



Community Experience Distilled

Learning Web Development with React and Bootstrap

**Build real-time responsive web apps using React
and Bootstrap**

**Harmeet Singh
Mehul Bhatt**

[PACKT] open source*
PUBLISHING community experience distilled

Table of Contents

Chapter 1: Getting Started with React and Bootstrap	1
ReactJS	2
Setting up the environment	3
Installing ReactJS and Bootstrap	4
Installing React	5
Bootstrap	6
Installing Bootstrap	7
Using React	8
Static form with React and Bootstrap	10
Summary	16
Chapter 2: Lets build a responsive theme with React-Bootstrap and React	18
Setting up	18
Scaffolding	19
Navigation	20
React-Bootstrap	23
Installing React-Bootstrap	23
Using React-bootstrap	23
Benefits of React – Bootstrap	24
Bootstrap Grid System	27
Helper Classes	32
Floats	32
Center elements	32
Show and hide	32
React Components	33
React.createElement()	34
Summary	38
Chapter 3: ReactJS - JSX	39
What is JSX in React	39
Advantages of using JSX in React	39
How to make your code neat and clean	39
Acquaintance or understanding	40
Semantics / structured syntax	40
Composite component	41

Namespace components	42
JSXTransformer	48
Attribute expressions	49
Boolean attributes	49
JavaScript expressions	49
Styles	49
Events	50
Attributes	50
Spread Attributes	50
Example of a Dynamic Form with JSX	51
Summary	56
Chapter 4: DOM Interaction with ReactJS	58
Props and state	58
Form components	59
Props in form components	60
Controlled component	60
Uncontrolled component	62
Getting the form values on submit	63
Ref attribute	63
Bootstrap Helper Classes	75
Caret	75
Clearfix	75
Summary	75

1

Getting Started with React and Bootstrap

There are many different ways to build modern web application with JavaScript and CSS, including a lot of different tool choices, and a lot of new theory to learn. This book introduce you to ReactJS and Bootstrap which you will likely come across as you learn about modern web app development, then takes you through the theory behind **Model View Controller (MVC)** and it's comparison, the most common type of app architecture used on the Web.

If you want to learn code then you have to write code, whatever way you feel comfortable, that way try to create small components / code samples which gives you more clarity / understanding of any technology. Now, let's see how this book is going to make your life easy when it comes to Bootstrap and ReactJS as we are going to cover theory and will build two super simple real time examples.

- “Hello World!” with ReactJs
- Simple static form application with React and Bootstrap

Facebook has really changed the way we think about front-end UI development with the introduction of React. One of the main advantages of this component based approach is that it is easy to reason about as the view is just a function of props and state.

We're going to cover the following topics:

- Setting up the environment
- ReactJS setup
- Bootstrap setup
- Why Bootstrap
- Static Form example with react and bootstrap

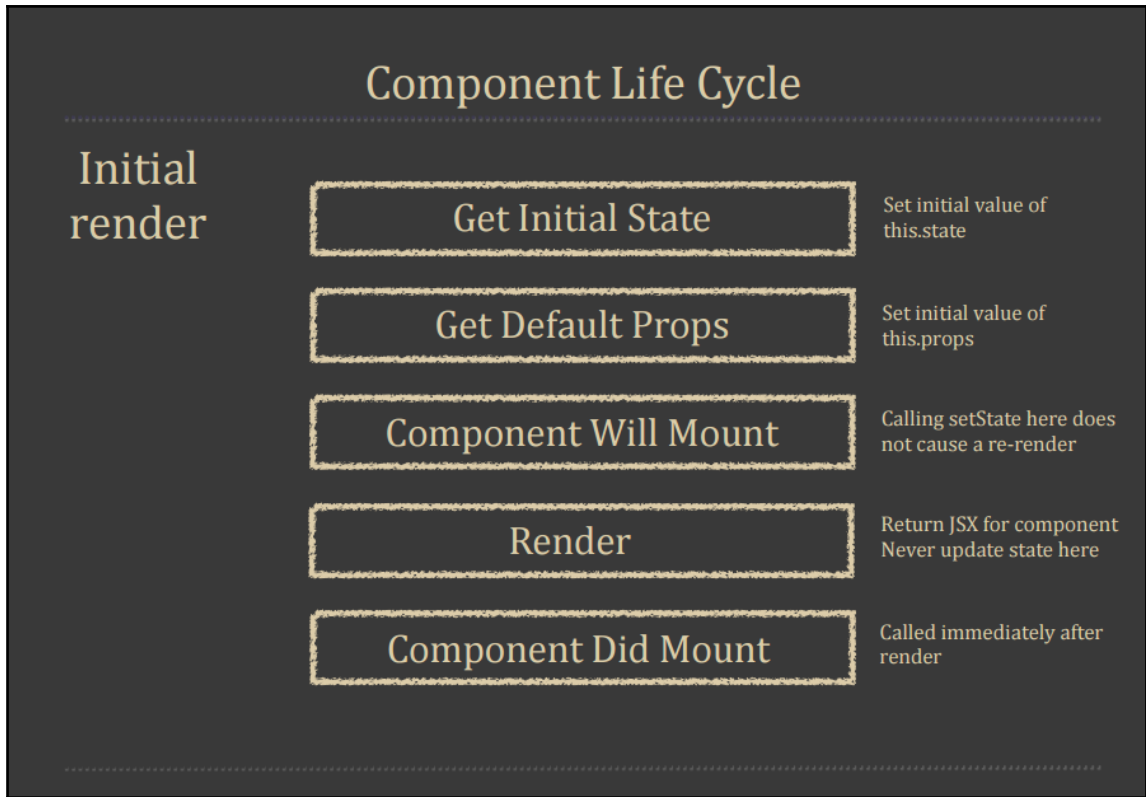
ReactJS

React (sometimes styled React.js or ReactJS) is an open-source JavaScript library which provides a view for data rendered as HTML. Components have been used typically to render React views which contain additional components specified as custom HTML tags. React gives you a trivial virtual DOM, powerful views without templates, unidirectional data flow and explicit mutation. It is very methodical in updating the HTML document when data changes; and a clean separation between components on a modern single-page application.

As your app comes into existence and develops, it's advantageous to ensure that your components are used in right manner and React app consists of Reusable components, which makes code reuse, testing, and separation of concerns easy.

React is not only “V” in MVC, it has stateful components, it handles mapping from input to state changes, and it renders components. In this sense, it does everything that an MVC does.

Let's look at React's component Life Cycle and it's different levels. We will discuss more on that in coming chapters.



A note about React: React isn't an MVC framework it's library for building composable user interface and reusable components. React used at Facebook in production and instagram.com is entirely built on react.

Setting up the environment

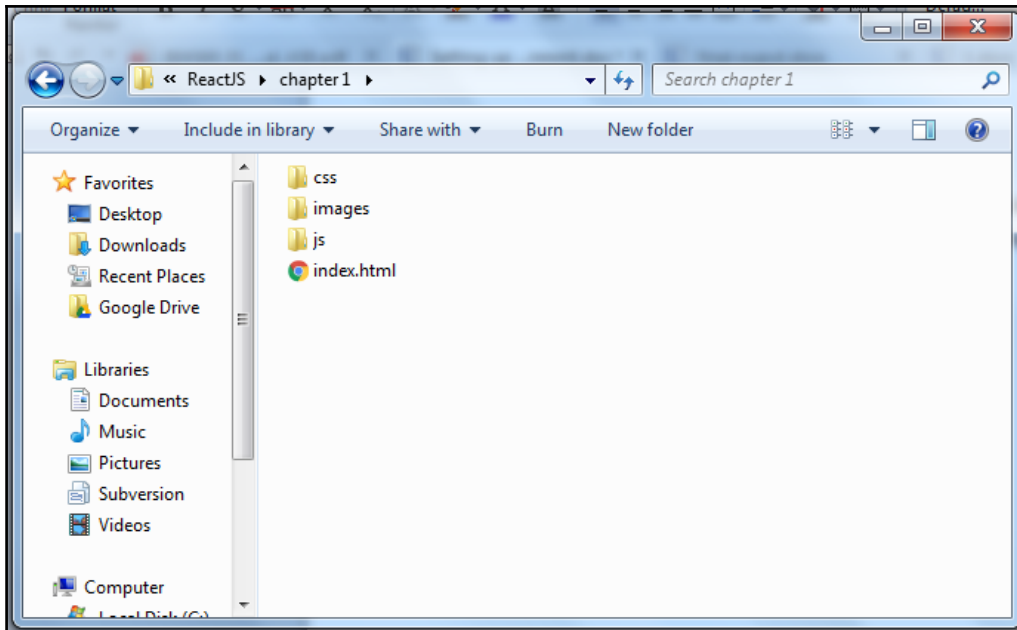
Whilst, we start to make application with ReactJS, we need to do some setup, which just involves HTML page and including a few files. First, we create a directory (folder) called Chapter 1 and open up it in your any favorite code editor. Create a new file called index.html directly inside it and add in this HTML5 Boilerplate code.

```
<!doctype html>
<html class="no-js" lang="">
  <head>
    <meta charset="utf-8">
```

```
<title>ReactJS Chapter 1</title>
</head>
<body>
  <!--[if lt IE 8]>
    <p class="browserupgrade">You are using an <strong>outdated</strong>
browser.
    Please <a href="http://browsehappy.com/">upgrade your browser</a> to
improve
    your experience.</p>
  <![endif]-->
  <!-- Add your site or application content here -->
  <p>Hello world! This is HTML5 Boilerplate.</p>
</body>
</html>
```

This is standard HTML page that we can update with once we have included React and bootstrap libraries.

Now, we need to create couple of folders inside Chapter 1 folder named images, css and js (JavaScript) to make your application manageable. Once you complete with creating folder structure it look like this.

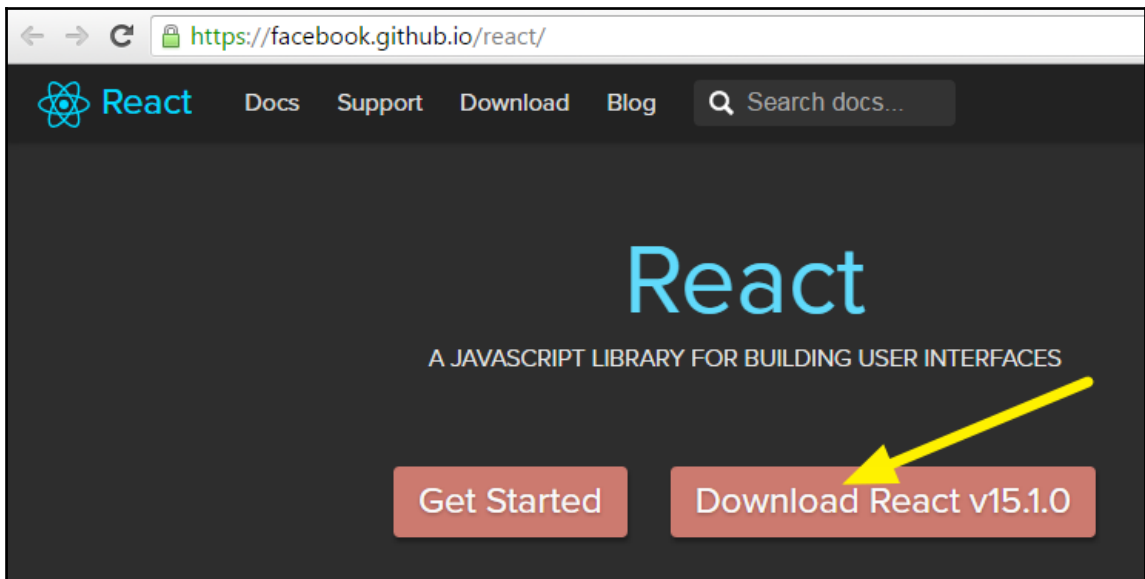


Installing ReactJS and Bootstrap

Once we done with creating folder structure we need to do installation of both of our frameworks reactjs and bootstrap. It's simple as including JavaScript and CSS files in your page. We can do this via content delivery network (CDN) like Google or Microsoft but we are going to fetch the files manually in our application so, we don't have to be dependent on internet while working offline.

Installing React

First, we have to go to this URL <https://facebook.github.io/react/> and hit the download button.



This will give you a zip file of the latest version of reactjs that includes reactjs libraries files and some sample code for reactjs.

For now, we will only need two files at this moment in our application: `react.min.js` and `react-dom.min.js` from the `build` directory of extracted folder.

