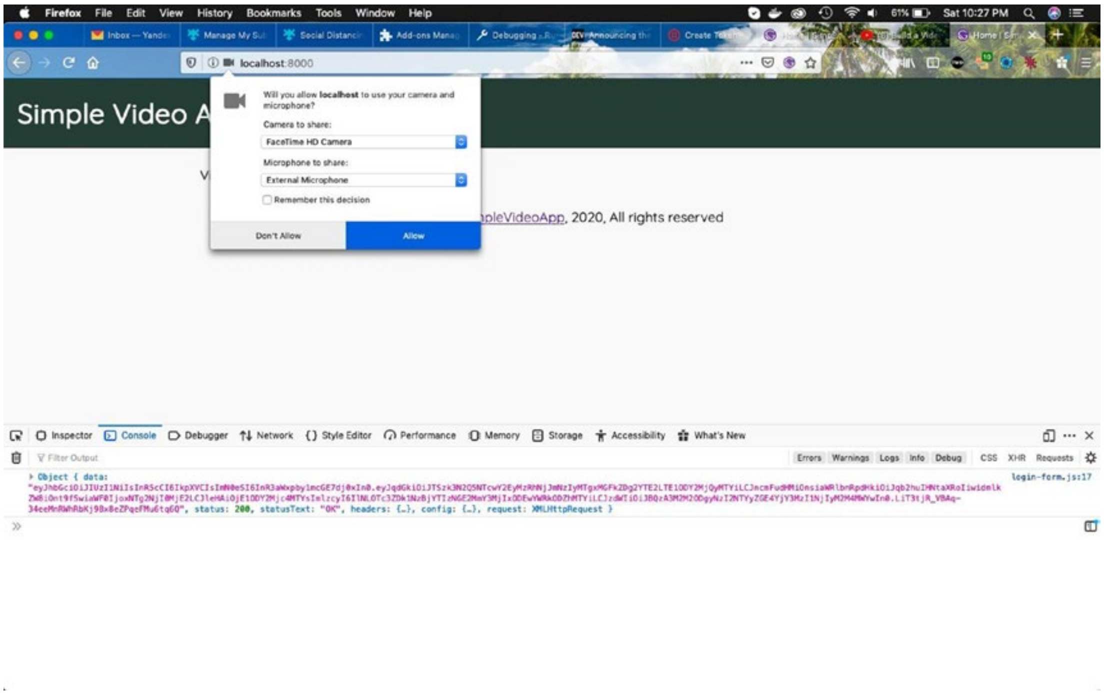


Figure 7-45. *Log in*

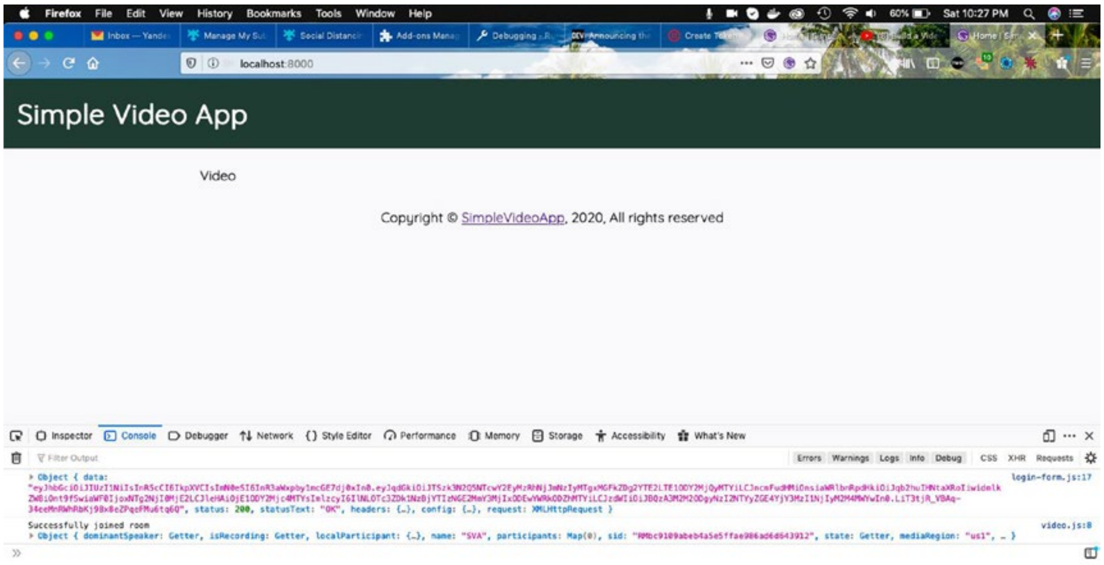
This will open a popup to ask for permission to use video and audio. We need to click Allow (see Figure 7-46).



```
▸ Object { data:
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCIsImN0eSI6InR3aWxpby1mcGE7djBxIn0.eyJqdGkiOiJTSzk3N2Q5NTcwY2EyMzRhNjJmNzIyMTgxMGFkZDg2YTE2LTE1ODY2MjQyMTYiLCJncmFudHMiOnsiaWRlbnRpdHkiOiJqb2huIHNtaXRoIiwidmlkZW8iOnt9fSwiaWF0IjoxNTg2NjI0MjE2LCJleHAiOjE1ODY2Mjc4MTYsImlzcyI6IlNLOTc3ZDk1NzBjYTIzNGE2MmY3MjIxODEwYWRkODZhMTYiLCJzdWIiOiJBQzA3M2M2ODgyNzI2NTYyZGE4YjY3MzI1NjIyM2M4MWYwIn0.LiT3tjR_VBAq-34eeMnRWhRbKj98x8eZPqeFMu6tq6Q", status: 200, statusText: "OK", headers: {…}, config: {…}, request: XMLHttpRequest }
login-form.js:17
```

Figure 7-46. *Choose Allow to continue*

It will successfully log you in and show Video (see Figure 7-47).



```
▸ Object { data:
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCIsImN0eSI6InR3aWxpby1mcGE7djBxIn0.eyJqdGkiOiJTSzk3N2Q5NTcwY2EyMzRhNjJmNzIyMTgxMGFkZDg2YTE2LTE1ODY2MjQyMTYiLCJncmFudHMiOnsiaWRlbnRpdHkiOiJqb2huIHNtaXRoIiwidmlkZW8iOnt9fSwiaWF0IjoxNTg2NjI0MjE2LCJleHAiOjE1ODY2Mjc4MTYsImlzcyI6IlNLOTc3ZDk1NzBjYTIzNGE2MmY3MjIxODEwYWRkODZhMTYiLCJzdWIiOiJBQzA3M2M2ODgyNzI2NTYyZGE4YjY3MzI1NjIyM2M4MWYwIn0.LiT3tjR_VBAq-34eeMnRWhRbKj98x8eZPqeFMu6tq6Q", status: 200, statusText: "OK", headers: {…}, config: {…}, request: XMLHttpRequest }
login-form.js:17
Successfully joined room
▸ Object { dominantSpeaker: Getter, isRecording: Getter, localParticipant: {…}, name: "SVA", participants: Map(0), sid: "RMbc9109abeb4a5e5ffae986ad6d643912", state: Getter, mediaRegion: "us1", … }
video.js:8
```

Figure 7-47. *Video*

We will now start showing the local webcam video. Open the `video.js` file and update the highlighted parts. Here, we are using `useRef`, as we want to attach the video to the `div`.

