Bootstrap and React for Web Development

L A Liggett

2019-08-05

Contents

1 Introduction **5**

2 HTML **7**
2.1 HTML Properties 7
2.2 CSS 11

3 Bootstrap **19**
3.1 Setup 19

4 Hackernews **21**
4.1 Setup 21

Chapter 1

Introduction

Chapter 2

HTML

2.1 HTML Properties

Commenting in HTML.

```html
<!--
These are some comments.
-->
```

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>
```

Links to local pages or hyperlinks are included within the `<a>` tag.

```
<a href="about.html">About</a>
<a href="http://www.google.com">Google</a>
```

Links can also refer to locations on the same page.

```
<a href="section1">Section 1</a>
<h1 id="section1">Some stuff.</h1>
```

A newline can be inserted within the body of text by using `<br />`.

```
<a href="about.html">About</a><br />
```

2.1.1 Forms

Generic text input fields can be created with the text type.

```
<div>
<input name="name" type="text" placeholder="Name">
</div>
```

A password field is pretty similar to a text field, but the characters are obscured.

```
<input name="password" type="password" placeholder="Password">
```

Dropdown lists of the possible valid choices for a field can be used with `datalist`.

```
<input name="country" list="countries" placeholder="Country">
<datalist id="countries">
    <option value="Afghanistan">
    <option value="Albania">
    <option value="Algeria">
```

2.2 CSS

Commenting in CSS.

```
/*
These are some comments.
*/
```

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>
```

Fonts can be imported from locations like google's hosted fonts and used directly to avoid problems with a browser not supporting them. The link to the fonts goes within the header portion of the html code.

```
<head>
    <link href="https://fonts.googleapis.com/css?family=Cormorant+Garamond|Proza+Libre&display=swap" rel
</head>
```

The imported font families can then be used within the CSS directly.

```
body {
    font-family: Proza Libre, Cormorant Garamond
}
```

Nested elements can be styled in a grouped manner.

```
<style>
    ol li {
        color: red;
    }
</style>
<body>
    <ol>
        <li>list item</li>
        <li>second list item</li>
    </ol>
</body>
```

A similar use is to style the immediately nested child elements and none other using the `>` operator.

```
<style>
    ol > li {
        color: red;
    }
</style>
```

```
<body>
    <ol>
        <li>this will be colored</li>
        <ul>
            <li>this won't be colored</li>
        </ul>
        <li>this will also be colored</li>
    </ol>
</body>
```

Fields of particular types can be styled based on their type. These are examples of a text field that allows letters and numbers and a number field that only allows numbers.
The fields can then be styled based on the type of field that they are.

```
<style>
    input[type=text] {
    background-color: red;
}
</style>

<body>
    <input name="name" type="text" placeholder="First Name">
    <input name="name" type="number" placeholder="Age">
</body>
```

2.2.1 Selectors

CSS Selectors allow specific classes or elements to be selected and styled individually.

```
a, b /* Multiple element selector */
a b /* Descendant selector */
a > b /* Child selector */
a + b /* Adjacent sibling selector */
[a=b] /* Attribute selector */
a:b /* Pseudoclass selector */
a::b /* Pseudoelement selector */
```

Pseudo-classes allow for different styling effects depending on the state of the element.

```
<style>
    button {
        background-color: green;
    }
    button:hover {
        background-color: orange;
    }
</style>
<body>
    <button>Click</button>
</body>
```

Pseudo-elements are similar but select elements also allow things to be styled by placing information at the beginning of an item.
What is happening here is that there is a link amd before the link it says “Click here:”, and to the left of that the `\21d12` specifies a unicode arrow.

```
<style>
    a::before {
        content: "\21d2 Click here: ";
        font-weight: bold;
    }
</style>
<body>
    <a href="#">A link</a>
</body>
```

Text highlighting can also be controlled with pseudo-elements. Here the text color turns red and the highlight is in yellow when the text is highlighted.

```
<style>
    p::selection {
        color: red;
        background-color: yellow;
    }
</style>
<body>
    <p>This is some text</p>
</body>
```

2.2.2 Responsive Design

Media queries are CSS rules that are only used if certain properties are true. A commonly used property is screen size to adjust page layouts fore mobile. It is generally a good idea to design for mobile first and adjust properties to fit desktop, as this will ensure mobile gets the fastest performace.

Here the width of a column is being altered if the browser window is at least 768px in size. Altering the design in this way illustrates how development can be done “mobile-first”, as the property is altered if a desktop is used instead of mobile.

```
@media only screen and (min-width: 768px) {
  /* For desktop: */
  .col-1 {width: 8.33%;}
}
```

Media queries can also control content what content gets printed. Here both paragraphs get displayed on the page, but only the first paragraph appears when the page is printed.

```
<style>
    @media print {
        .screen-only {
            display: none;
        }
    }
</style>
<body>
    <p>This gets printed</p>
    <p class="screen-only">This does not get printed</p>
</body>
```

The content of the page can also be changed using media queries.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
    @media (min-width: 500px) {
        h1::before {
            content: "Welcome to My Web Page!";
        }
    }

    @media (max-width: 499px) {
        h1::before {
            content: "Welcome!";
        }
    }
</style>
<body>
    <h1></h1>
</body>
```

2.2.3 Flexbox

Flexbox styling allows elements to be dynamically arranged to fit the screen. With a wide enough screen the div elements will all be in one row, but as the screen shrinks, they will move down into new rows.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
    .container {
        display: flex;
        flex-wrap: wrap;
    }

    .container > div {
        background-color: springgreen;
    }
</style>
<body>
    <div class="container">
        <div>Some stuff</div>
        <div>Some stuff</div>
        <div>Some stuff</div>
    </div>
</body>
```

2.2.4 Grid Styling

In the following example a grid system is being used for the items being displayed. the grid creates the first two colums as 200px and the last column is automatically sized to fill the rest of the remaining screen.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
    .grid {
        background-color: green;
        display: grid;
        grid-template-columns: 200px 200px auto;
    }

    .grid-item {
        background-color: white;
    }
</style>
<body>
    <div class="grid">
        <div class="grid-item">1</div>
        <div class="grid-item">2</div>
        <div class="grid-item">3</div>
    </div>
</body>
```

Chapter 3

Bootstrap

3.1 Setup

The bootstrap stylesheet `<link>` can be used directly `stackpath.com` by including the following reference in the `<head>` before any other listed stylesheets.

```
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" i
```

The `charset` and `viewport` meta tags are often required for proper bootstrap responsive behaviors, and should be included when using the bootstrap css.
The `viewport` line is a responsive meta tag that ensures proper rendering and touch zooming for mobile devices.

```
<!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
```

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',
};
```