Here are some few steps we need to follow:

1. Copy it to your project directory Chapter 1/js folder and open up your `index.html` file in your editor.

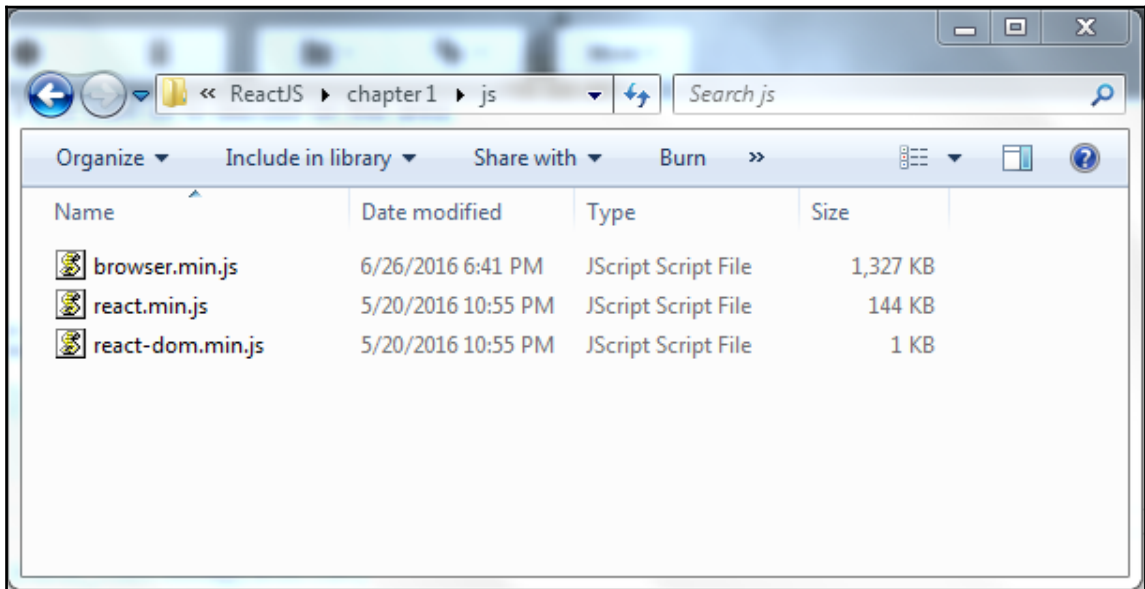
2. Now you just need to add following script in your page head tag section.

```
<script type="text/js" src="js/react.min.js"></script> <script  
type="text/js" src="js/react-dom.min.js"></script>
```

3. Now we need to include the compiler in our project to build the code because right now we are not using any tool like npm. we will download the file from below mentioned CDN path
<https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.23/browser.min.js> or you can give the CDN path directly.
4. Head tag section will be look like:

```
<script type="text/js" src="js/react.min.js"></script>  
<script type="text/js" src="js/react-dom.min.js"></script>  
<script type="text/js" src="js/browser.min.js"></script>
```

Here is your final folder structure of js folder will look alike



Bootstrap

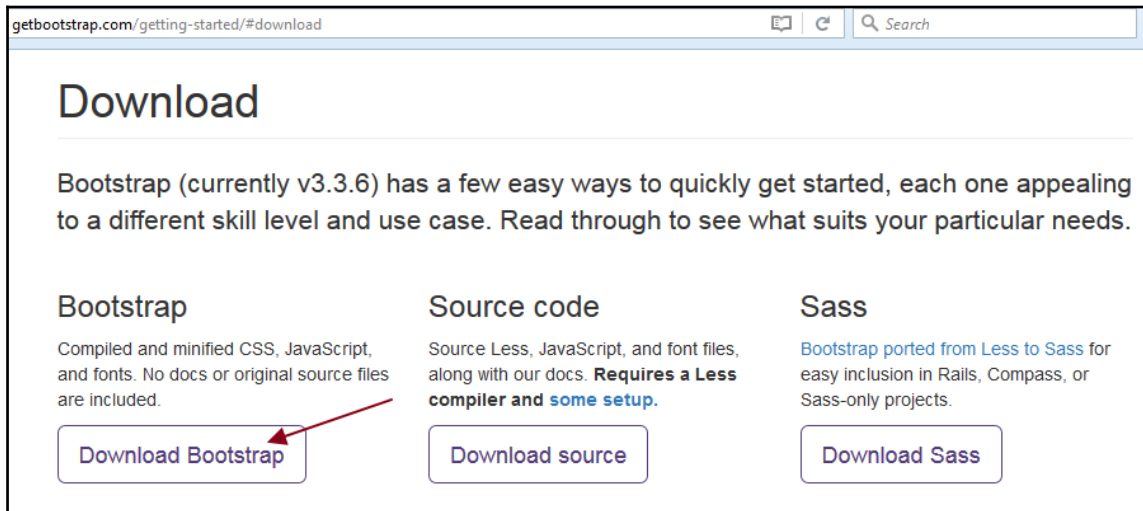
Bootstrap is open-source front-end framework maintained by twitter for developing responsive websites and web application. It includes HTML,CSS and javascript code to

build user interface components. It's faster and easy way to develop powerful mobile first user interface.

Bootstrap grid system allow us to create responsive 12 column grids, layout and components. It includes predefined classes for easy layout options(fixed-width and full width). Bootstrap have pre-styled dozen reusable components and custom jquery plugins like Button, Alerts, Dropdown, Modal, Tooltip Tab, Pagination, Carousel, Badges, icons and many more.

Installing Bootstrap

Now, we need to install bootstrap. Visit <http://getbootstrap.com/getting-started/#download> now hit on download bootstrap button.



It includes the compiled and minified version of css and js for our app we just need css `bootstrap.min.css` and fonts folder. This stylesheet will provide you the look and feel of all components, responsive layout structure for our application. In previous version bootstrap included icons as image but in version 3 they replaced have icons as fonts. We can also customize the bootstrap css stylesheet as per the component using in your application.

1. Extract the zip folder and copy the bootstrap css from css folder to your project folder css
2. Now copy the fonts folder of bootstrap in your project root directory
3. Open your index.html in your editor and add this link tag in your head section.

```
<link rel="stylesheet" href="css/bootstrap.min.css">
```

That's it now we can open up index.html again, but this time in your browser to see what we are working with. Here is the code that we have written so far:

```
<!doctype html>
<html class="no-js" lang="">
  <head>
    <meta charset="utf-8">
    <title>ReactJS Chapter 1</title>
    <link rel="stylesheet" href="css/bootstrap.min.css">

    <script type="text/javascript" src="js/react.min.js"></script>
    <script type="text/javascript" src="js/react-dom.min.js"></script>
    <script src=
"https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.23/browser.min.js">
    </script>
  </head>
  <body>
    <!--[if lt IE 8]>
      <p class="browserupgrade">You are using an <strong>outdated</strong>
browser.
      Please <a href="http://browsehappy.com/">upgrade your browser</a> to
improve
      your experience.</p>
    <![endif]-->
    <!-- Add your site or application content here -->
  </body>
</html>
```

Using React

So now we've got Reactjs and bootstrap stylesheet and there we've initialized our app. Now let's start to write our first “Hello World” app with using the ReactDOM.render().

The first argument of ReactDOM render method is the component we want to render and second is which DOM node it should mount (append) to. ReactDOM.render(ReactElement element, DOMElement container, [function callback]) .

In order to translate it to vanilla JavaScript we use wraps our react code <script type="text/babel"> tag that actually performs the transformation in browser.

Let's start out by putting one div tag in our body tag:

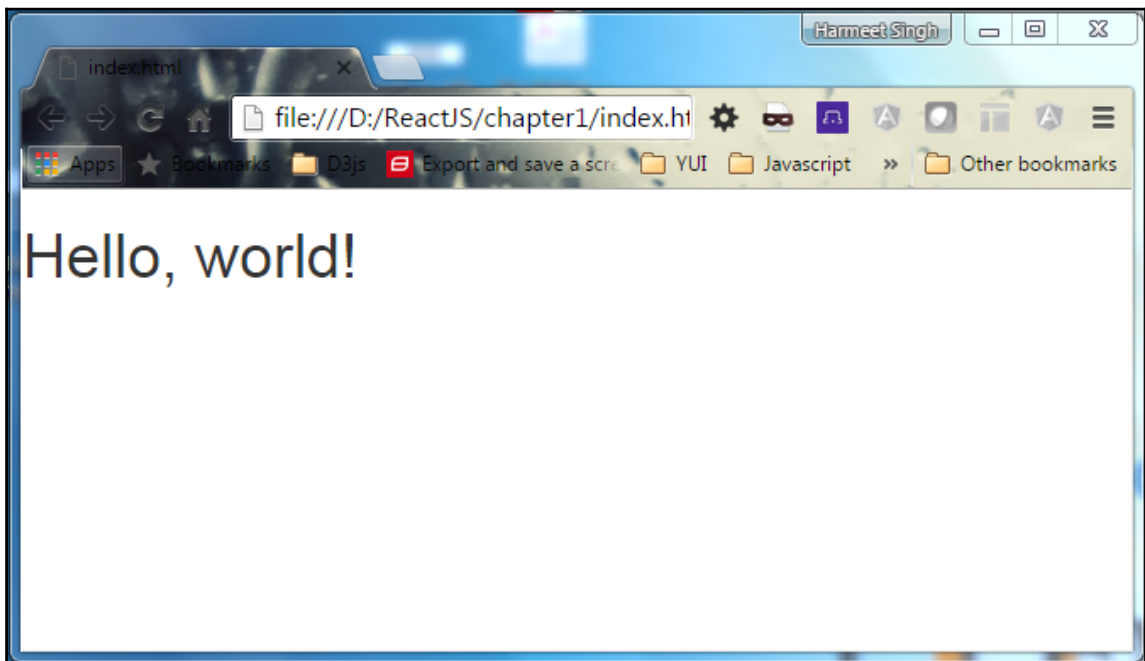
```
<div id="hello"></div>
```

Now, add script tag with react code:

```
<script type="text/babel">
  ReactDOM.render(
    <h1>Hello, world!</h1>,
    document.getElementById('hello')
  );
</script>
```

The XML syntax of JavaScript is called JSX. We will give you the brief about JSX in further chapters.

Let's open your HTML page in your browser if you see “Hello, World!” in your browser then we are on good track.



In above image you can see, it shows the “Hello, World!” in your browser that's great it's amazing we have successfully completed our setup and build our first “hello world” app. Here is our full code what we have written so far now:

```
<!doctype html>
<html class="no-js" lang="">
  <head>
    <meta charset="utf-8">
```

```
<title>ReactJS Chapter 1</title>
<link rel="stylesheet" href="css/bootstrap.min.css">

<script type="text/javascript" src="js/react.min.js"></script>
<script type="text/javascript" src="js/react-dom.min.js"></script>
<script src=
"https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.23/browser.min.js">
</script>
</head>
<body>
  <!--[if lt IE 8]>
    <p class="browserupgrade">You are using an <strong>outdated</strong>
browser.
    Please <a href="http://browsehappy.com/">upgrade your browser</a> to
improve
    your experience.</p>
  <![endif]-->
  <!-- Add your site or application content here -->
  <div id="hello"></div>
  <script type="text/babel">
    ReactDOM.render(
      <h1>Hello, world!</h1>,
      document.getElementById('hello')
    );
  </script>
</body>
</html>
```

Static form with React and Bootstrap

Now, that we have completed our first Hello world app with react and bootstrap and everything is looks good as expected. Not it's time do more and create one static login form and applying bootstrap look and feel on it. Bootstrap is a great way to make you app responsive grid system for different mobile devices and apply the fundamental styled on HTML elements with the inclusion of few classes and div.



Responsive grid system is easy, flexible and quick way to make your web application responsive, Mobile first that appropriately scales up to 12 column as per device and viewport size.

First let's start to make a HTML structure to follow the bootstrap grid system

Create a div and add a class name .container for (fixed width) and .container-fluid for (full-

width). Use `className` attribute instead of using `class`

```
<div className="container-fluid"></div>
```

1. Such as `class` and `for` are discouraged as XML attribute names. Instead of this we can use `className` and `html` for respectively,
2. Create `div` and add the `className` `row`. `Row` must be placed within a `.container-fluid`

```
<div className="container-fluid">  
  <div className="row"></div>  
</div>
```

3. Now create columns that must be immediate children of `row`

```
<div className="container-fluid">  
  <div className="row">  
    <div className="col-lg-6"></div>  
  </div>  
</div>
```

`.row` and `.col-xs-4` are predefined classes that available for quickly making grid layouts. Add the `h1` tag for title of the page:

```
<div className="container-fluid">  
  <div className="row">  
    <div className="col-sm-6">  
      <h1>Login Form</h1>  
    </div>  
  </div>  
</div>
```

Grid columns are created by the given the specify number `Col-sm-*` of twelve available columns for example if we are using 4 column layout we need to specify `col-sm-4` three equal columns.

Col-sm-*	Small Devices
Col-md-*	Medium device
Col-lg-*	Large Devices

We are using `col-sm-*` prefix to resize our columns for small devices. Inside columns we need to wrap our form elements `label` and `input` tag into a `div` tag with the `form-group` class.

```
<div className="form-group">
  <label for="emailInput">Email address</label>
  <input type="email" className="form-control" id="emailInput"
    placeholder="Email"/>
</div>
```

Forget the style of bootstrap we need to add the `form-control` class in our input elements. If we need extra padding on our label tag then we can add `control-label` class on label.

Let's quickly add rest of elements in. I am going to add password and submit button

In previous versions of Bootstrap, form elements would usually have been wrapped in an element with the `form-actions` class. However, in Bootstrap 3, we just need to use the same `form-group` instead of `form-actions`. We will give you more brief about bootstrap classes and responsiveness in our next chapter.

Now, here is our complete HTML code:

```
<div className="container-fluid">
  <div className="row">
    <div className="col-lg-6">
      <form>
        <h1>Login Form</h1><hr/>
        <div className="form-group">
          <label for="emailInput">Email address</label>
          <input type="email" className="form-control" id="emailInput"
            placeholder="Email"/>
        </div>
        <div className="form-group">
          <label for="passwordInput">Password</label>
          <input type="password" className="form-control" id="passwordInput"
            placeholder="Password"/>
        </div>
        <button type="submit" className="btn btn-default col-xs-offset-9
          col-xs-3">Submit</button>
      </form>
    </div>
  </div>
</div>
```

Now create one object inside var `loginFormHTML` script tag and assign this HTML them.