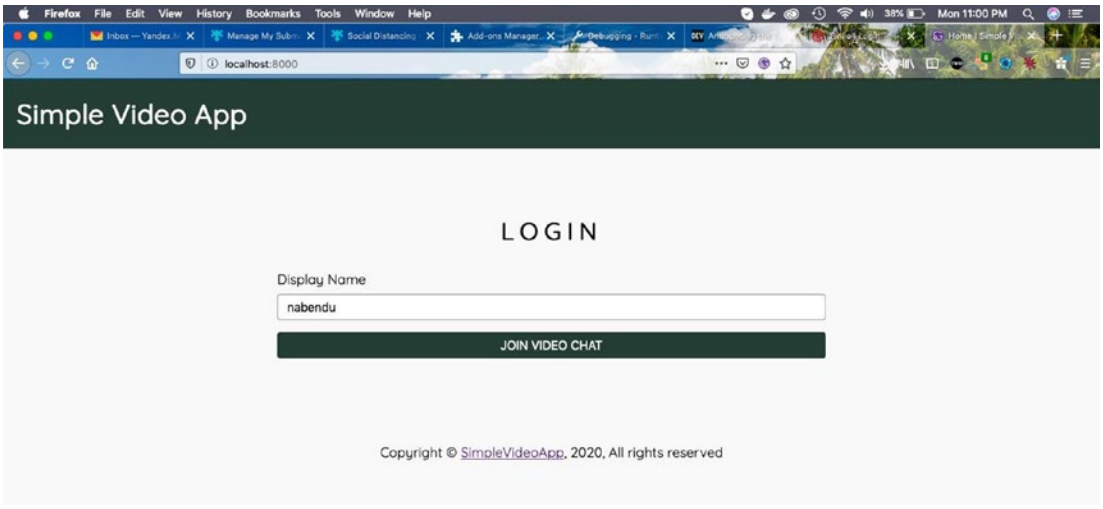
We are using a built-in `createLocalVideoTrack()` variable from Twilio to attach the video of the webcam to the `div localVidRef`. The updated code is shown in Listing 7-23.

Listing 7-23. Local Video in video.js

```
import React, { useEffect, useRef } from 'react'
import TwilioVideo from "twilio-video"

const Video = ({ token }) => {
    const localVidRef = useRef()

    useEffect(() => {
        TwilioVideo.connect(token, { video: true, audio: true, name: "SVA"
        }).then(
            result => {
                            TwilioVideo.createLocalVideoTrack().then(track
                            => {
                                localVidRef.current.appendChild(track.
                                attach())
                        })
            })
    }, [token])

    return (
        <div>
            <div ref={localVidRef} />
        </div>
    )
}

export default Video;
```

Go to `http://localhost:8000/`[9] again and log in with any username (see Figure 7-48).

[9]`http://localhost:8000/`

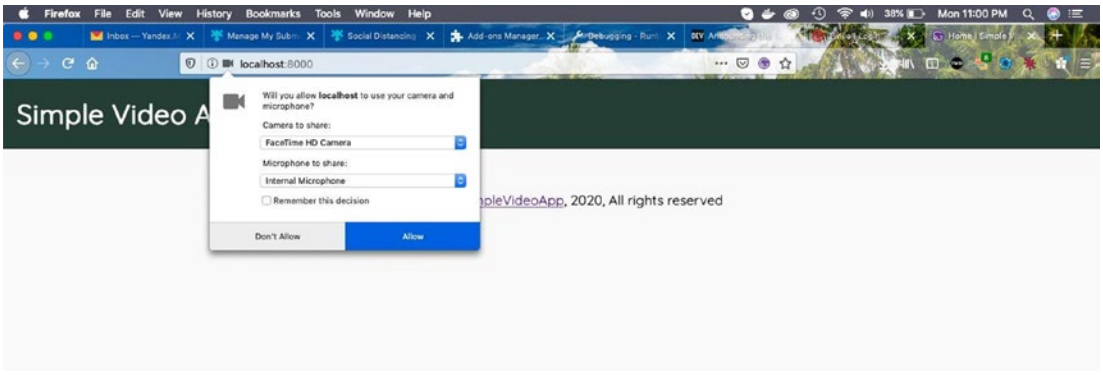


Display Name: nabendu

Copyright © SimpleVideoApp, 2020, All rights reserved

Figure 7-48. *Log in again*

After that, allow the browser to use the camera and microphone (see Figure 7-49).



Will you allow **localhost** to use your camera and microphone?

Camera to share: FaceTime HD Camera

Microphone to share: Internal Microphone

Remember this decision

Figure 7-49. *Allow the browser to use the camera and microphone*

You will be able to see yourself in the video (see Figure 7-50).

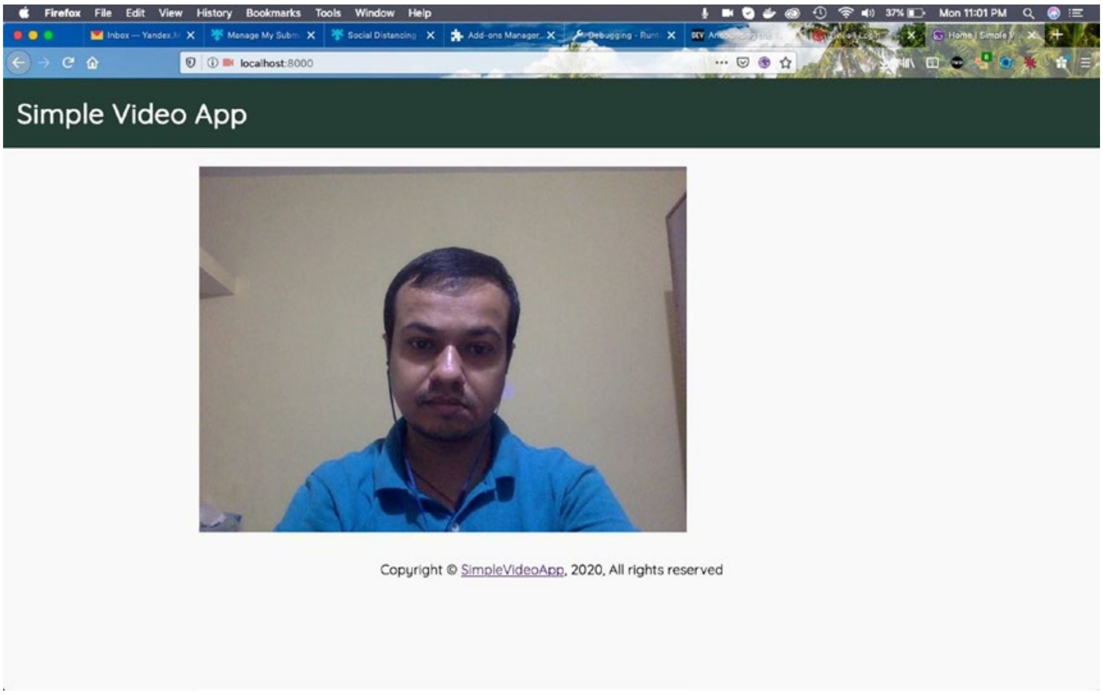


Figure 7-50. *The video is working*

It's time to attach the remote participants. Update the `video.js` file as shown in Listing 7-24. We are adding another `ref remoteVidRef` to a `div`.

We then take the result and loop through each participant. Each participant can have one or more tracks (cameras), so we are looping through them as well. We then append it to `remoteVidRef`. The updated code is shown in Listing 7-24.

Listing 7-24. Remote Video in video.js

```
import React, { useEffect, useRef } from 'react'
import TwilioVideo from "twilio-video"

const Video = ({ token }) => {
    const localVidRef = useRef()
    const remoteVidRef = useRef()
    useEffect(() => {
        TwilioVideo.connect(token, { video: true, audio: true, name: "SVA"
        }).then(
            result => {
                        TwilioVideo.createLocalVideoTrack().then(track
                        => {
                localVidRef.current.appendChild(track.attach())
              })

              const addParticipant = participant => {
                  participant.tracks.forEach(publication => {
                      if (publication.isSubscribed) {
                          const track = publication.track
                          remoteVidRef.current.appendChild(track.attach())
                      }
                  })

              }

              result.participants.forEach(addParticipant)
            })
    }, [token])

    return (
        <div>
            <div ref={localVidRef} />
            <div ref={remoteVidRef} />
        </div>
    )
}

export default Video;
```

It seems to be working, as we are able to see two videos now (see Figure 7-51).

