Bootstrap and React for Web Development

L A Liggett

2019-07-31

Contents

Chapter 1

Introduction

Chapter 2

HTML

2.1 HTML Properties

The head tag allows metadata to be labeled, the text of title for instance is typically listed in the tab or the status bar of the page in a browser.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>
            My Web Page!
        </title>
    </head>
</html>
```

The body specifies text for the page body.

```html
<!DOCTYPE html>
<html>
    <body>
        Hello, world!
    </body>
</html>
```

Headings specifies header text of increasingly small sizes.

```html
<!DOCTYPE html>
<html>
    <body>
        <h1>This is the largest headline</h1>
        <h2>This is also a large headline</h2>
        <h3>This is a slightly smaller headline</h3>
        <h4>This is an even smaller headline</h4>
        <h5>This is the second-smallest headline</h5>
        <h6>This is the smallest headline</h6>
    </body>
</html>
```

Unordered lists specify bullet points.

```
<!DOCTYPE html>
<html>
    <body>
        An Unordered List:
        <ul>
            <li>One Item</li>
            <li>Another Item</li>
            <li>Yet Another Item</li>
        </ul>
    </body>
</html>
```

Ordered lists number lines in increasing order.

```
<!DOCTYPE html>
<html>
    <body>
        An Ordered List:
        <ol>
            <li>First Item</li>
            <li>Second Item</li>
            <li>Another Item Here</li>
            <li>Fourth Item</li>
        </ol>
    </body>
</html>
```

The image tag refers to and inserts an image as an html attribute. The alt gives alternative code if the image is missing. The height and width sets the image size in number of pixels. When the image size is set to 50% sets the image size dynamically to 50% of the browser width or height.

```
<!DOCTYPE html>
<html>
    <body>
        <img src="cat.jpg" alt="cat" width="300" height="200">
        <img src="cat.jpg" alt="cat" width="50%" >
    </body>
</html>
```

Tables display data in a table format that can be styled in various ways. The `th` tag specifies the headings of each of the columns. The `td` tag specifies the data in each of the columns.

```
<!DOCTYPE html>
<html>
    <body>
        <table>
            <tr>
                <th>First Name</th>
                <th>Last Name</th>
                <th>Years in Office</th>
            </tr>
            <tr>
                <td>George</td>
                <td>Washington</td>
                <td>1789-1797</td>
            </tr>
            <tr>
                <td>John</td>
                <td>Adams</td>
                <td>1797-1801</td>
            </tr>
            <tr>
                <td>Thomas</td>
                <td>Jefferson</td>
                <td>1801-1809</td>
            </tr>
        </table>
    </body>
</html>
```

Tables can be styled within the header of the html document. Both the `th` and the `td` styles are defined together. `border-collapse` combines the borders of cells together.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Presidents</title>
        <style>
            table {
                border: 2px solid black;
                border-collapse: collapse;
                width: 50%;
            }

            th, td {
                border: 1px solid black;
                padding: 5px;
                text-align: center;
            }

            th {
                background-color: lightgray;
            }
        </style>
    </head>
    <body>
        <table>
            <tr>
                <th>First Name</th>
                <th>Last Name</th>
                <th>Years in Office</th>
            </tr>
            <tr>
                <td>George</td>
                <td>Washington</td>
                <td>1789-1797</td>
            </tr>
        </table>
    </body>
</html>
```

Forms can be created and labeled as such. The `placeholder` text is what is written within the form before anything is entered into it. The `name` is similar to a variable name and can be used to refer to the form and the data that is entered into it. The text within the button is the text that will appear on the button in the page.

```
<!DOCTYPE html>
<html>
    <body>
        <form>
            <input type="text" placeholder="Full Name" name="name">
            <button>Submit!</button>
        </form>
    </body>
</html>
```

Text can be aligned and colord by specifying styles within the respective tags of text.

```
<!DOCTYPE html>
<html>
    <body>
        <h1 style="color:red;text-align:center;">Welcome to My Web Page!</h1>
        <h1 style="color:#4290f5;text-align:center;">Second heading</h1>
    </body>
</html>
```

Style elements can be separated from the actual body of the webpage. In this example every `h1` is styled within the style portion of the header.

```
<!DOCTYPE html>
<html>
    <head>
        <title>My Web Page!</title>
        <style>
            h1 {
                color: red;
                text-align: center;
            }
        </style>
    </head>
    <body>
        <h1>Welcome to My Web Page!</h1>
    </body>
</html>
```

2.2 CSS

CSS properties can be found here.

Instead of putting the css styles within the header of the html file, they can be included in a separate css file and referenced. In this example, the type of file being referenced is classified as a `stylesheet` and the code is within `styles.css`.

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" href="styles.css">
    </head>
</html>
```

The code that goes within the css file is here, and it is simply the same code that was put into the style headers in the above example.

```
h1 {
    color: blue;
    text-align: center;
}
```

Divisions define sections of the code that can be separated so it can be controlled in a particular manner. Font priorities are taken left to right if some fonts are not found.