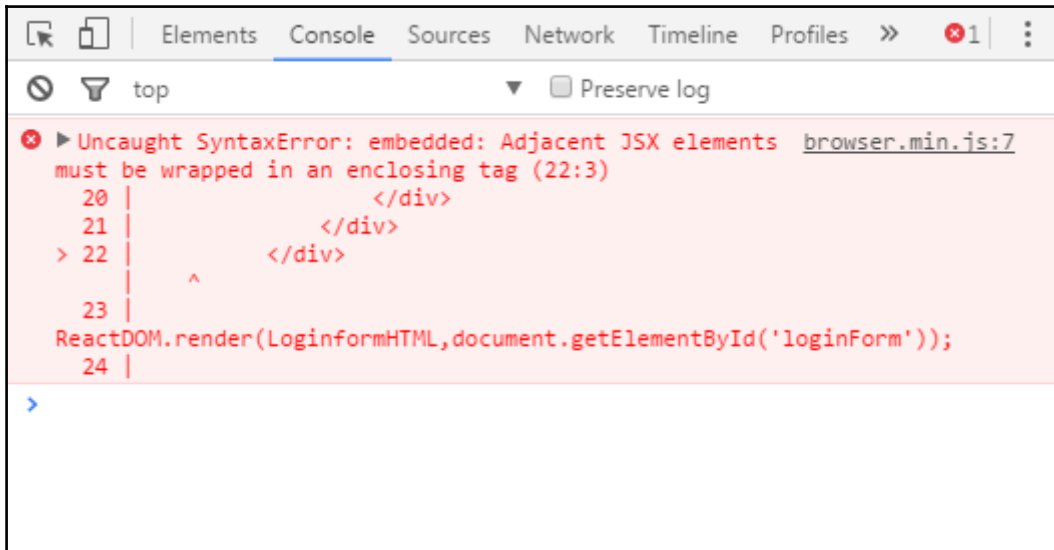
```
Var loginFormHTML = <div className="container-fluid">
  <div className="row">
```

```
<div className="col-lg-6">
  <form>
    <h1>Login Form</h1><hr/>
    <div className="form-group">
      <label for="emailInput">Email address</label>
      <input type="email" className="form-control" id="emailInput"
        placeholder="Email"/>
    </div>
    <div className="form-group">
      <label for="passwordInput">Password</label>
      <input type="password" className="form-control" id="passwordInput"
        placeholder="Password"/>
    </div>
    <button type="submit" className="btn btn-default
      col-xs-offset-9 col-xs-3">Submit</button>
  </form>
</div>
</div>
```

We will pass this object in `ReactDOM()` method instead of passing directly HTML.

```
ReactDOM.render(LoginformHTML, document.getElementById('hello'));
```

Our form is ready now let's see how it looks in browser.



Compiler unable to parse our HTML because we have not enclosed the one of “div” tags properly. If you see in our HTML we have not closed the wrapper “container-fluid” in the

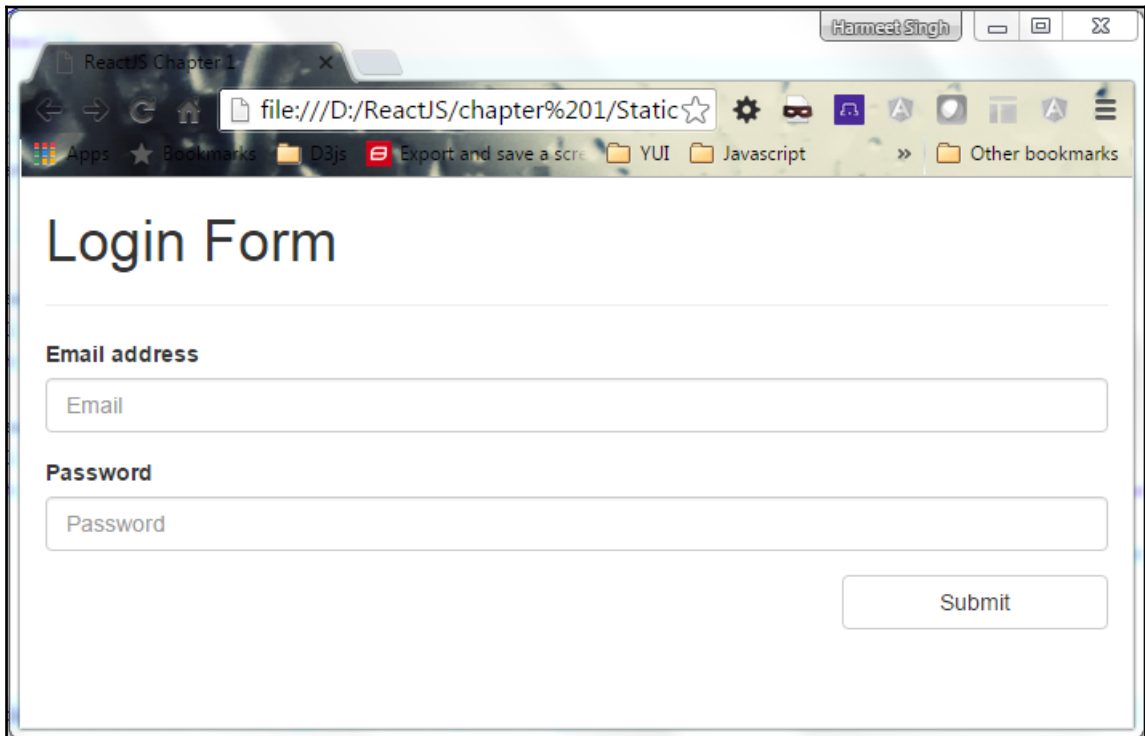
end. Now close the wrapper tag in the end open the file again in your browser.



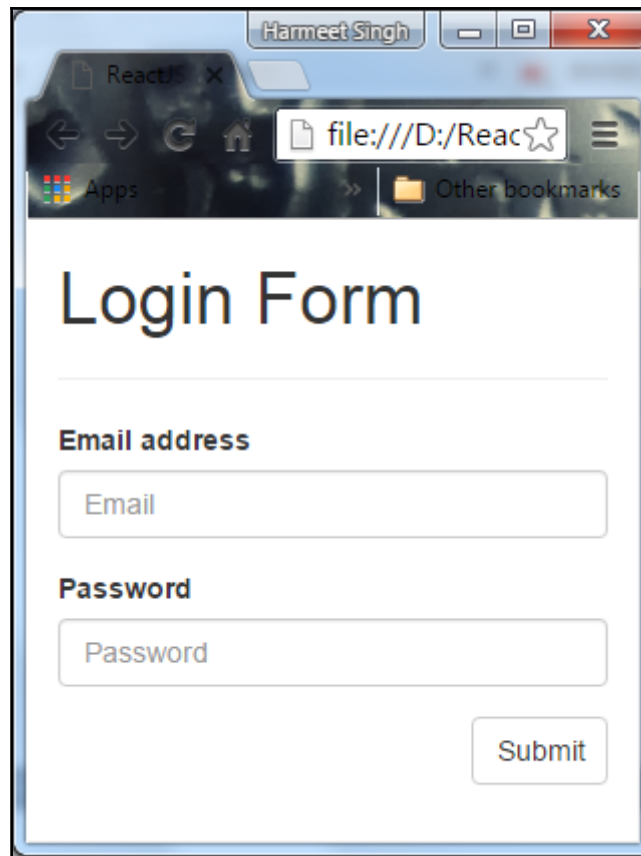
Note: Whenever you hand-code (Write) your HTML code, please double check your “start Tag” and “End Tag”, it should be written / closed properly otherwise it will break your UI / frontend look and feel.

Here is the HTML after closing the “div” tag:

```
<!doctype html>
<html class="no-js" lang="">
  <head>
    <meta charset="utf-8">
    <title>ReactJS Chapter 1</title>
    <link rel="stylesheet" href="css/bootstrap.min.css">
    <script type="text/javascript" src="js/react.min.js"></script>
    <script type="text/javascript" src="js/react-dom.min.js"></script>
    <script src="js/browser.min.js"></script>
  </head>
  <body>
    <!-- Add your site or application content here -->
    <div id="loginForm"></div>
    <script type="text/babel"> var LoginformHTML =
      <div className="container-fluid">
        <div className="row">
          <div className="col-lg-6">
            <form> <h1>Login Form</h1> <hr/>
            <div className="form-group">
              <label for="emailInput">Email address</label>
              <input type="email" className="form-control" id="emailInput"
                placeholder="Email"/>
            </div>
            <div className="form-group">
              <label for="passwordInput">Password</label>
              <input type="password" className="form-control" id="passwordInput"
                placeholder="Password"/>
            </div>
            <button type="submit" className="btn btn-default col-xs-offset-9
              col-xs-3">Submit</button>
          </form>
        </div>
      </div>
    </div>
    ReactDOM.render(LoginformHTML,document.getElementById('loginForm'));
  </script>
</body>
</html>
```



Now it's working fine and look good. Bootstrap also provides the two additional classes to make your elements smaller and larger: `input-lg` and `input-sm`. you can also check the responsive behavior by re-sizing the browser.



That's look great. Our small static login form application is ready with responsive behavior.

Summary

Our simple static login form application and Hello World examples are looking great and working exactly how it should, so let's recaps what we've learn't in the first chapter.

To begin with, we saw just how easy it is to get reactjs and Bootstrap installed with the inclusion of JavaScript files and stylesheet. We also looked at how react application is initialized and started building our first form application.

The “Hello world!” and Form application we have created, demonstrates some of React's and bootstrap's basic features:

- ReactDOM
- Render
- Browserify
- Bootstrap

With Bootstrap, we utilized to have responsive grid system for different mobile devices and apply the fundamental styled on HTML elements with the inclusion of few classes and div.

We also saw the framework's new mobile first responsive design in action without cluttering up our markup with unnecessary classes or elements.

In *Chapter 2, Let's Build Responsive Theme with Twitter Bootstrap*, delve into its features and how to use Grid. We are going to explore some more Bootstrap fundamentals and introduce the project we are going to build over the course of this book.

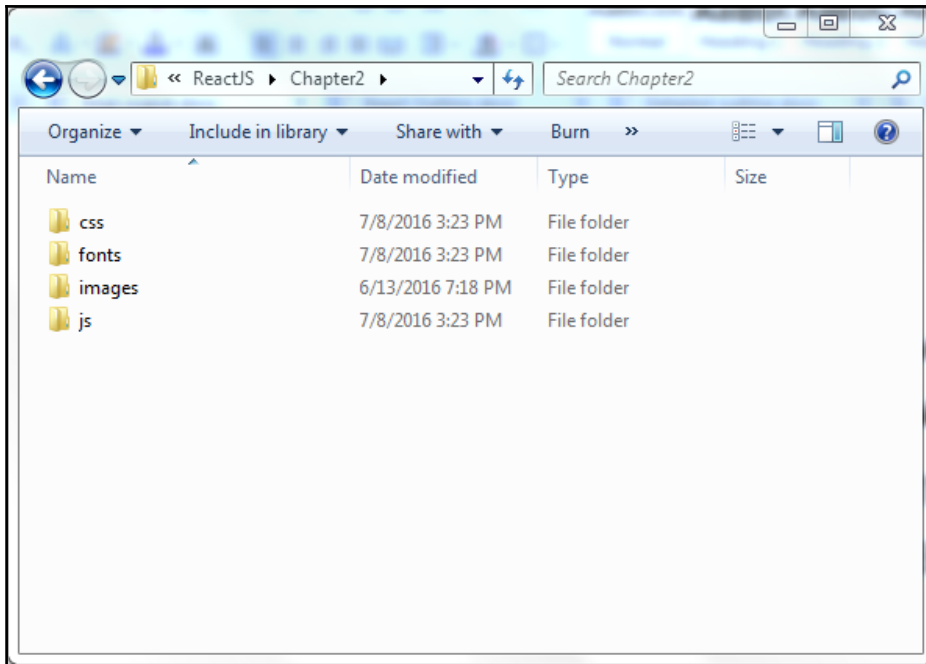
2

Lets build a responsive theme with React-Bootstrap and React

Now, that you've completed your first web app using ReactJS and Bootstrap, we're going to build a first responsive theme for your app using both the frameworks. We'll also be touching the full potential of both frameworks. So, let's start!

Setting up

Firstly, we need to create a similar folder structure as our “Hello World” app which we have made in chapter1. Please find following image which describes folder structure:



Now you need to copy ReactJS and Bootstrap files from chapter1 into the significant directories of chapter2 and create an index.html file in the root. The following code snippet is just a base HTML page which includes bootstrap and reacts.

Here is markup of our HTML page.