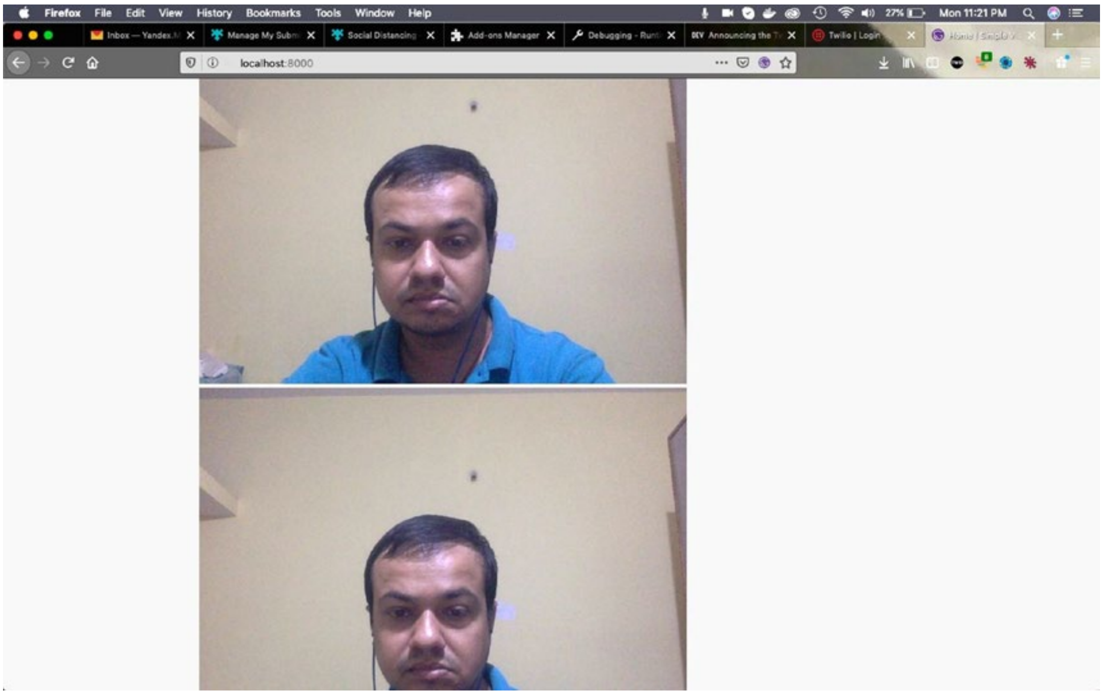
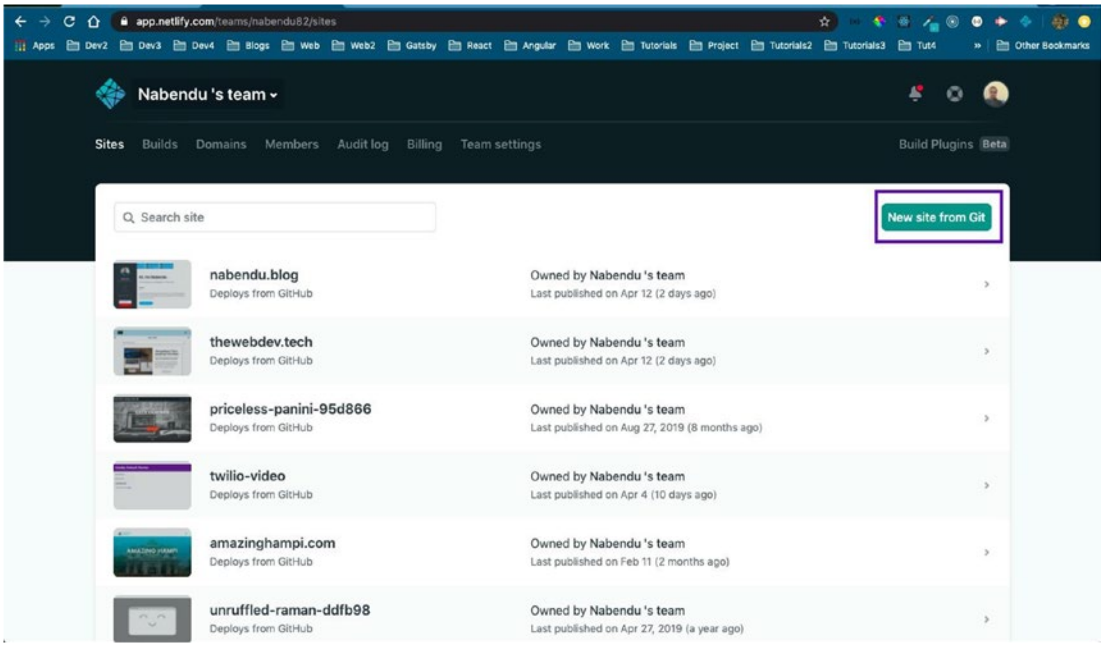


Figure 7-51. *Two videos are now displayed*

Deploying Netlify

It's time to deploy to Netlify, so that you can test from two devices. Open your Netlify dashboard and click the New Site from Git button (see Figure 7-52).



nabendu.blog — Deploys from GitHub — Owned by Nabendu 's team — Last published on Apr 12 (2 days ago)

thewebdev.tech — Deploys from GitHub — Owned by Nabendu 's team — Last published on Apr 12 (2 days ago)

priceless-panini-95d866 — Deploys from GitHub — Owned by Nabendu 's team — Last published on Aug 27, 2019 (8 months ago)

twilio-video — Deploys from GitHub — Owned by Nabendu 's team — Last published on Apr 4 (10 days ago)

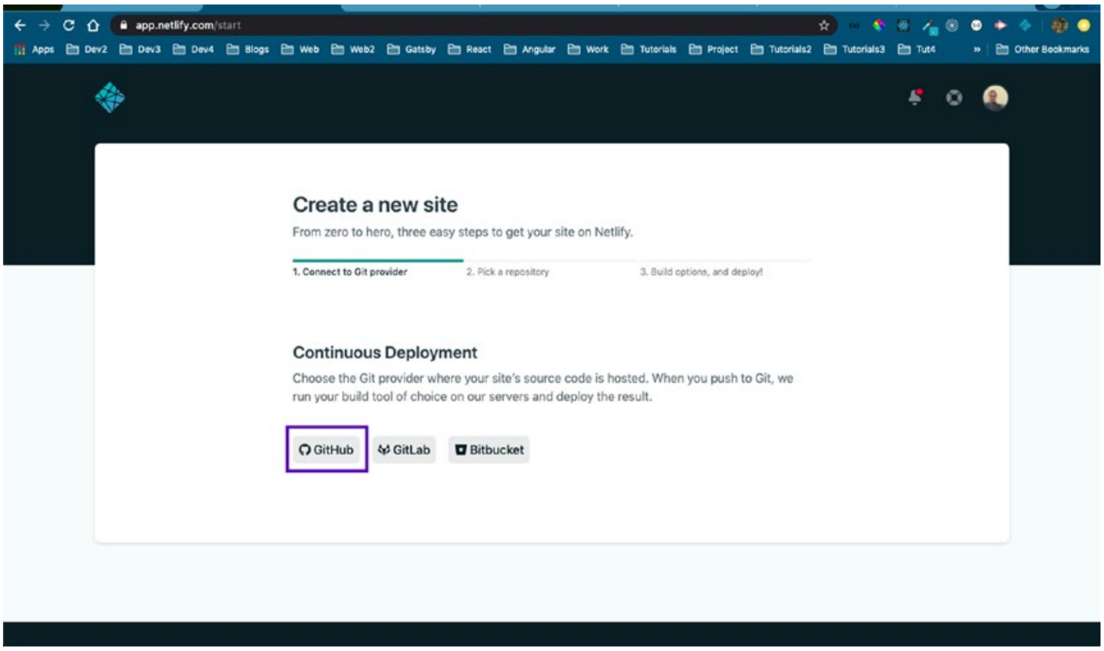
amazinghampi.com — Deploys from GitHub — Owned by Nabendu 's team — Last published on Feb 11 (2 months ago)

unruffled-raman-ddfb98 — Deploys from GitHub — Owned by Nabendu 's team — Last published on Apr 27, 2019 (a year ago)

Figure 7-52. *Netlify*

Next, click GitHub (see Figure 7-53), as my code is in https://github.com/nabendu82/SimpleVideoApp.[10]

[10]https://github.com/nabendu82/SimpleVideoApp


Create a new site

From zero to hero, three easy steps to get your site on Netlify.