```
<!DOCTYPE html>
<html>
    <head>
        <title>My Web Page!</title>
        <style>
            div {
                background-color: teal;
                width: 500px;
                height: 400px;
                margin: 30px;
                padding: 20px;
                font-family: Arial, sans-serif;
                font-size: 28px;
                font-weight: bold;
                border: 1px dotted black;
            }
        </style>
    </head>
    <body>
        <div>
            Hello, world!
        </div>
    </body>
</html>
```

Divisions and spans can be named and used to refer to different parts of the html document specifically.

```
<!DOCTYPE html>
<html>
    <head>
        <title>My Web Page!</title>
        <style>
            #top {
                font-size: 36px;
                color: red;
            }

            .name {
                font-weight: bold;
                color: blue;
            }
        </style>
    </head>
    <body>
        <div id="top">
            This is the <span class="name">top</span> of my web page.
        </div>
    </body>
</html>
```

Link styling can be done by adjusting colors and text decorations of links.

```
<style>
    a:link {
      color: blue;
      background-color: transparent;
      text-decoration: none;
    }

    a:visited {
      color: red;
      background-color: transparent;
      text-decoration: none;
    }

    a:hover {
      color: pink;
      background-color: transparent;
      text-decoration: underline;
    }

    a:active {
      color: orange;
      background-color: transparent;
      text-decoration: underline;
    }
</style>
```

Chapter 3

Bootstrap

3.1 Setup

Create a folder that will contain the webfiles use npm to initialize a `package.json` file. Follow through the prompts to add the desired information. Set the entry point to be `index.html`. It can also be helpful to add the node_modules folder to `.gitignore`.

```
npm init
```

Then just initialize some basic `index.html` file for testing purposes.

```
<body>
    <h1>This is a Header</h1>
    <p>This is a paragraph</p>
</body>
```

Install the lite server, which will serve up the content from the folder. The `save-dev` flag will add the information to the json file that the lite-server should be used to serve the content. This should add lite-server under the devDependencies listing within `package.json` and a node-modules folder.

```
npm install lite-server --save-dev
```

Within the `package.json` file, add the start and lite listings so it looks like the following.

```
"scripts": {
  "start": "npm run lite",
  "test": "echo \"Error: no test specified\" && exit 1",
  "lite": "lite-server"
},
```

The lite-server can then be run using npm start.

```
npm start
```

Chapter 4

Hackernews

4.1 Setup

First make sure create react app is installed. The project here follows this tutorial. There are lots of other good looking tutorials like The React Handbook, and others at gitconnected.

```
npm i -g create-react-app
```

Then create a new directory for the app.

```
create-react-app hacker-news-clone
```

Change into the newly created directory and then create a file to handle environmental variables.

```
cd hacker-news-clone
touch .env
```

Within the `.env` file refer to the `src` folder. This will allow dependencies to be more easily imported. Add the following to the `.env` file.

```
NODE_PATH=src
```

Make a components directory within `src` to hold all of the components for the project.

```
mkdir -p src/components/App
```

Make a services directory within `src` to add additional functionality to the app and reference other site APIs.

```
mkdir src/services
```

Make a styles directory within `src` to add styles that can be used across the app.

```
mkdir -p src/styles
```

Make a store directory within `src` to add styles that will add Redux function.

```
mkdir -p src/store
```

Make a utils directory within `src` for shared functions across the app.

```
mkdir -p src/utils
```

Now move `App.js` to components just to keep the components bundled together. Rename `App.js` to index so that it can be imported from the mycomponents app.

```
mv src/App*js src/components/App/
mv src/components/App/App.js src/components/App/index.js
mv src/logo.svg src/components/App/
```

Delete the css files because style components will be used instead.

```
rm src/*css
```

Remove the imports of the css files in `src/components/App/index.js`.

```
import './App.css';
```

And remove the import within `src/index.js`.

```
import './index.css';
```

Now create some styles to be used throughout the app.

```
mkdir src/styles
touch src/styles/globals.js
touch src/styles/palette.js
```

The js files above contain routine code that can be copied from the author's github page. Alternatively, here is the code for `global.js`.

```
import { injectGlobal } from 'styled-components';
import { colorsDark } from './palette';

const setGlobalStyles = () =>
  injectGlobal`
    * {
      box-sizing: border-box;
    }
    html, body {
      font-family: Lato,Helvetica-Neue,Helvetica,Arial,sans-serif;
      width: 100vw;
      overflow-x: hidden;
      margin: 0;
      padding: 0;
      min-height: 100vh;
      background-color: ${colorsDark.background};
    }
    ul {
      list-style: none;
      padding: 0;
    }
    a {
      text-decoration: none;
      &:visited {
        color: inherit;
      }
    }
  `;

export default setGlobalStyles;
```

And here is the code for `palette.js`.

```
export const colorsDark = {
  background: '#272727',
  backgroundSecondary: '#393C3E',
  text: '#bfbebe',
  textSecondary: '#848886',
  border: '#272727',
};

export const colorsLight = {
  background: '#EAEAEA',
  backgroundSecondary: '#F8F8F8',
  text: '#848886',
  textSecondary: '#aaaaaa',
  border: '#EAEAEA',
};
```