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>ReactJS theme with bootstrap</title>
    <link rel="stylesheet" href="css/bootstrap.min.css">

    <script type="text/javascript" src="js/react.min.js"></script>
    <script type="text/javascript" src="js/react-dom.min.js"></script>
    <script src="js/browser.min.js"></script>
  </head>

  <body>

  </body>
</html>
```

Scaffolding

So now we have base file and folder structure sorted, next step would be, we can start to scaffold out our app using bootstrap CSS.

I'm sure you have a question, what is Scaffolding? In simple sense, it gives support structure to make your base concrete.

Apart from this, we will use React-Bootstrap JS in which we have collection of bootstrap components rebuilt for react, which we can use throughout our EIS (Employee information system). Bootstrap also includes an extremely powerful responsive grid system which helps us to create responsive theme layout / template / structure for app.

Navigation

Navigation is very important element of any static or dynamic page. So now, we are going to build a **“navbar” (navigation)** to switch between each of our page. It could be placed at the top of our page.

Here, is the basic HTML structure of bootstrap navigation:

```
<nav className="navbar navbar-default navbar-static-top" role="navigation">
  <div className="container">
    <div className="navbar-header">
      <button type="button" className="navbar-toggle" data-toggle="collapse"
        data-target=".navbar-collapse">
        <span className="sr-only">Toggle navigation</span>
        <span className="icon-bar"></span>
        <span className="icon-bar"></span>
        <span className="icon-bar"></span>
      </button>
      <a className="navbar-brand" href="#">EIS</a>
    </div>
    <div className="navbar-collapse collapse">
      <ul className="nav navbar-nav">
        <li className="active"><a href="#">Home</a></li>
        <li><a href="#">Edit Profile</a></li>
        <li className="dropdown">
          <a href="#" className="dropdown-toggle" data-toggle="dropdown">
            Help Desk <b className="caret"></b></a>
          <ul className="dropdown-menu">
            <li><a href="#">View Tickets</a></li>
            <li><a href="#">New Ticket</a></li>
          </ul>
        </li>
      </ul>
    </div>
  </div>
</nav>
```

```
        </li>
      </ul>
    </div>
  </div>
</nav>
```

The `<nav>` tag that used to hold everything within navbar, is instead split into two sections: `navbar-header` and `navbar-collapse`, if you see the navigation structure. Navbars are responsive component so `navbar-header` element is exclusively for mobile navigation and control the expand and collapse the navigation with toggle button. The `data-target` attribute on the button directly corresponds with the `id` attribute of the `navbar-collapse` element so Bootstrap knows what element should be wrap in mobile devices to control the toggling.

Now we also need to include jQuery in your page because bootstrap's JS have dependency on it. You can get the latest jquery version from <http://jquery.com/>. Now you need to copy `bootstrap.min.js` from bootstrap extracted folder and add this to your app js directory, as well as include it on your page before `bootstrap.min.js`.

Please make sure, your JavaScript files are included in following order:

```
<script type="text/javascript" src="js/react.min.js"></script>
<script type="text/javascript" src="js/react-dom.min.js"></script>
<script src="js/browser.min.js"></script>
<script src="js/jquery-1.10.2.min.js"></script>
<script src="js/bootstrap.min.js"></script>
```

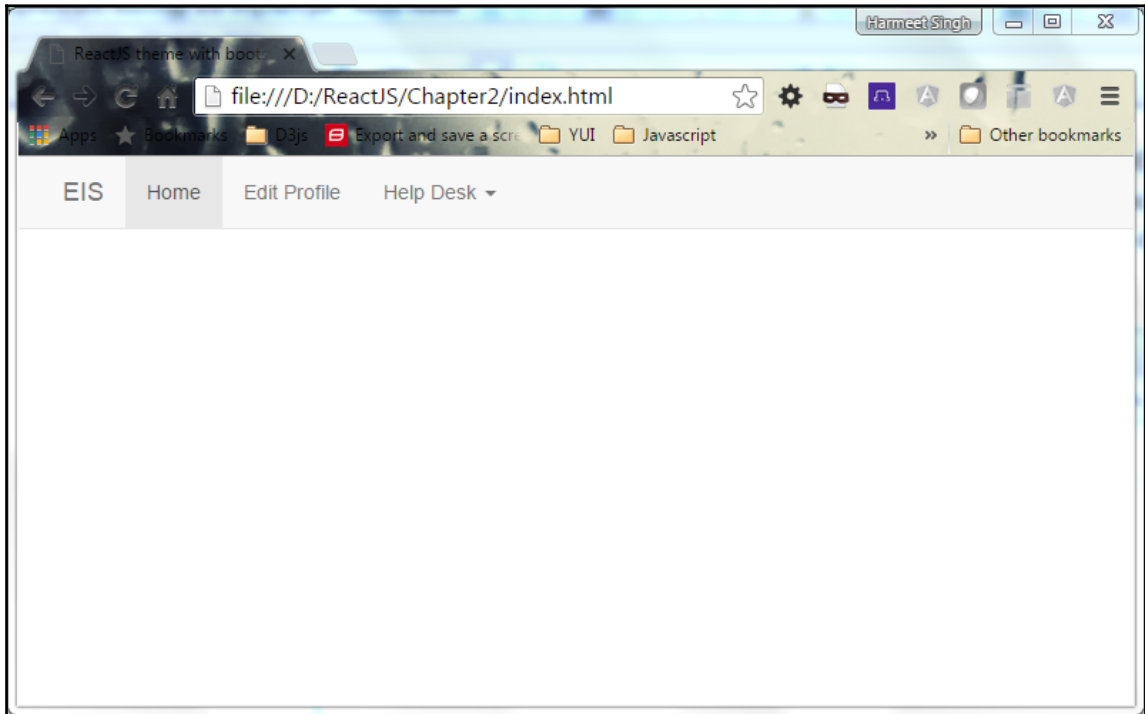
Let's take a quick look of navbar component code after integrating in react:

```
<div id="nav"></div>
<script type="text/babel">
  var navbarHTML = <nav className="navbar navbar-default navbar-static-top"
role="navigation">
  <div className="container">
    <div className="navbar-header">
      <button type="button" className="navbar-toggle"
        data-toggle="collapse" data-target=".navbar-collapse">
        <span className="sr-only">Toggle navigation</span>
        <span className="icon-bar"></span>
        <span className="icon-bar"></span>
        <span className="icon-bar"></span>
      </button>
      <a className="navbar-brand" href="#">EIS</a>
    </div>
    <div className="navbar-collapse collapse">
```



```
<ul className="nav navbar-nav">
  <li className="active"><a href="#">Home</a></li>
  <li><a href="#">Edit Profile</a></li>
  <li className="dropdown">
    <a href="#" className="dropdown-toggle" data-toggle="dropdown">
      Help Desk <b className="caret"></b></a>
    <ul className="dropdown-menu">
      <li><a href="#">View Tickets</a></li>
      <li><a href="#">New Ticket</a></li>
    </ul>
  </li>
</ul>
</div>
</div>
</nav>
ReactDOM.render(NavbarHTML, document.getElementById('nav'));
</script>
```

Open your index.html in your browser to see the navbar component. The following screenshot shows how our navigation will look alike.



We have included navigation directly within our <body> tag for cover the full width of the browser. Now we do the same thing with using React-Bootstrap js framework to know the difference between bootstrap js and React-bootstrap js.

React-Bootstrap

React-bootstrap JavaScript framework is similar like bootstrap rebuilt for React. It's a complete re-implementation of the bootstrap front-end reusable components in react. React-Bootstrap has no dependency on any other framework like bootstrap.js or jquery. It means if you are using react-bootstrap then we don't need to include the jquery in your project as a dependency. Using React-Bootstrap we can assure that, there won't be external JavaScript calls to render the component which might be incompatible with the react DOM render but still you can achieve the same functionality and look and feel like twitter bootstrap, but with much cleaner code with React.

Installing React-Bootstrap

To get this react-bootstrap either we can use CDN directly <https://cdnjs.cloudflare.com/ajax/libs/react-bootstrap/0.29.5/react-bootstrap.min.js> or open this URL and save it in your local for fast performance. When you download file in your local, please make sure to download the source-map (react-bootstrap.min.js.map) file along with them to make debugging much easier. Once you are done with download, add that library in your app's js directory and include in your page head section as shown below.

Your head section will be look like this:

```
<script type="text/javascript" src="js/react.min.js"></script>
<script type="text/javascript" src="js/react-dom.min.js"></script>
<script src="js/browser.min.js"></script>
<script src="js/react-bootstrap.min.js"></script>
```

Using React-bootstrap

Now, you must have question like once we have bootstrap file already and we are also adding React-bootstrap js file then it won't conflict with each other? No, it will not. As React-bootstrap is compatible with existing bootstrap style so we don't need to worry about any conflict.

Now we are going to create same “navbar” component in react-bootstrap.

Here, is the structure of “navbar” component in react-bootstrap

```
var Nav= ReactBootstrap.Nav;
var Navbar= ReactBootstrap.Navbar;
var NavItem= ReactBootstrap.NavItem;
var NavDropdown = ReactBootstrap.NavDropdown;
var MenuItem= ReactBootstrap.MenuItem;
var navbarReact =(
  <Navbar>
    <Navbar.Header>
      <Navbar.Brand>
        <a href="#">EIS</a>
      </Navbar.Brand>
      <Navbar.Toggle />
    </Navbar.Header>
    <Navbar.Collapse>
      <Nav>
        <NavItem eventKey={1} href="#">Home</NavItem>
        <NavItem eventKey={2} href="#">Edit Profile</NavItem>
        <NavDropdown eventKey={3} title="Help Desk"
          id="basic-nav-dropdown">
          <MenuItem eventKey={3.1}>View Tickets</MenuItem>
          <MenuItem eventKey={3.2}>New Ticket</MenuItem>
        </NavDropdown>
      </Nav>
    </Navbar.Collapse>
  </Navbar>
);
```

Here is enlightenment of the above code: (order changed from below of benefits section to above of it)

<Navbar> tag is container of the component and it splits into two sections **<Navbar.header>** and **<Nav>**.

For responsive behavior we have added the **<Navbar.Toggle/>** tag that controls expand and collapse and wrap the **<Nav>** into the **<Navbar.Collapse>** to show and hide the nav items.

For capture the event we have used **eventKey={1}** when we select any menu item, a callback fired which takes two argument (**eventKey: any, event: Object**) => **any**.

Benefits of React – Bootstrap

Let's check out benefit of using React-bootstrap.

If you can see in above code, it looks cleaner than the twitter bootstrap component because we can import the individual component from react bootstrap rather than including the entire library. By doing this, it pulls only specific component that we want to include and helps to reduce your app bundle size significantly.

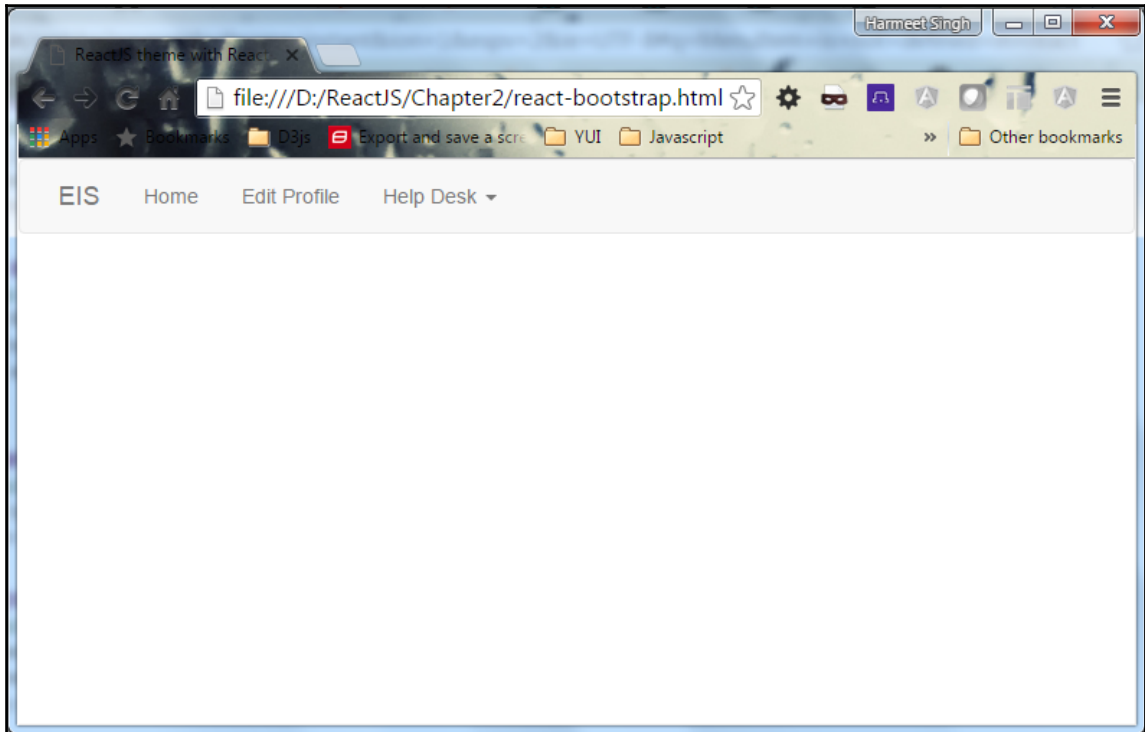
- It saves a bit of typing and bugs by compressing the bootstrap.
- It reduces typing efforts and more importantly conflicts by compressing the bootstrap.
- We don't need to think different approaches taken by Bootstrap vs. React
- Easy to use
- Encapsulation in elements
- JSX Syntax
- Avoid React rendering of virtual DOM
- Easy to detect DOM changes and update the DOM without any conflict
- Don't have any dependency on other library like jQuery

Here, is the full code view of our “navbar” component:

```
<div id="nav"></div>
<script type="text/babel">
  var Nav= ReactBootstrap.Nav;
  var Navbar= ReactBootstrap.Navbar;
  var NavItem= ReactBootstrap.NavItem;
  var NavDropdown = ReactBootstrap.NavDropdown;
  var MenuItem= ReactBootstrap.MenuItem;
  var navbarReact =(
    <Navbar>
      <Navbar.Header>
        <Navbar.Brand>
          <a href="#">EIS</a>
        </Navbar.Brand>
        <Navbar.Toggle />
      </Navbar.Header>
      <Navbar.Collapse>
        <Nav>
          <NavItem eventKey={1} href="#">Home</NavItem>
          <NavItem eventKey={2} href="#">Edit Profile</NavItem>
          <NavDropdown eventKey={3} title="Help Desk"
            id="basic-nav-dropdown">
            <MenuItem eventKey={3.1}>View Tickets</MenuItem>
            <MenuItem eventKey={3.2}>New Ticket</MenuItem>
          </NavDropdown>
        </Nav>
      </Navbar.Collapse>
    </Navbar>
  );
</script>
```

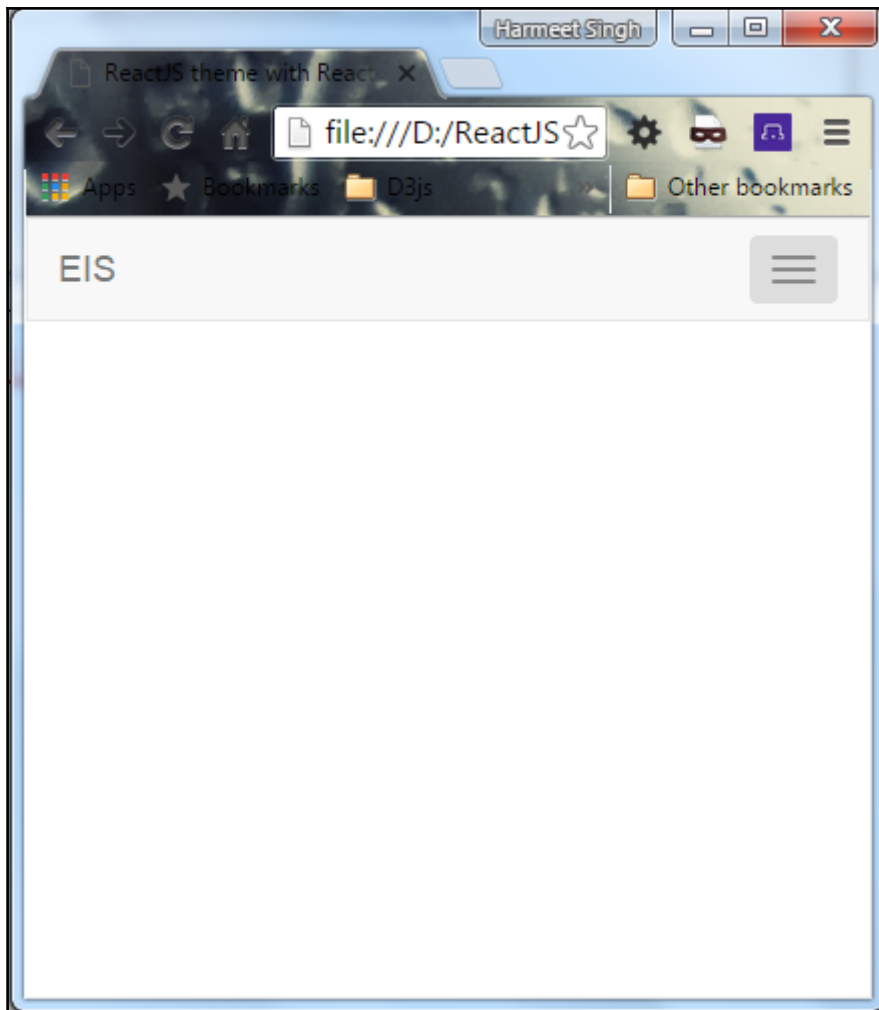
```
    </Navbar>
  );
  ReactDOM.render(navbarReact, document.getElementById('nav'));
```

Woohoo!!! Let's take a look of our first react-bootstrap component in browser. The following screenshot shows how component will look alike,



Now to check “navbar”, if you resize your browser window, you'll notice bootstrap display the mobile header with the toggle button below 768px screen size of tablet in portrait mode. However, if you click the button to toggle the navigation, you can see the navigation for mobile.

The following screenshot of how mobile navigation will look,



So now we have major understanding about React-Bootstrap and Bootstrap. React-Bootstrap has active development efforts being put in to keep it updated.

Bootstrap Grid System

Bootstrap is based on 12 column grid system which includes a powerful responsive structure and mobile first fluid grid system that allows us to scaffold our web app with very few elements. In bootstrap, we have predefined series of classes to compose of rows and columns, so before we start we need to include `<div>` tag with "container" class to wrap our rows and columns otherwise framework won't respond as expected as bootstrap has

written CSS which is dependent on it which we need to add below our navbar:

```
<div class="container"><div>
```

This will make your web app center of the page as well as control the rows and columns to work as expected in responsive.

There are four class prefixes which helps to define behavior of columns. All the classes are related to different device screen sizes and react in familiar way. The following table is from <http://getbootstrap.com/> to define variations between all four classes.

	Extra small devices Phones (<768 px)	Small devices Tablets (≥768 px)	Medium devicesDesktops (≥992 px)	Large devices Desktops (≥1200 px)
Grid behavior	Horizontal at all times	Collapsed to start, horizontal above breakpoints		
Container width	None (auto)	750 px	970 px	1170 px
Class prefix	.col-xs-	.col-sm-	.col-md-	.col-lg-
# of columns	12			
Column width	Auto	~62 px	~81 px	~97 px
Gutter width	30 px (15 px on each side of a column)			
Nestable	Yes			
Offsets	Yes			
Column ordering	Yes			

In our application we need to create two columns layout for main content area and sidebar. As we know bootstrap has 12 columns grid layout so divide your content in a way which covers whole area.



Note: Please understand, bootstrap divides 12 column grid by using col-*-1 to col-*-12 classes.

We'll divide 12 columns into two parts one is 9 columns for main content and 3 columns for sidebar, sounds perfect. So, here is the way to implement that.

First we need to include `<div>` tag inside our container and add the class “row”. We can have as many div tags with “row” class as per design need which can each house up to twelve columns.

```
<div class="container">
  <div class="row">
  </div>
</div>
```

As we all know if we want our columns to stack on mobile devices, we should use col-sm- prefixes. Creating a column is as simple as taking the desired prefix and appending the number of columns as you wish it to add.

Let's take a quick look at how we can create a 2 column layout:

```
<div class="container">
  <div class="row">
    <div class="col-sm-3">
      Column Size 3 for smaller devices
    </div>
    <div class="col-sm-9">
      Column Size 9 for smaller devices
    </div>
  </div>
</div>
```

If we want our columns to stack not only for smaller devices, Use the extra small and medium grid classes by adding col-md-* and col-xs* to your columns.

```
<div class="container">
  <div class="row">
    <div class="col-xs-12 col-md-4">
```

Columns in mobile one full-width and the other half-width

```
    </div>
    <div class="col-xs-12 col-md-8">
```

Columns in mobile one full-width and the other half-width

```
    </div>
  </div>
</div>
```

So when it displays in larger screen than a mobile device, Bootstrap will automatically add

30px gutter spacing (space between two elements) between each column (15px on either side). If we want to add additional space between the columns bootstrap will provides a way to do this just adding the additional class to the column.

```
<div class="container">
  <div class="row">
    <div class="col-xs-12 col-md-7 col-md-offset-1">
```

Columns in mobile one full-width and the other half-width with more space from left

```
    </div>
  </div>
</div>
```

So this time we have used `offset` keyword. The number of end controls the number of columns you want to offset.



Remember: Offset column count as equal of total number 12 column in row.

Now, let's create some complex layout with nested additional rows and columns.

```
<div class="row">
  <div class="col-sm-9">
    Level 1 - Lorem ipsum...
    <div class="row">
      <div class="col-xs-8 col-sm-4">
        Level 2 - Lorem ipsum...
      </div>
      <div class="col-xs-4 col-sm-4">
        Level 2 - Lorem ipsum...
      </div>
    </div>
  </div>
</div>
```

If open it up in your browser you will see this will create two columns within our main content container “col-sm-9” which we have created earlier. However, as our grid is nested, we can create a new row and have a single column or two columns whatever your layout requires. I have added some dummy text to demonstrate the nested columns.

Bootstrap will also provide the option to change the ordering of the columns in grid system by using the `col-md-push-*` and `col-md-pull-*` classes.

```
<div class="row">
```

```
<div class="col-sm-9">
  Level 1 - Lorem ipsum...
  <div class="row">
    <div class="col-xs-8 col-sm-4 col-sm-push-4">
      Level 2 - col-xs-8 col-sm-4 col-sm-push-4
    </div>
    <div class="col-xs-4 col-sm-4 col-sm-pull-4">
      Level 2 - col-xs-8 col-sm-4 col-sm-pull4
    </div>
  </div>
</div>
</div>
```

Level 1 - Lorem ipsum...		
Level 2 - col-xs-4 col-sm-4 col-sm-pull-4	Level 2 - col-xs-8 col-sm-4 col-sm-push-4	

Bootstrap also includes some pre-defined classes to enable elements to be shown or hidden at specific screen sizes. The classes use the same pre-defined sizes as Bootstrap's grid.

For example, the following will hide an element at a specific screen size:

```
<div class="hidden-md"></div>
```

This will hide the element on medium devices but it will still be visible on mobiles, tablets, and large desktops. To hide an element on multiple devices, we need to use multiple classes:

```
<div class="hidden-md hidden-lg"></div>
```

Likewise, same with the visible classes work in reverse, showing elements at specific sizes.

However, unlike the hidden classes, they also require us to set the display value. This can be block, inline, or inline-block:

```
<div class="visible-md-block"></div>
<div class="visible-md-inline"></div>
<div class="visible-md-inline-block"></div>
```

Of course, we can use the various classes in one element. If, for example, we wanted a block-level element on a smaller screen but have it become inline-block later, we would use the following code:

```
<div class="visible-sm-block visible-md-inline-block"></div>
```

If you can't remember the various class sizes, be sure to take another look at the Getting to Bootstrap's grid section to know the screen sizes.

Helper Classes

Bootstrap also includes a few helper classes that we can use to adapt our layout. Let's take a look at some examples.

Floats

Floating classes of bootstrap will help you to create decent layout on the web. Here, are two bootstrap classes to pull your elements left and right.

```
<div class="pull-left">...</div>
<div class="pull-right">...</div>
```

When we are using floats on elements, we need to wrap our floated elements in a `clearfix` class. This will clear the elements and you can able to see actual height of container element:

```
<div class="helper-classes">
<div class="pull-left">...</div>
<div class="pull-right">...</div>
<div class="clearfix">
</div>
```

If the float classes are directly within an element with the row class, then our floats are cleared automatically by Bootstrap and the `clearfix` class does not need to be applied manually.

Center elements

To make it center block-level elements. Bootstrap allows us to do this with the `center-block` class:

```
<div class="center-block">...</div>
```

This will make your element property `margin-left` and `margin-right` properties to `auto`, which will center the element.

Show and hide

You may wish to show and hide elements with CSS, and Bootstrap gives you a couple of classes to do this:

```
<div class="show">...</div>
<div class="hidden">...</div>
```



Important: The show class sets the display property to block, so only apply this to block-level elements and not elements you wish to be displayed inline or inline-block.

React Components

React is basically based on build modular, encapsulated components that manage their own state so it will efficiently update and render your components when data changes. In react, components logic is written in JavaScript instead of templates so you can easily pass rich data through your app and manage the state out of the DOM.

Using `.render()` method we are rendering a component in react that takes input data and return what you want to display. It can either take HTML tag (strings) or React components (Classes).

Let's take quick look on examples of both

```
var myReactElement = <div className="hello" />;
ReactDOM.render(myReactElement, document.getElementById('example'));
```

In this example, we are passing HTML as a string into render method which we have used before creating `<Navbar>`.

```
var ReactComponent = React.createClass({/*...*/});
var myReactElement = <ReactComponent someProperty={true} />;
ReactDOM.render(myReactElement, document.getElementById('example'));
```

In above example, we are rendering component, just to create a local variable that starts with an upper-case. Using the upper vs. lower case convention in React's JSX it will distinguish between local component classes and HTML tags.

So, we can create our React elements or components with two ways either we can use Plain JavaScript with `React.createElement` or React's JSX.

So let's create our sidebar elements for app to make better understanding of

React.createElement.

React.createElement()

Using JSX in React is completely optional for creating react app. As we know we can create elements with React.createElement which take three argument tag name or component, a properties object and variable number of child elements which is optional.