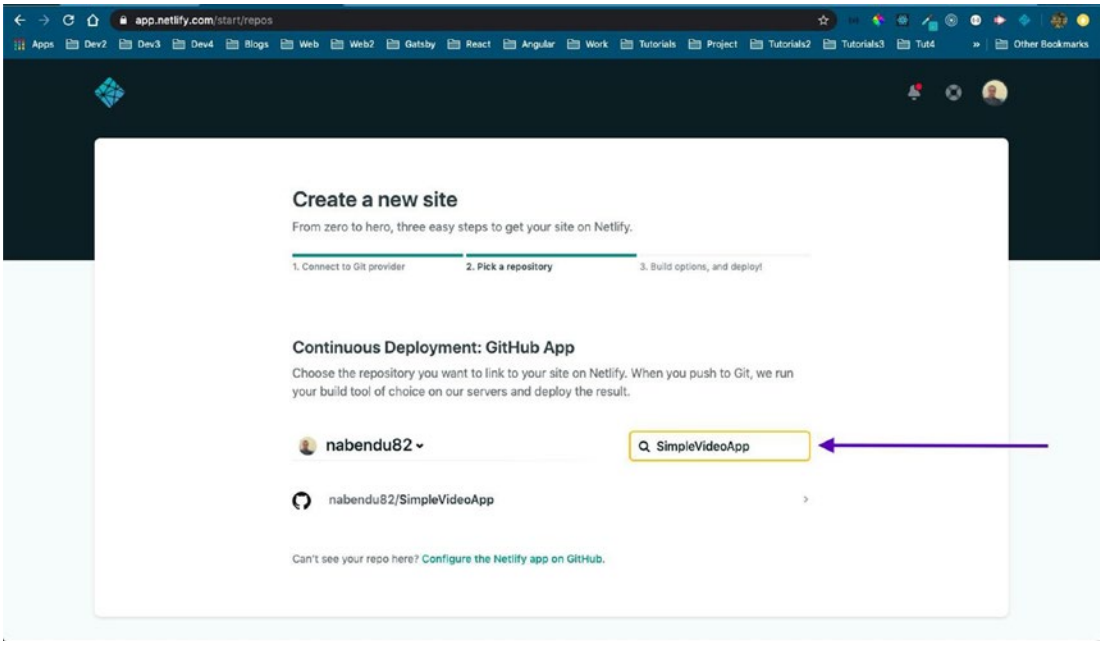
1. Connect to Git provider 2. Pick a repository 3. Build options, and deploy!

Continuous Deployment

Choose the Git provider where your site's source code is hosted. When you push to Git, we run your build tool of choice on our servers and deploy the result.

Figure 7-53. *GitHub*

After that, I need to search the repo, as I have a lot of them. After getting the correct repo, click it (see Figure 7-54).


app.netlify.com/start/repos

Create a new site

From zero to hero, three easy steps to get your site on Netlify.

1. Connect to Git provider 2. Pick a repository 3. Build options, and deploy!

Continuous Deployment: GitHub App

Choose the repository you want to link to your site on Netlify. When you push to Git, we run your build tool of choice on our servers and deploy the result.

nabendu82

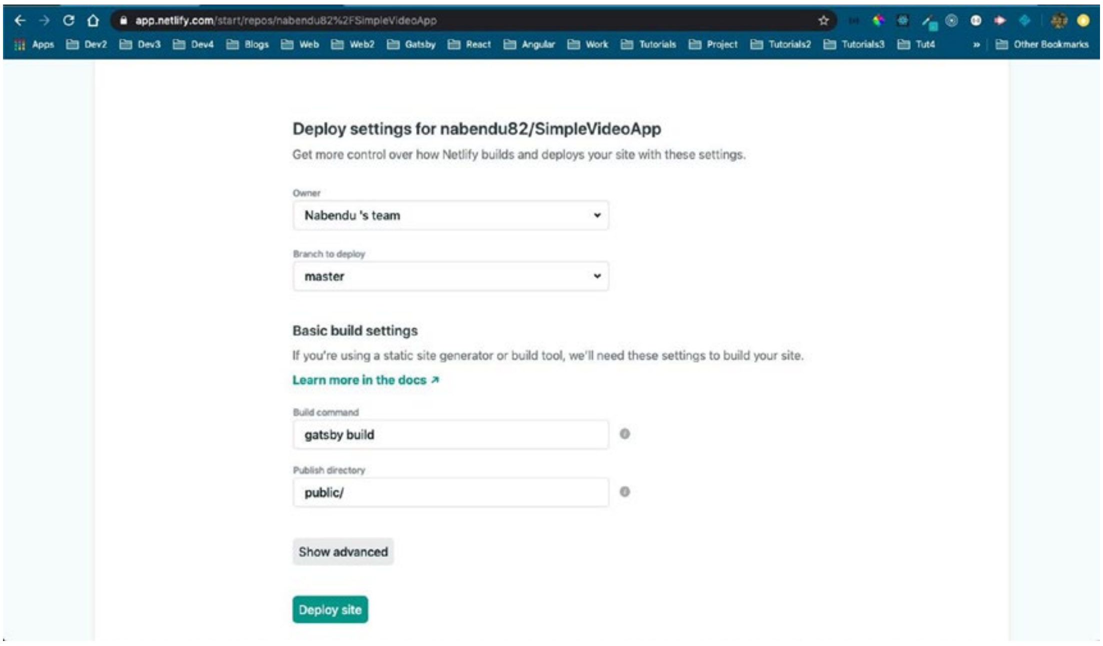
SimpleVideoApp

nabendu82/SimpleVideoApp

Can't see your repo here? Configure the Netlify app on GitHub.

Figure 7-54. *Searching for the correct repo*

After that, keep all the default settings and click the Deploy Site button (see Figure 7-55).


app.netlify.com/start/repos/nabendu82%2FSimpleVideoApp

Deploy settings for nabendu82/SimpleVideoApp

Get more control over how Netlify builds and deploys your site with these settings.

Owner: Nabendu 's team

Branch to deploy: master

Basic build settings

If you're using a static site generator or build tool, we'll need these settings to build your site.

Learn more in the docs

Build command: gatsby build

Publish directory: public/

Show advanced

Deploy site

Figure 7-55. *Deploy the site*

On the next screen, click the Site Settings button quickly (see Figure 7-56).

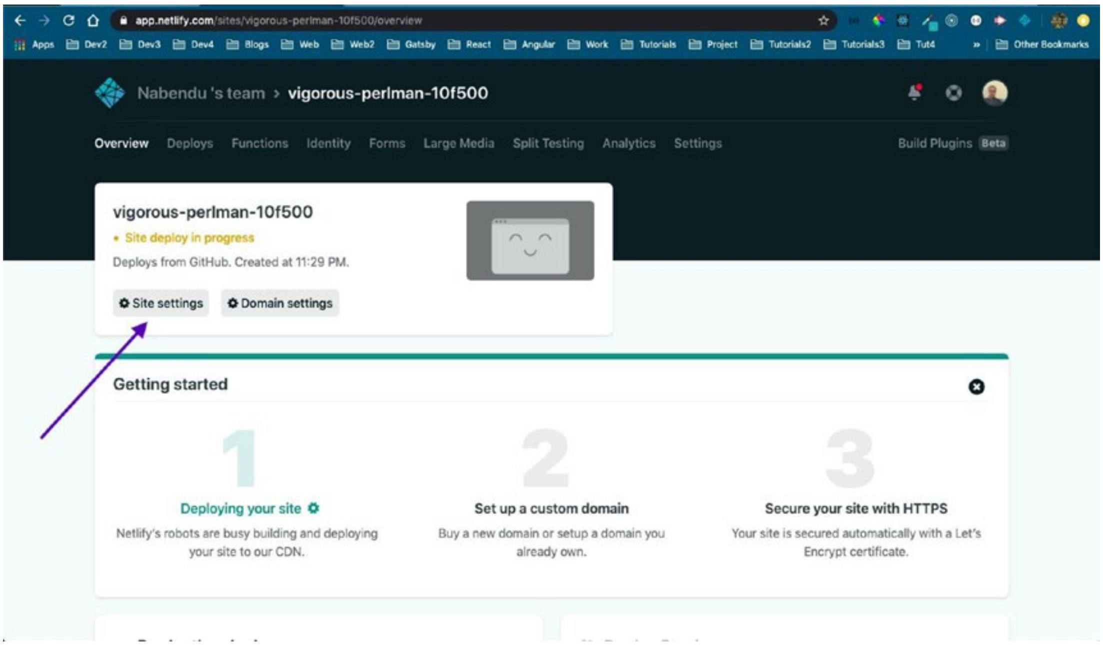
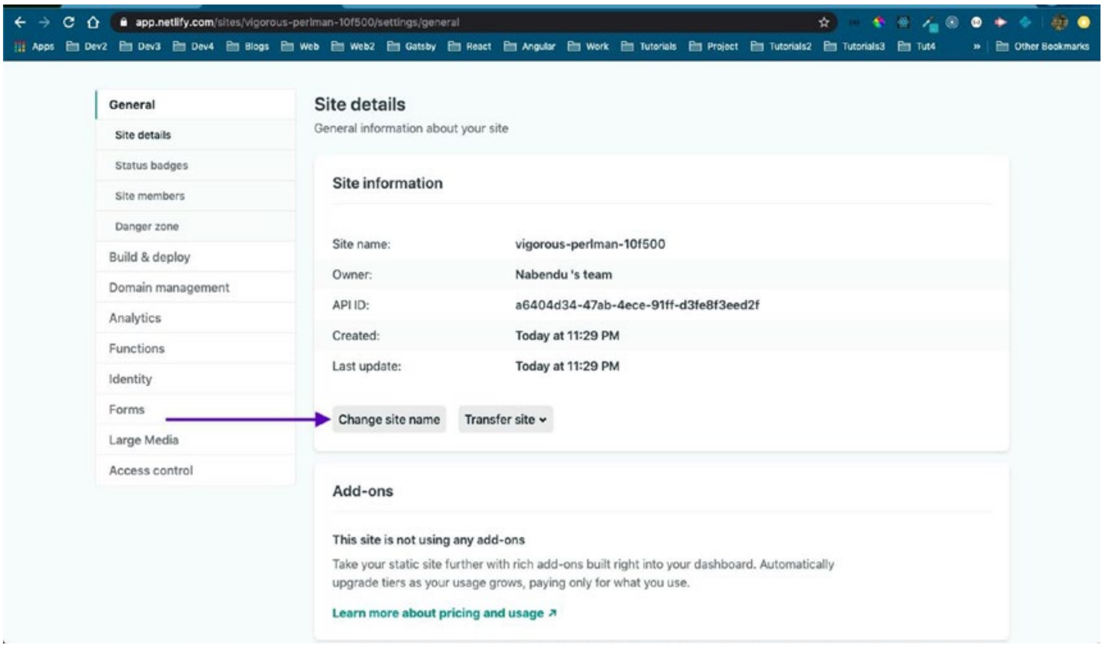