```
var profile = React.createElement('li',{className:'list-group-item'},
'Profile');

var profileImageLink = React.createElement('a',{className:'center-block
text-center',href:'#'},'Image');

var profileImageWrapper = React.createElement('li',{className:'list-group-
item'}, profileImageLink);

var sidebar = React.createElement('ul', { className: 'list-group' },
profile, profileImageWrapper);

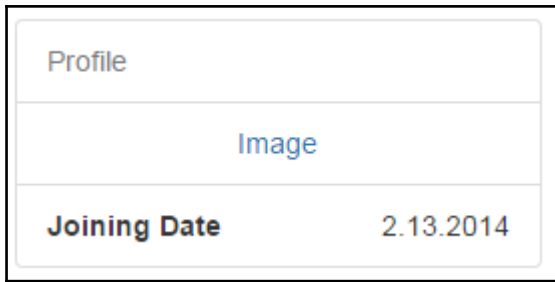
ReactDOM.render(sidebar, document.getElementById('sidebar'));
```

In above example we have used `React.createElement` to generate ul li structure. React already has built-in factories for common DOM HTML tags.

Here is the example for this:

```
var Sidebar = React.DOM.ul({ className: 'list-group' },
  React.DOM.li({className:'list-group-item text-muted'},'Profile'),
  React.DOM.li({className:'list-group-item'},
    React.DOM.a({className:'center-block text-center',href:'#'},'Image')
  ),
  React.DOM.li({className:'list-group-item text-right'},'2.13.2014',
    React.DOM.span({className:'pull-left'},
      React.DOM.strong({className:'pull-left'},'Joining Date')
    ),
    React.DOM.div({className:'clearfix'})
  )
);
ReactDOM.render(Sidebar, document.getElementById('sidebar'));
```

Let's take a quick look of our code in browser which should resemble below image:



Here is our full code so far what we have written to include `<Navbar>` component,

```
<script type="text/babel">
var Nav= ReactBootstrap.Nav;
var Navbar= ReactBootstrap.Navbar;
var NavItem= ReactBootstrap.NavItem;
var NavDropdown = ReactBootstrap.NavDropdown;
var MenuItem= ReactBootstrap.MenuItem;
var navbarReact =(
<Navbar>
  <Navbar.Header>
    <Navbar.Brand>
      <a href="#">EIS</a>
    </Navbar.Brand>
    <Navbar.Toggle />
  </Navbar.Header>
  <Navbar.Collapse>
    <Nav>
      <NavItem eventKey={1} href="#">Home</NavItem>
      <NavItem eventKey={2} href="#">Edit Profile</NavItem>
      <NavDropdown eventKey={3} title="Help Desk" id="basic-nav-dropdown">
        <MenuItem eventKey={3.1}>View Tickets</MenuItem>
        <MenuItem eventKey={3.2}>New Ticket</MenuItem>
      </NavDropdown>
    </Nav>
  </Navbar.Collapse>
</Navbar>
);
ReactDOM.render(navbarReact,document.getElementById('nav'));

var Sidebar = React.DOM.ul({ className: 'list-group' },
  React.DOM.li({className:'list-group-item text-muted'}, 'Profile'),
  React.DOM.li({className:'list-group-item'},
    React.DOM.a({className:'center-block text-center',href:'#'}, 'Image')
  ),
  React.DOM.li({className:'list-group-item text-right'}, '2.13.2014',
    React.DOM.span({className:'pull-left'},
      React.DOM.strong({className:'pull-left'}, 'Joining Date')
```

```
),
  React.DOM.div({className: 'clearfix'})
));
ReactDOM.render(Sidebar, document.getElementById('sidebar'));

</script>
<div id="nav"></div>
<div class="container">
<hr>
<div class="row">
<div class="col-sm-3" id="sidebar"><!--left col-->

</div><!--/col-3-->
<div class="col-sm-9 profile-desc"></div><!--/col-9-->
</div>
</div><!--/row-->
```

Our app code looks very messy. Now it's time to make our code clean and structured properly.

Copy the navbar code in another file and save as `navbar.js`

Now copy the sidebar code in another file and save as `sidebar.js`

Create one folder in your root directory with the name of components and copy both `navbar.js` and `sidebar.js` inside it.

And include both js file in your head section.

Head section will be look alike,

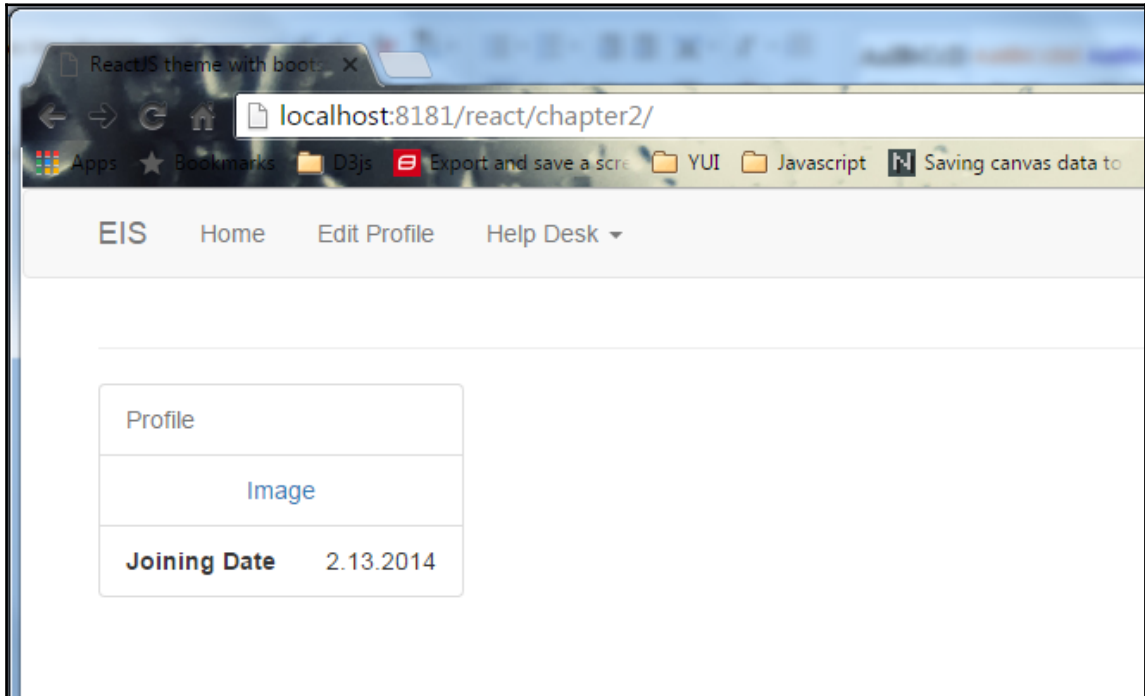
```
<script type="text/javascript" src="js/react.min.js"></script>
<script type="text/javascript" src="js/react-dom.min.js"></script>
<script src="js/browser.min.js"></script>
<script src="js/jquery-1.10.2.min.js"></script>
<script src="js/react-bootstrap.min.js"></script>
<script src="components/navbar.js" type="text/babel"></script>
<script src="components/sidebar.js" type="text/babel"></script>
```

And here is your HTML:

```
<div id="nav"></div>
<div class="container">
  <hr>
  <div class="row">
    <div class="col-sm-3" id="sidebar"><!--left col-->
```

```
</div><!--/col-3-->
<div class="col-sm-9 profile-desc"></div><!--col-9-->
</div>
</div><!--/row-->
```

Now our code looks cleaner. Let's take a quick look of your code output in your browser.



Note: when we are referring Reactjs file from external then we need web server or full stack app like WAMP or XAMP because some browsers (Chrome, e.g.) will fail to load the file unless it's served via HTTP.

Summary

We have good amount of basic knowledge of Bootstrap and React-bootstrap in this chapter, so let's quickly check out what we have learnt.

While going through definition and use of bootstrap and React-bootstrap we saw that React-bootstrap is very strong candidate with more flexibility and smart solution.

We have seen that how we can create mobile navigation by just using few nice features of Bootstrap and React-bootstrap which works well on all expected devices as well as on desktop browsers.

We also looked at the powerful responsive grid system including Bootstrap and created a simple two column layout. While we were doing this, we learnt about the four different column class prefixes as well as nesting our grid.

We have also seen some very good features of bootstrap like offset, col-md-push-*, col-md-pull-*, hidden-md, hidden-lg, visible-sm-block, visible-md-inline-block, Helper Classes etc.

Hope you are also ready with your responsive layout and navigation. So now let's jump on to the next chapter.

3

ReactJS - JSX

What is JSX in React

JSX is extension of JavaScript syntax and if you observe the syntax or structure of JSX, you will find it similar to XML coding.

JSX is doing pre-processor footstep which adds XML syntax to JavaScript. Though, you can certainly use React without JSX but JSX makes react a lot more neat and elegant. Similar like XML, JSX tags are having tag name, attributes, and children and in that if an attribute value is enclosed in quotes that value becomes a string.

The way XML is working with balanced opening and closing tags, JSX works similarly and it also helps to understand and read huge amount of structures easily than JavaScript functions and objects.

Advantages of using JSX in React

1. JSX is very simple to understand and think about than JavaScript functions.
2. Mark-up of JSX would be more familiar to non-programmers.
3. Using JSX, your mark-up becomes more semantic, organized and significant.

How to make your code neat and clean

As I said earlier, the structure / syntax are so easy to visualize / notice which is intended more clean and understandable in JSX format when we compare it with JavaScript syntax.

Below are simple code snippets which will give you clear idea. Let's see code snippets in below example of JavaScript syntax while rendering.

```
render: function () {  
  return React.DOM.div({className:"divider"},  
    "Label Text",  
    React.DOM.hr());  
}
```

JSX syntax:

```
render: function () {  
  return <div className="divider">  
    Label Text<hr />  
  </div>;  
}
```

I'm assuming that it is clear now that JSX is really easy to understand for programmers who are not generally dealing with coding and they can learn it and execute it as they are executing html language.

Acquaintance or understanding

In development region, user interface developer, user experience designer and QA – quality assurance people are not much familiar with any programming language but JSX makes their life easy by providing easy syntax structure which visually similar to HTML structure.

JSX shows a path to indicate and see through your mind's eye, the structure in a solid and concise way.

Semantics / structured syntax

Till now, we have seen how JSX syntax is easy to understand and visualize, behind this there is big reason of having semantic syntax structure.

JSX with pleasure converts your JavaScript code into more standard way, which gives clarity to set your semantic syntax and significance component. With the help of JSX syntax you can declare structure of your custom component with information the way you do in HTML syntax and that will do all magic to transform your syntax to JavaScript functions.

React.DOM namespace helps to use all HTML elements with help of ReacJs, isn't it amazing

feature? It is. Moreover, the good part is, you can write your own named components with help of React.DOM namespace.

Please check out below HTML simple mark-up and how JSX component helps you to have semantic mark-up.

```
<div className="divider">
  <h2>Questions</h2><hr />
</div>
```

As you see in above example, we have wrapped `<h2>Questions</h2><hr />` with `<div>` tag which has `className "divider"` so, in React composite component, you can create similar structure and it is as easy as you do your HTML coding with semantic syntax.

```
<Divider> Questions </Divider>
```

Let's see in detail what composite component is and how we can build it.

Composite component

As we know that, you can create your custom component with JSX mark-up and JSX syntax will transform your component to JavaScript syntax component.

Let's set up JSX:

```
<script type="text/javascript" src="js/react.min.js"></script>
<script type="text/javascript" src="js/react-dom.min.js"></script>
<script src="js/browser.min.js"></script>
<script src="js/divider.js" type="text/babel"></script>
```

Include these files in your HTML.

```
<div>
  <Divider>...</Divider>
  <p>...</p>
</div>
```

Add this HTML in your body section.

Now, we are all set to define custom component using JSX as we have JSX file ready to be worked on.

To create custom component, we have to express below mentioned HTML mark-up as a React custom component. You just have to follow the below example to execute your wrapped syntax/code and in return after rendering it will give you expected mark-up

result.

```
Divider.js
var Divider = React.createClass({
  render: function () {
    return (
      <div className="divider">
        <h2>Questions</h2><hr />
      </div>
    );
  }
});
```

How to append child node in your components, like we have seen in earlier topic that we want to execute mark-up with “divider” in JSX as shown below:

```
<Divider>Questions</Divider>
```

Like in HTML syntax, here child nodes are captured between the open and close tags in an array which you can set it in your component's props (properties).

In this example, we will use `this.props.children === [“Questions”]` where `this.props.children` is React's method.

```
var Divider = React.createClass({
  render: function () {
    return (
      <div className="divider">
        <h2>{this.props.children}</h2><hr />
      </div>
    );
  }
});
```

As we have seen in above example, we can create component with open and close tags the way we are doing in any html coding.

```
<Divider>Questions</Divider>
```

And we will get expected output as shown below.

```
<div className="divider">
  <h2>Questions</h2><hr />
</div>
```

Namespace components

It's another feature request which is available in React JSX. I know you have question, what is Namespace component? Ok, let me explain.

We know that JSX is just extension of JavaScript syntax so, React is not using XML namespaces rather it has used for a standard JavaScript syntax approach which is object property access. This feature is useful for assigning component directly as `<Namespace.Component />` rather than we assign variables to access components which are stored in an object.

Let's start by looking at the following Show/Hide example:

```
var MessagePanel = React.createClass({
  render: function() {
    return <div className='collapse in'> {this.props.children} </div>
  }
});
var MessagePanelHeading = React.createClass({
  render: function() {
    return <h2>{this.props.text}</h2>
  }
});
var MessagePanelContent = React.createClass({
  render: function() {
    return <div className='well'> {this.props.children} </div>
  }
});
```

From below listing, we will see how we can compose a MessagePanel.

```
<MessagePanel>
  <MessagePanelHeading text='Show/Hide' />
  <MessagePanelContent>
    Phasellus sed velit venenatis, suscipit eros a, laoreet dui.
  </MessagePanelContent>
</MessagePanel>
```

A MessagePanel is component which consents rendering a message in your user Interface.

It primarily has two sections.

- MessagePanelHeading: Which displays heading/title of the message? Followed by
- MessagePanelContent: The content of the message.

There is a healthier way to compose MessagePanel by “namespacing” the children. This

can be achieved by making children components as attributes on the parent.

Let's see how.