app.netlify.com/sites/vigorous-perlman-10f500/overview
Nabendu 's team > vigorous-perlman-10f500
Overview
Deploys
Functions
Identity
Forms
Large Media
Split Testing
Analytics
Settings
Build Plugins Beta
vigorous-perlman-10f500
Site deploy in progress
Deploys from GitHub. Created at 11:29 PM.
Site settings
Domain settings
Getting started
Deploying your site
Netlify's robots are busy building and deploying your site to our CDN.
Set up a custom domain
Buy a new domain or setup a domain you already own.
Secure your site with HTTPS
Your site is secured automatically with a Let's Encrypt certificate.

Figure 7-56. *Site settings*

Next, scroll a bit and click the Change Site Name button (see Figure 7-57).



Site details

General information about your site

Site information

Site name:	vigorous-perlman-10f500
Owner:	Nabendu 's team
API ID:	a6404d34-47ab-4ece-91ff-d3fe8f3eed2f
Created:	Today at 11:29 PM
Last update:	Today at 11:29 PM

Add-ons

This site is not using any add-ons

Take your static site further with rich add-ons built right into your dashboard. Automatically upgrade tiers as your usage grows, paying only for what you use.

Figure 7-57. *Change the site's name*

It will open a popup, in which you can change the random site name to something meaningful (see Figure 7-58).

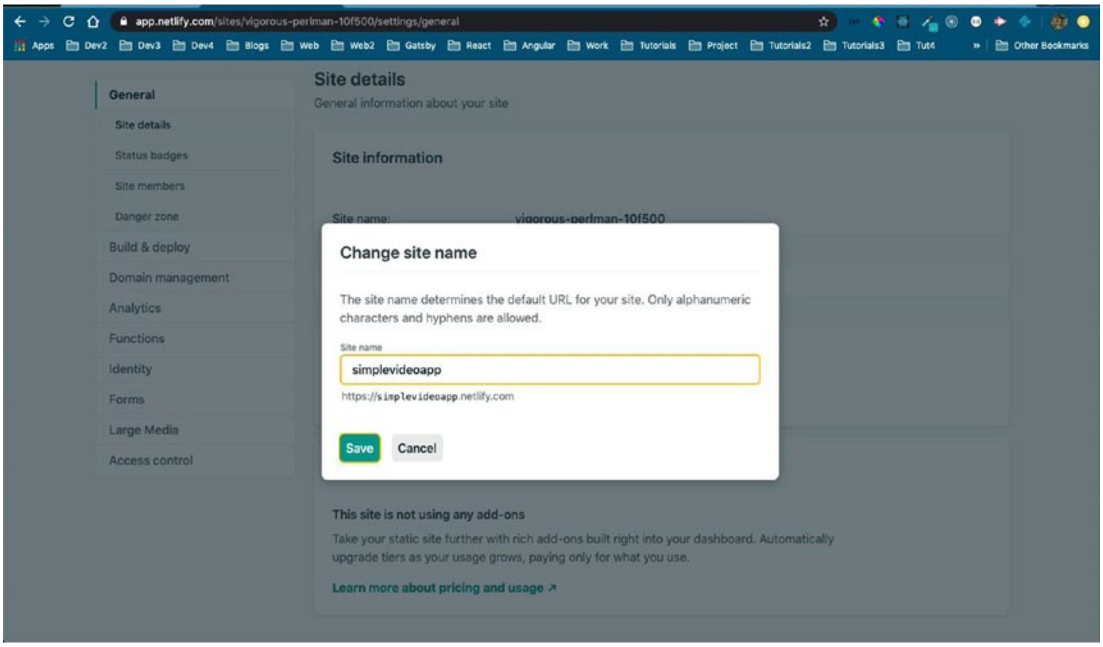

Site details
General information about your site
Site information
Change site name
The site name determines the default URL for your site. Only alphanumeric characters and hyphens are allowed.
Site name
simplevideoapp
https://simplevideoapp.netlify.com
Save
Cancel

Figure 7-58. Choose a meaningful name for your site

Finally, the site is deployed (see Figure 7-59).

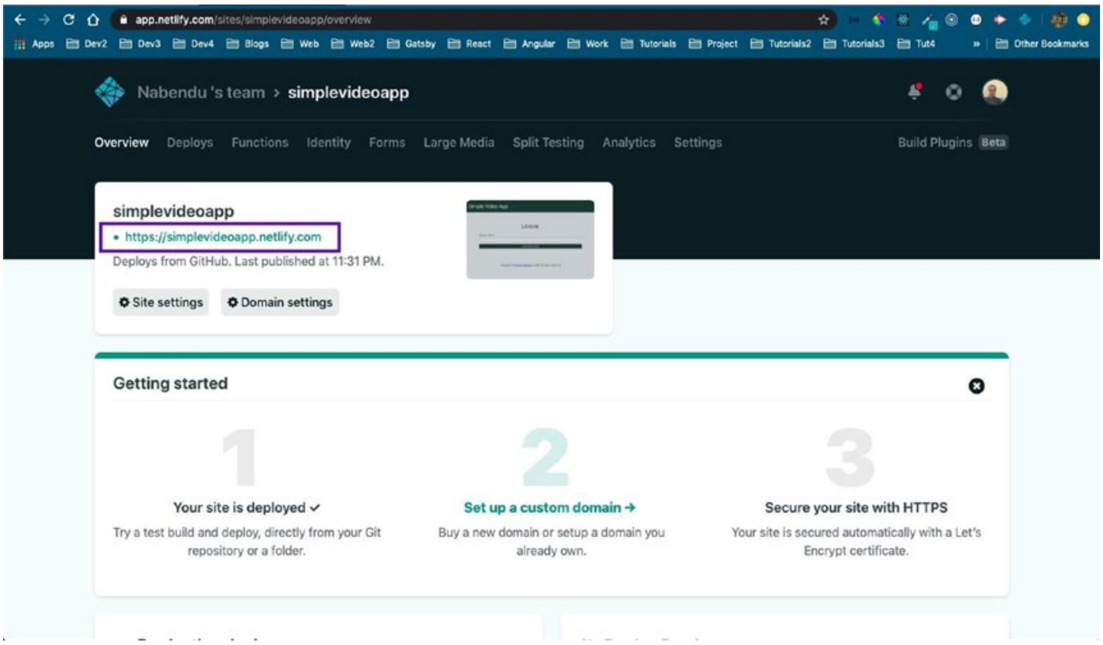


Figure 7-59. The site is deployed

The app is deployed and I logged in from two machines, but I was not able to see both videos. Per the YouTube video, I realized I missed a part. We need to add the updated part to the `video.js` file. The updated code is shown in Listing 7-25.

Listing 7-25. Fixes to the video.js File

```
...
...
const Video = ({ token }) => {
    const localVidRef = useRef()
    const remoteVidRef = useRef()

    useEffect(() => {
        TwilioVideo.connect(token, { video: true, audio: true, name: "SVA"
        }).then(
            result => {
                    TwilioVideo.createLocalVideoTrack().then(track => {
                    localVidRef.current.appendChild(track.attach())
                  })

                const addParticipant = participant => {
                    participant.tracks.forEach(publication => {
                        if (publication.isSubscribed) {
                            const track = publication.track
                            remoteVidRef.current.appendChild(track.attach())
                        }
                    })
                    participant.on("trackSubscribed", track => {
                        remoteVidRef.current.appendChild(track.attach())
                    })
                }

                result.participants.forEach(addParticipant)
                result.on("participantConnected", addParticipant)
            })
    }, [token])
            ...
            ...
```

I then push the code, which automatically deploys to Netlify. I am able to log in from multiple devices and see the videos (see Figure 7-60).

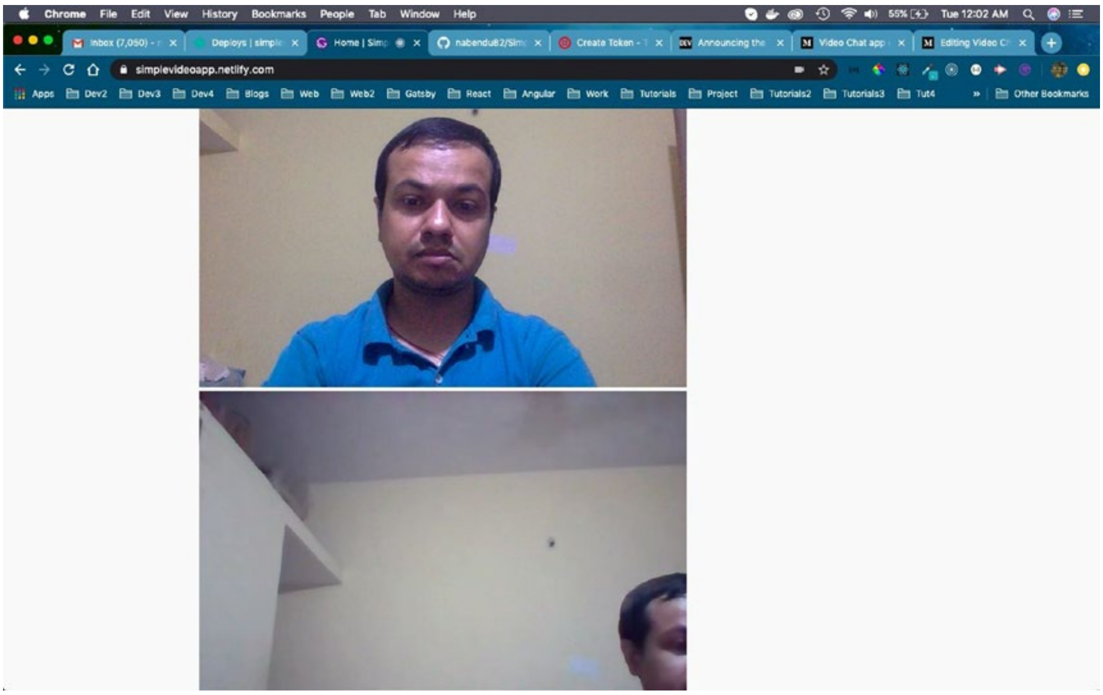


Figure 7-60. *Working!*

There is some CSS and some other changes remaining, before I can submit my app to the hackathon.

Making CSS Changes

Our web app is almost finished; only some CSS remains, so let's add it.

I also want to show the name of the organizer, so let's get it from `login-form.js`. We will use a callback function called `storeName`, which is similar to `storeToken`, to send the name back. The updated code is shown in Listing 7-26.

Listing 7-26. storeName in login-form.js

```
...
...
const LoginForm = ({ storeToken, storeName }) => {
    const [name, setName] = useState("")

    const handleSubmit = async event => {
        event.preventDefault()
        const result = await axios({
            method: "POST",
            url: "https://tan-cat-7689.twil.io/create-token",
            data: {
                identity: name,
            },
        })
        console.log(result);
        const jwt = result.data;
        storeToken(jwt);
        storeName(name);
    }

            ...
            ...
```

Next, let's update `index.js` and use the logic similar to that of token. We are passing the name to the Video component. The updated code is shown in Listing 7-27.

Listing 7-27. storeName in index.js

```
...
...

const IndexPage = () => {
  const [token, setToken] = useState(false);
  const [name, setName] = useState(false);
  return (
    <Layout>
      <SEO title="Home" />
      {!token ? <LoginForm storeToken={setToken} storeName={setName} /> :
      <Video token={token} name={name} />}
    </Layout>
  )
}

export default IndexPage
```

Next, let's update the Video component in the `video.js` file. Here, we are importing a `video.css` file. We will make it soon. After that, we are destructuring the `name` prop.

After that, in the `video.js` file, make the following changes. Change the enclosing `div` to a fragment and add `Organizer` and use the `name` prop.

Also add an `h2` for remote participants. Lastly, add a `className` for `remoteVideRef` `div`, which we are going to style next. The updated code is shown in Listing 7-28.

Listing 7-28. Styles in video.js

```
import React, { useEffect, useRef } from 'react'
import TwilioVideo from "twilio-video"
import './video.css'

const Video = ({ token, name }) => {
            ...
            ...
    return (

            <h2>Organizer: {name}</h2>
            <div ref={localVidRef} />
            <h2>Remote Participants</h2>
            <div className="remoteVideo" ref={remoteVidRef} />

    )
}

export default Video;
```

Add a `video.css` file to the same folder and add the simple style shown in Listing 7-29.

Listing 7-29. The video.css File

```
.remoteVideo{
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(260px, 1fr));
    grid-gap: 10px;
    justify-items: center;
}

video{
    width: 100%;
    max-width: 240px
}

h2 {
    margin: 1rem 0;
}
```

All the changes are done, so it's time to test it in `localhost`. It is working fine and I checked it in three different browsers (see Figure 7-61).