```
var MessagePanel = React.createClass({
  render: function() {
    return <div className='collapse in'> {this.props.children} </div>
  }
});
MessagePanel.Heading = React.createClass({
  render: function() {
    return <h2>{this.props.text}</h2>
  }
});
MessagePanel.Content = React.createClass({
  render: function() {
    return <div className='well'> {this.props.children} </div>
  }
});
```

So, in above snippets, you can see how we have extended `MessagePanel` by just adding new react component as `Heading` and `Content`.

Now, let's check out if we want to bring namespace notation then how to do composition changes.

```
<MessagePanel>
  <MessagePanel.Heading text='Show/Hide' />
  <MessagePanel.Content>
    Phasellus sed velit venenatis, suscipit eros a, laoreet dui.
  </MessagePanel.Content>
</MessagePanel>
```

Now, we will see practical example of namespace component code after integrating in react with bootstrap:

```
<!doctype html>
<html>
  <head>
    <title>React JS - Namespacing component</title>
    <link rel="stylesheet" href="css/bootstrap.min.css">
    <link rel="stylesheet" href="css/custom.css">
    <script type="text/javascript" src="js/react.min.js"></script>
    <script type="text/javascript"
      src="js/JSXTransformer.js"></script>
  </head>
  <script type="text/jsx">
    /** @jsx React.DOM */
```

```
var MessagePanel = React.createClass({
  render: function() {
    return <div className='collapse in'> {this.props.children} </div>
  }
});

MessagePanel.Heading = React.createClass({
  render: function() {
    return <h2>{this.props.text}</h2>
  }
});

MessagePanel.Content = React.createClass({
  render: function() {
    return <div className='well'> {this.props.children} </div>
  }
});

var MyApp = React.createClass({
  getInitialState: function() {
    return {
      collapse: false
    };
  },
  handleToggle: function(evt){
    var nextState = !this.state.collapse;
    this.setState({collapse: nextState});
  },

  render: function() {
    var showhideToggle = this.state.collapse ?
      (<MessagePanel>
        <MessagePanel.Heading text='Show/Hide' />
        <MessagePanel.Content>
          Phasellus sed velit venenatis, suscipit eros a, laoreet dui.
        </MessagePanel.Content>
      </MessagePanel>)
      : null;
    return (<div>
      <h1>Namespaced Components Demo</h1>
      <p><button onClick={this.handleToggle}
        className="btn btn-primary">Toggle</button></p>
      {showhideToggle}
    </div>)
  }
});

React.render(<MyApp/>, document.getElementById('toggle-example'));
</script>
</head>
<body>
```



```
<div class="container">
  <div class="row">
    <div id="toggle-example" class="col-sm-12">
    </div>
  </div>
</div>
</body>
</html>
```

Let me explain you the above code:

State property contains the state set by `setState` and `getInitialState` of our component. `setState(changes)` method applies the given changes to `this.state` and re-render it `handleToggle` function will handling the state of our component and returning the boolean value `true` or `false`.

We have also used some bootstrap classes to give look and feel of our component.

`.collapse` is for hiding the content

`.collapse.in` is for showing the content

`.well` is for background, border and spacing around the content

`.btn.btn-primary` is for button look and feel.

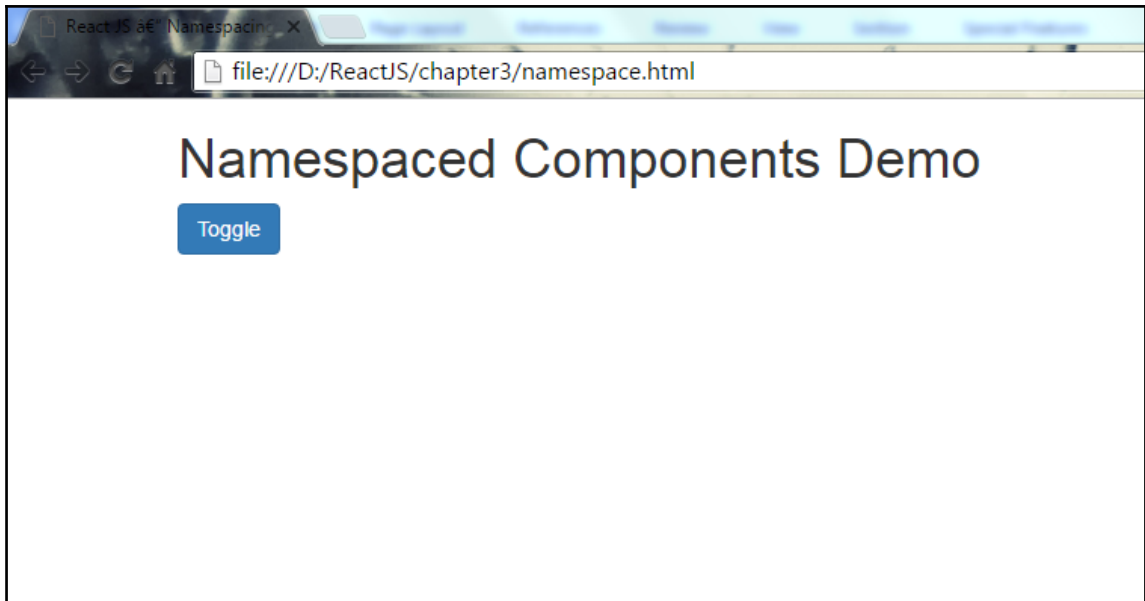
Bootstrap has also provided you some different classes with different color style that help readers to provide a visual indication.

`.btn-default .btn-success .btn-info .btn-warning .btn-danger .btn-link`

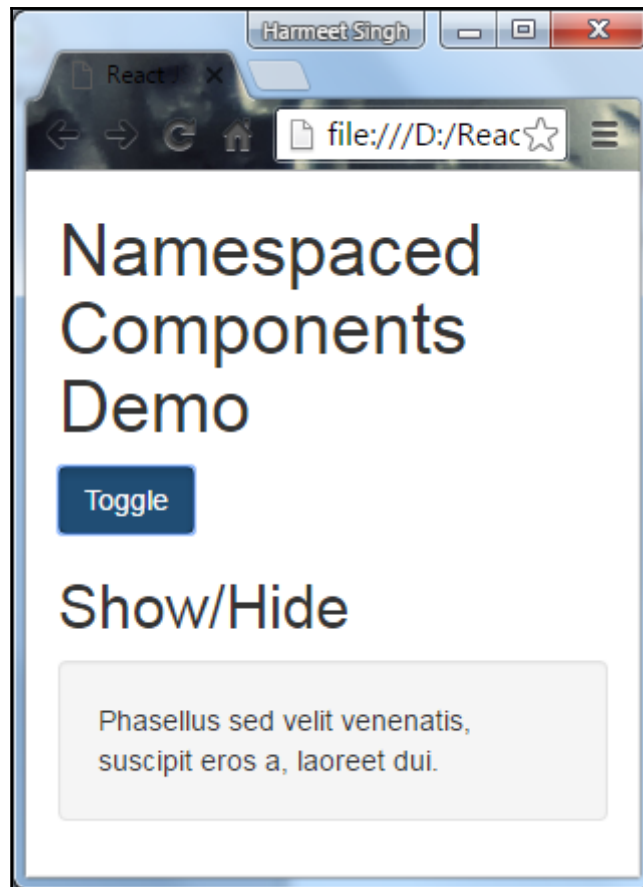
We can use `<a>`, `<button>`, or `<input>` element.

`.col-sm-12` is to make your component responsive for small screens.

Now, let's open your HTML into your browser and see the output.



Now re-size the screen and see:



It looks amazing.

JSXTransformer

JSXTransformer is another tool to compile JSX in the browser. While reading a code, browser will read attribute `type="text/jsx"` in your mentioned `<script>` tag and it will only transform those scripts which has mentioned type attribute and then it will execute your script or written function in that file. The code will be executed in same manner the way React-tools executes on the server.

JSXTransformer is deprecating in current version of react, but you can find the current version on any provided CDNs and Bower. As per my opinion, it would be great to use Babel REPL tool to compile JavaScript. It has already adopted by React and broader JavaScript community.



This example will not work with latest version of React use older version 0.13 as JSXTransformer is deprecated and it's replaced by Babel to transform and run the JSX code in browser. Browser will only understand your `<script>` tags when it has type attribute `type="text/babel"` which we have used in previously examples of chapter 1 and chapter 2.

Attribute expressions

If you can see above example of show/Hide we have used attribute expression for show the message panel and hide it. In react, there is a bit change in writing an attribute value, in JavaScript expression we write attribute in quotes (“”) but we have to provide pair of curly braces ({}).

```
var showhideToggle = this.state.collapse ?
  (<MessagePanel>):null/>;
```

Boolean attributes

As in Boolean attribute, there are two values, either it can be “true” or “false” and if we neglect its value in JSX while declaring attribute, it by default takes value as “true”. If we want to have attribute value “false” then we have to use an attribute expression. This scenario can come regularly when we use HTML form elements, for example “disabled” attribute, “required” attribute, “checked” attribute and “readOnly” attribute.

In bootstrap example `aria-haspopup="true" aria-expanded="true"`

```
// example of writing disabled attribute in JSX

<input type="button" disabled />;
<input type="button" disabled={true} />;
```

JavaScript expressions

As seen in above example, you can embed JavaScript expressions in JSX using syntax that will be accustomed to any handlebars user, for example `style = { displayStyle }` allocates the value of the JavaScript variable `displayStyle` to the element's `style` attribute.

Styles

Same as expression, you can set styles by assigning an ordinary JavaScript object to the `style` attribute. How interesting, if someone tells you, not to write CSS syntax but you can write JavaScript code to achieve the same, no extra efforts. Isn't it superb stuff? Yes, it is.

Events

There is a set of event handlers that you can bind in a way that should look much acquainted to anybody who knows HTML.

Attributes

If you are aware of all the properties well in advance then it would be helpful while creating your component in JSX.

```
var component = <Component foo={x} bar={y} />;
```

If you will change Props then it is bad practice, let's see how:

Generally, as per our practice we set properties on to the object which is anti-pattern in JSX attribute standard.

```
var component = <Component />;
component.props.foo = x; // bad
component.props.bar = y; // also bad
```

As shown in above example, you can see the anti-pattern and it's not the best practice. If you don't know about properties of JSX attributes then propTypes won't be set and it will throw errors which would be difficult for you to trace.

Props is very sensitive part of attribute so, you should not change it, as each props is having predefined method and you should use it as it is meant for like we use other JavaScript methods or HTML tags.

Spread Attributes

Let's check out JSX feature – “spread attributes”:

```
var props = {};
props.foo = x;
props.bar = y;
var component = <Component {...props} />;
```

As you see in above example, your properties which you have declared have become part of your component's props as well.

Reusability of attributes is also possible here and you can also map it with other attributes. But you have to be very careful in ordering your attributes while you declare it, as it will override the previous declared attribute with lastly declared one.

```
var props = { foo: 'default' };
var component = <Component {...props} foo={'override'} />;
console.log(component.props.foo); // 'override'
```

Now hopefully, you have clear idea about JSX, JSX expression and attributes so, let's check out how we can build simple form with help of JSX dynamically.

Example of a Dynamic Form with JSX

Before starting on dynamic form with JSX we must be aware about JSX form libraries.

Generally, HTML form element Inputs are taking its value as display text / value but in React JSX, it takes prop values of respective element and displays it. As we have visually perceived that in our earlier example, we can't change props value directly, so input value won't have that transmuted value as exhibit value.

Let's discuss in bit details, to change value of form input you will use value attribute then you will see no change. That doesn't mean that we cannot change form input value but for that we need to listen to the input events and you will see the value changing.

Below described exceptions are self-explanatory but very important:

1. Textarea content will be considered as value prop in React.
2. As “For” is reserved keyword of JavaScript, HTML for attribute should be bounded like “htmlFor” prop.

Now it's time to see practically, to have form elements in output, we need to follow below script and need to replace with the earlier written code.

Now let's start on building an Add Ticket form for our application.

Create a `React-JSXform.html` file in the root. The following code snippet is just a base HTML page which includes bootstrap and reacts.

Here is markup of our HTML page.

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Dynamic form with JSX</title>
    <link rel="stylesheet" href="css/bootstrap.min.css">
  </head>
  <body>
    <script type="text/javascript" src="js/react.min.js"></script>
    <script type="text/javascript" src="js/react-dom.min.js"></script>
    <script src="js/browser.min.js"></script>

    </body>
  </html>
```

It is always a good practice to load all your scripts at the bottom of the page before your `<body>` tag close which load the component successfully in your DOM because when script is executed in the head section, document element is not available because script itself in head section, so best solution to resolve this problem is to keep scripts at bottom of your page before your `<body>` tag close and it will be executed after loading your all DOM elements, which will not throw any JavaScript error.

Now let's create form elements with bootstrap and JSX.

```
<form>
  <div className="form-group">
    <label htmlFor="email">Email <span style={style}>*</span></label>
    <input type="text" id="email" className="form-control"
      placeholder="Enter email" required/>
  </div>
</form>
```

See the above code where we have used class as “className” and for as “htmlFor” since, JSX is similar to JavaScript and for and class are identifier in JavaScript. We should use “className” and “htmlFor” as property names in ReactDOM component.