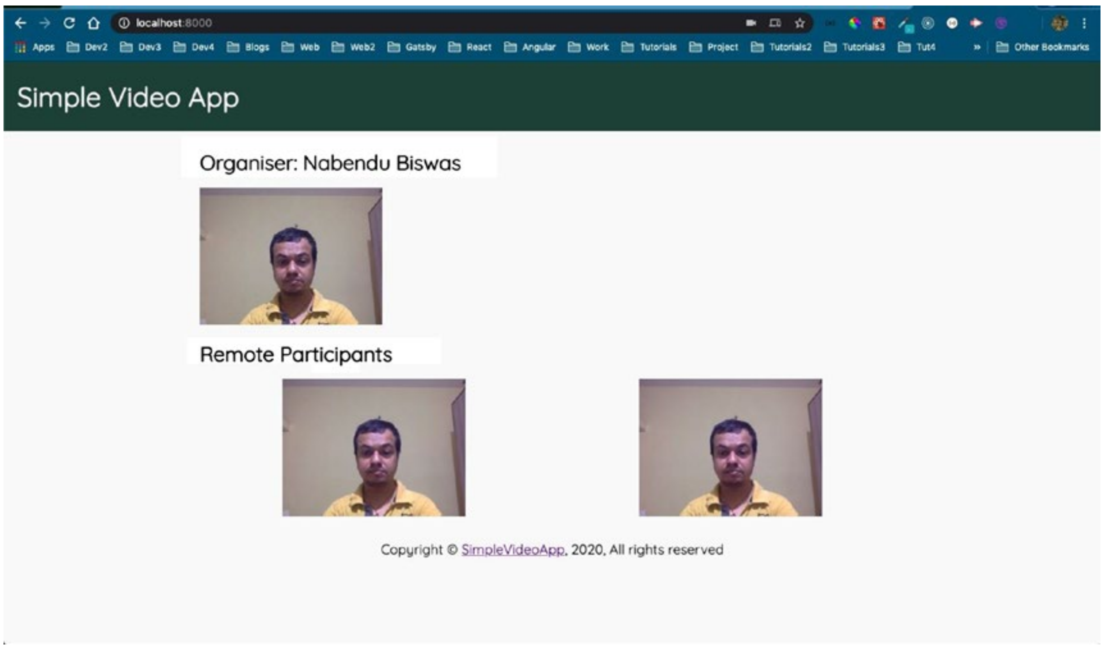
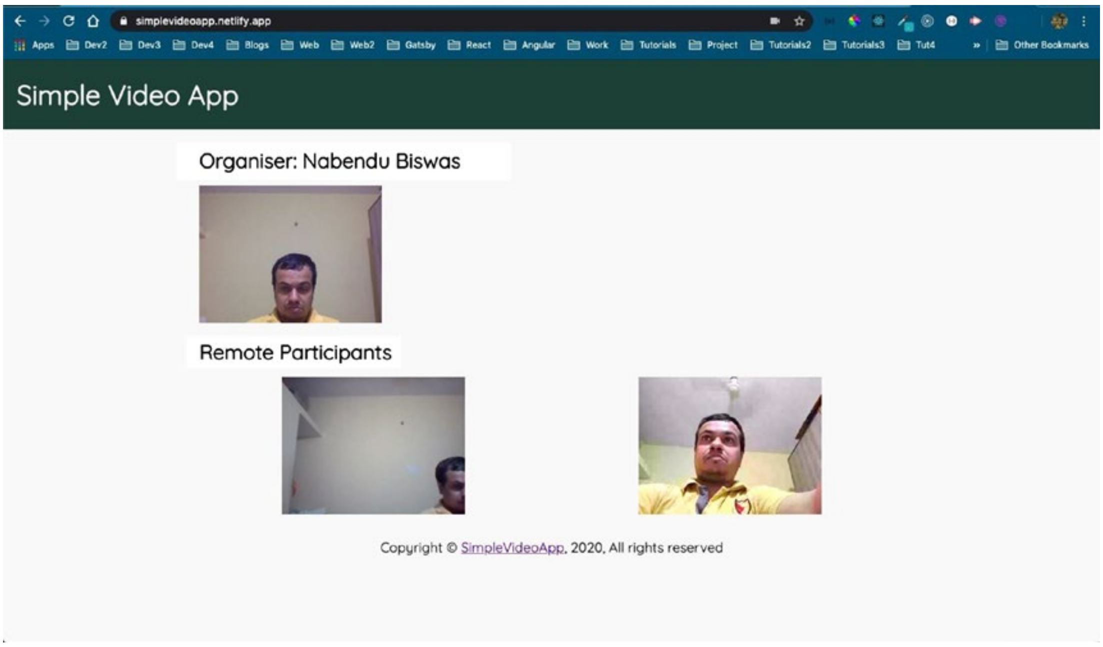


Figure 7-61. *Tested in the localhost*

Automatic Deployment

It's time to push the code in GitHub, to deploy it automatically. After it was deployed, I logged in from three different devices (see Figure 7-62).


simplevideoapp.netlify.app
Apps Dev2 Dev3 Dev4 Blogs Web Web2 Gatsby React Angular Work Tutorials Project Tutorials2 Tutorials3 Tut4 Other Bookmarks

Simple Video App

Organiser: Nabendu Biswas

Remote Participants

Copyright © SimpleVideoApp, 2020, All rights reserved

Figure 7-62. Deployed

It is working fine but I found a bug where, for remote participants, it will show their name in place of the word Organizer. Let's change the way it will look for each user. I am now showing the name in Connect, so it will be different for each user. The updated code is shown in Listing 7-30.

Listing 7-30. Bug Fix in video.js

```
    return (

            <h2>Organizer</h2>
            <div ref={localVidRef} />
            <h2>Remote Participants</h2>
            <div className="remoteVideo" ref={remoteVidRef} />
           <p>Connected : {name}</p>

 )
```

Let's add the code in Listing 7-31 to the `video.css` file.

Listing 7-31. The video.css File

```
p {
        margin-top: 1rem
}
```

It is now working fine for different users (see Figures 7-63 and 7-64).

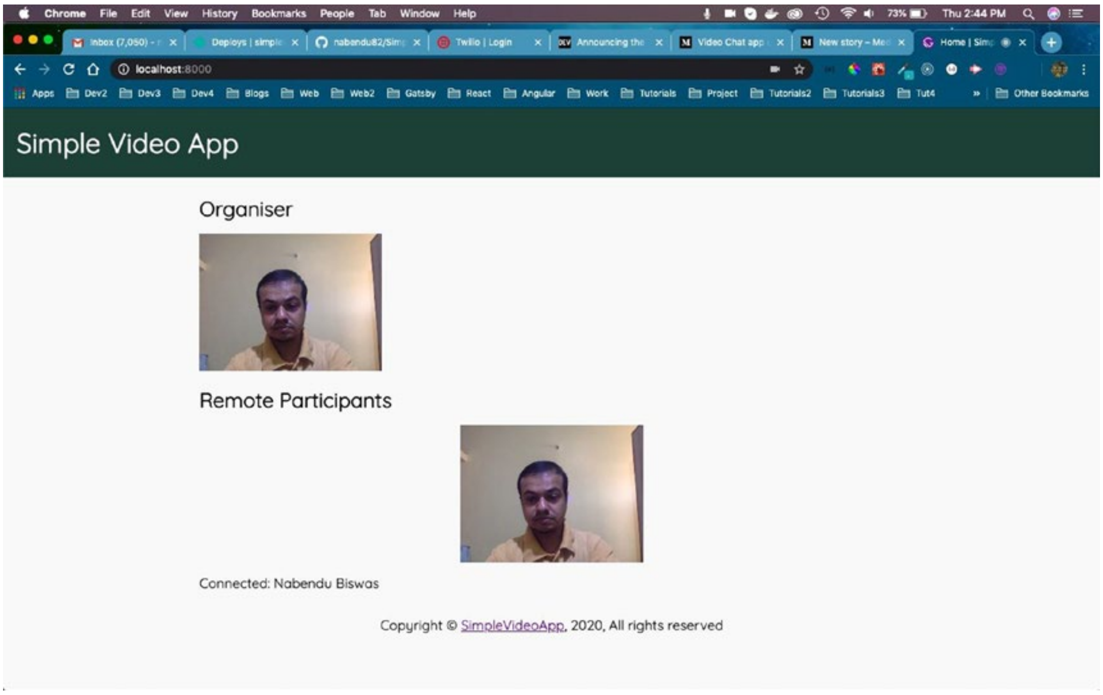

Chrome File Edit View History Bookmarks People Tab Window Help
localhost:8000
Simple Video App
Organiser
Remote Participants
Connected: Nabendu Biswas
Copyright © SimpleVideoApp, 2020, All rights reserved

Figure 7-63. *Organizer*

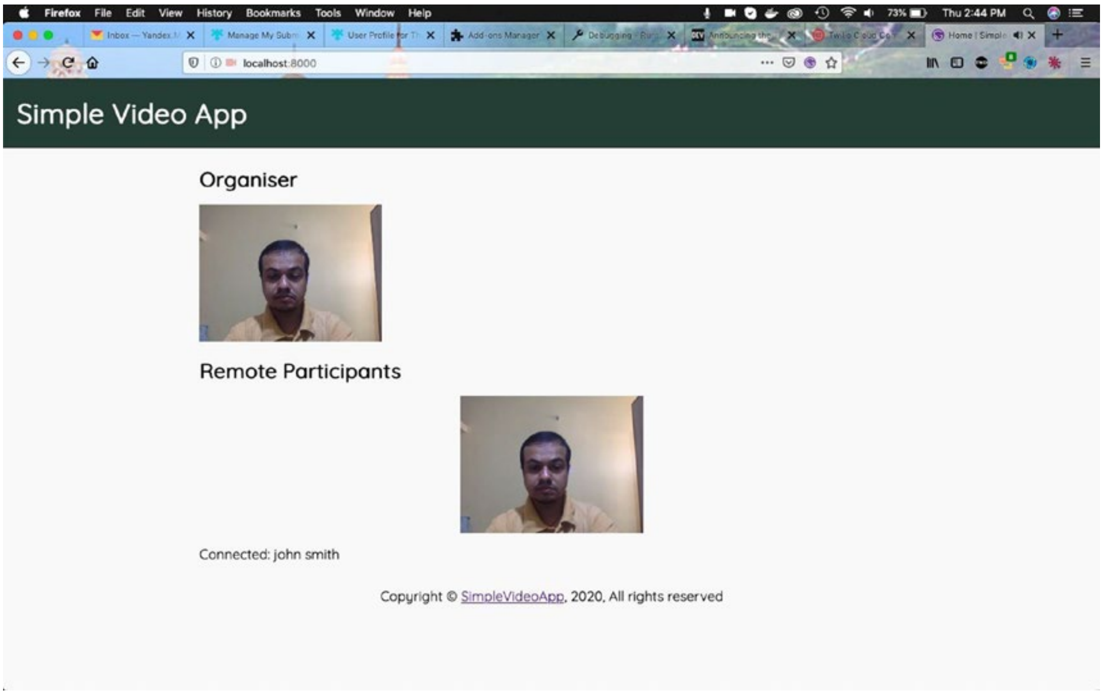


Figure 7-64. *Participant*

I will redeploy now and check the web app. I am testing with all three devices—two laptops and one phone (see Figure 7-65).

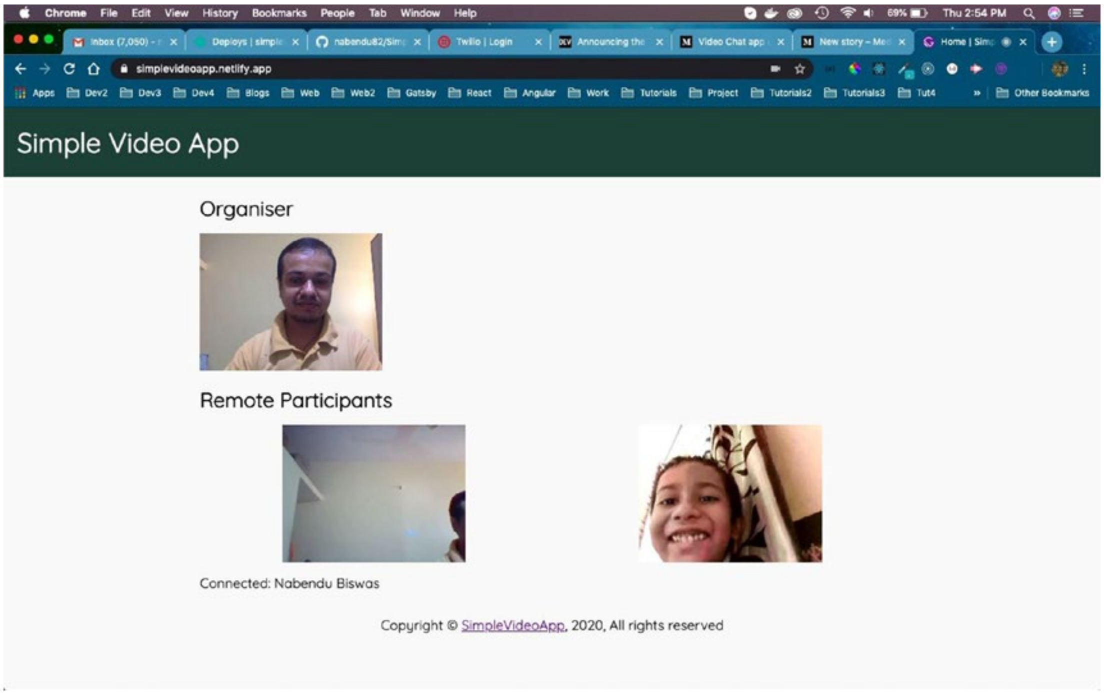
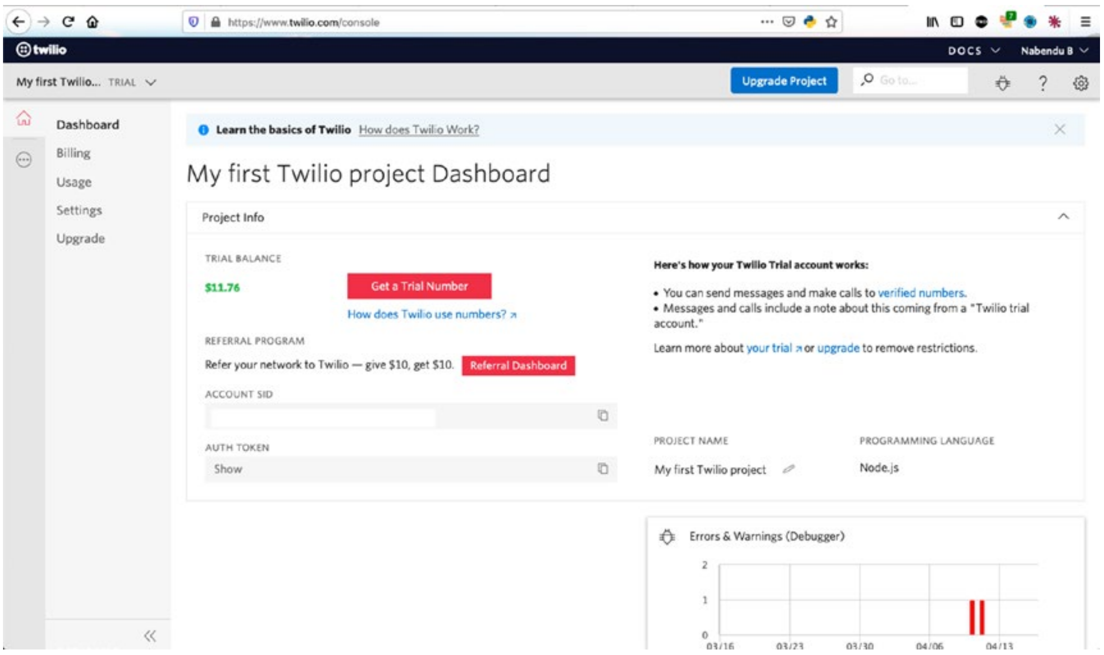

simplevideoapp.netlify.app
Simple Video App
Organiser
Remote Participants
Connected: Nabendu Biswas
Copyright © SimpleVideoApp, 2020, All rights reserved

Figure 7-65. *Working fine*

My app is complete and deployed. As I mentioned earlier, this is a very simple video app, which can be created with ease and deployed in no time with Netlify.

Share the link with your friends and enjoy video conferencing. We get $15 worth of free credits from Twilio and I used $3.24 while making this project (see Figure 7-66).



Learn the basics of Twilio How does Twilio Work?

My first Twilio project Dashboard

TRIAL BALANCE

$11.76

How does Twilio use numbers?

REFERRAL PROGRAM

Refer your network to Twilio — give $10, get $10.

ACCOUNT SID

AUTH TOKEN

Show

Here's how your Twilio Trial account works:

- You can send messages and make calls to verified numbers.
- Messages and calls include a note about this coming from a "Twilio trial account."

Learn more about your trial or upgrade to remove restrictions.

PROJECT NAME

My first Twilio project

PROGRAMMING LANGUAGE

Node.js

Figure 7-66. *Twilio dashboard*

I am submitting the web app soon for the hackathon, so I had to add README and LICENSE files. I added both in my GitHub and joined the Twilio CodeExchange Community.[11] Details about the submission rules are found here[12]. You can also find the code for this project at this[13] GitHub link.

Summary

I hope you liked the video chat app we created in this chapter. You can use it to create your own app. We covered the following topics in this chapter:

- Creating a video chat web app, using the awesome Twilio service
- Setting up the Twilio site
- Writing Twilio functions for the video chat app

[11]https://ahoy.twilio.com/code-exchange-community
[12]https://dev.to/devteam/announcing-the-twilio-hackathon-on-dev-2lh8
[13]https://github.com/nabendu82/SimpleVideoApp

Index