All the form elements `<input>`, `<select>` and `<textarea>` will get the global styling with `.form-control` class and apply the `width:100%;` by default so, when we are using label with inputs then we need to wrap with `.form-group` class for optimum spacing.

For our add ticket form, we need these following form fields along with the label.

```
Email: <input>
Issue type: <select>
Assign department: <select>
Comments: <textarea>
```

Button: <button>

To make it responsive Form, we will use **col-** classes.

Let's take a quick look at form component code:

```
var style = {color: "#ffaaaa"};
var AddTicket = React.createClass({
  handleSubmitEvent: function (event) {
    event.preventDefault();
  },
  render: function() {
    return (
      <form onSubmit={this.handleSubmitEvent}>
        <div className="form-group">
          <label htmlFor="email">Email
            <span style={style}>*</span></label>
          <input type="text" id="email" className="form-control"
            placeholder="Enter email" required/>
        </div>
        <div className="form-group">
          <label htmlFor="issueType">Issue Type
            <span style={style}>*</span></label>
          <select className="form-control" id="issueType" required>
            <option value="">-----Select-----</option>
            <option value="Access Related Issue">Access
              Related Issue</option>
            <option value="Email Related Issues">Email
              Related Issues</option>
            <option value="Hardware Request">Hardware Request</option>
            <option value="Health & Safety">Health & Safety</option>
            <option value="Network">Network</option>
            <option value="Intranet">Intranet</option>
            <option value="Other">Other</option>
          </select>
        </div>

        <div className="form-group">
          <label htmlFor="department">Assign Department
            <span style={style}>*</span></label>
          <select className="form-control" id="department" required>
            <option value="">-----Select-----</option>
            <option value="Admin">Admin</option>
            <option value="HR">HR</option>
            <option value="IT">IT</option>
            <option value="Development">Development</option>
          </select>
        </div>
      </form>
    );
  }
});
```



```
    </div>

    <div className="form-group">
      <label htmlFor="comments">Comments <span style={style}>*</span></label>(<span id="maxlength">200</span> characters left)
      <textarea className="form-control" rows="3" id="comments" required></textarea>
    </div>
    <div className="btn-group">
      <button type="submit" className="btn btn-primary">Submit</button>
      <button type="reset" className="btn btn-link">cancel</button>
    </div>
  </form>
);
}
});

ReactDOM.render(
  <AddTicket />,
  document.getElementById('form')
);
```

To apply a style or calling a function onSubmit in attribute value, rather than instead of quotes (") we have to use a pair of curly braces ({} in JavaScript expression.

Now, create one component folder and save this file as a form.js in that and include it in your HTML page.

Here, is our page will look like:

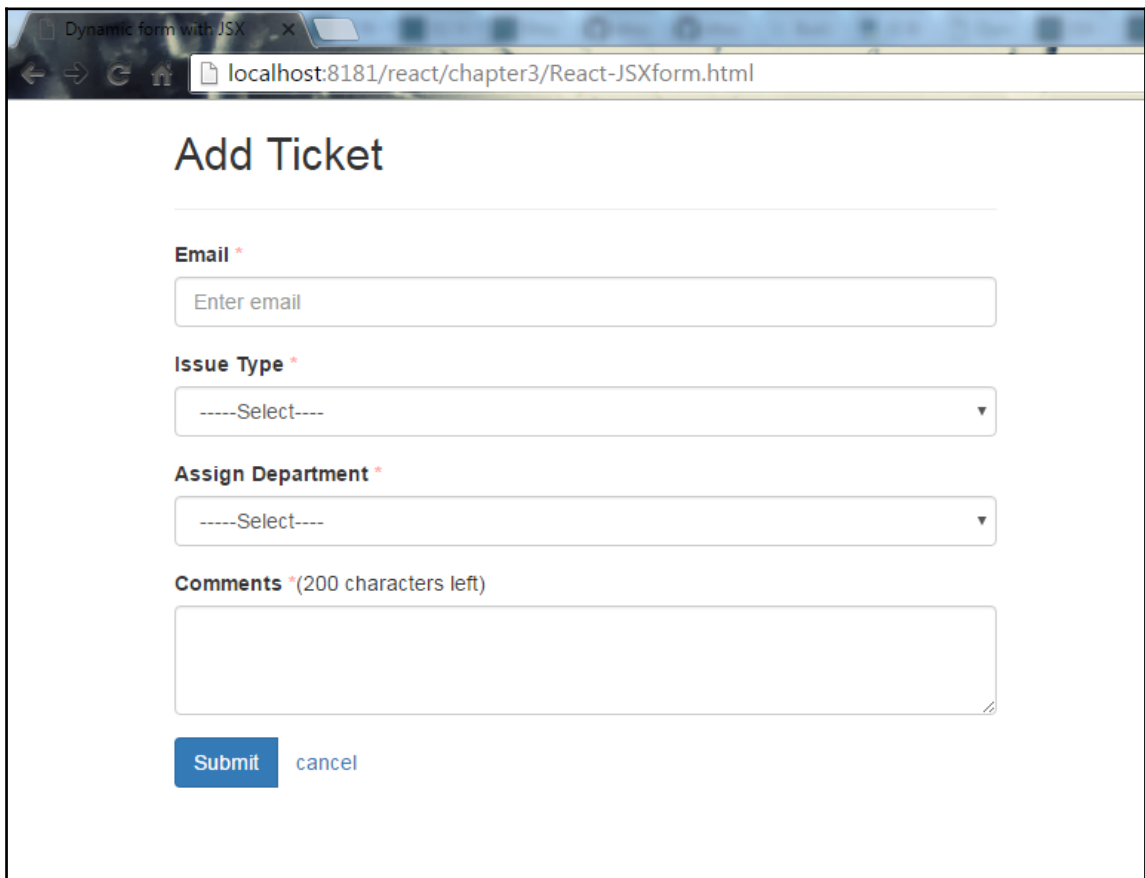
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Dynamic form with JSX</title>
    <link rel="stylesheet" href="css/bootstrap.min.css">
  </head>
  <body>
    <div class="container">
      <div class="row">
        <div class="col-sm-12 col-md-6">
          <h2>Add Ticket</h2>
          <hr/>
          <div id="form">
            </div>
          </div>
        </div>
      </div>
    </div>
  </body>
</html>
```

```

        </div>
      </div>
    </div>
    <script type="text/javascript" src="js/react.min.js"></script>
    <script type="text/javascript" src="js/react-dom.min.js"></script>
    <script src="js/browser.min.js"></script>
    <script src="component/form.js" type="text/babel"></script>
  </body>
</html>

```

Let's take a quick look of our component's output in the browser:



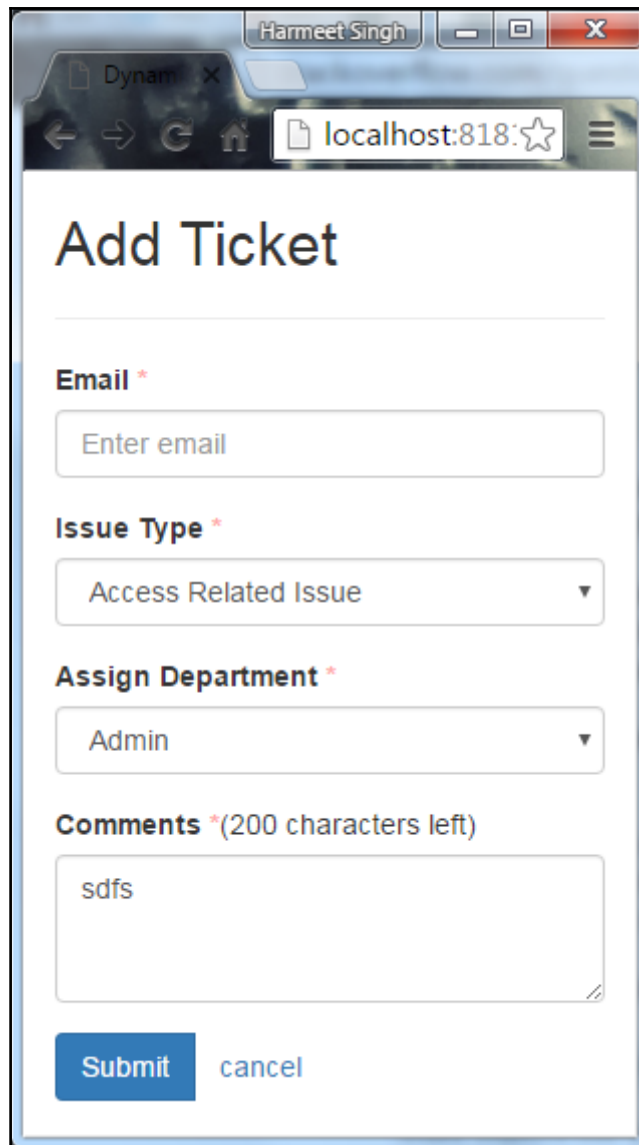
The screenshot shows a web browser window with the title 'Dynamic form with JSX'. The address bar displays 'localhost:8181/react/chapter3/React-JSXform.html'. The main content area features a form titled 'Add Ticket'. The form includes the following fields:

- Email ***: A text input field with the placeholder text 'Enter email'.
- Issue Type ***: A dropdown menu with the placeholder text '-----Select-----'.
- Assign Department ***: A dropdown menu with the placeholder text '-----Select-----'.
- Comments *(200 characters left)**: A text area for comments.

At the bottom of the form, there are two buttons: a blue 'Submit' button and a 'cancel' link.

Oh Cool! It's looking awesome.

Let's check out the form component's responsive behavior while resizing the browser.



The screenshot shows a web browser window with the title 'Harmeet Singh'. The address bar displays 'localhost:8181'. The page content is a form titled 'Add Ticket'. The form includes the following fields:

- Email ***: A text input field with the placeholder text 'Enter email'.
- Issue Type ***: A dropdown menu with the selected option 'Access Related Issue'.
- Assign Department ***: A dropdown menu with the selected option 'Admin'.
- Comments *(200 characters left)**: A text area containing the text 'sdfs'.

At the bottom of the form, there are two buttons: a blue 'Submit' button and a 'cancel' link.



Note: First character should be always capital when you create a component in react. For example, “AddTicket”.

Summary

We have seen that how JSX plays important role for making component custom as well as it's very simple to visualize, understand and write it.

We have taken a look at what is JSX in React and the advantages of using JSX in React. We understood Namespaced components, expressions, and attributes. We saw an example of a dynamic form with JSX.

The key examples shown in chapter will help you to understand or clear your concept about JSX syntax and its implementation.

Last example covers Responsive “Add ticket” form with JSX along with bootstrap which will give you more idea about JSX syntax execution and how to create your custom component. You can use it and instrument it as easy as you play with HTML.

I recommend, still if you are not sure about JSX and its behavior then please go through this chapter again which will help you to through further chapters.

If you are done with all understanding and learning then let's move on to discuss about chapter4, which is all about interacting with DOM with React.

4

DOM Interaction with ReactJS

In the previous chapter, we have learned what is JSX and how we can create a component in JSX. As with many other framework, React also has other prototypes to help us to build our web app. Every framework have different way to interact with DOM elements. React uses a fast, internal synthetic DOM to perform diffs and computes the most efficient DOM mutation for you where your component actually live.

In React components are similar like functions that takes props and state (will explain you further). React components are only render the single root node. If we want to render multiple node then they must be wrapped into the single root node.

Before we start working with form components we should first take a look at props and state.

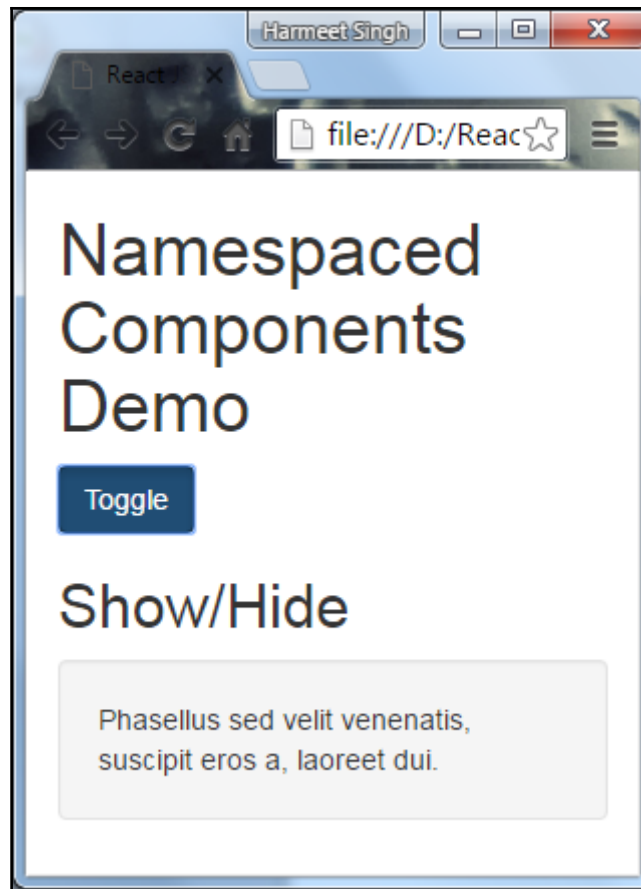
Props and state

React components translate your raw data into Rich HTML, the props and state together build with that raw data to keep your UI consistent.

Okay let's identify what exactly it is:

1. Props and state both are plain JS object.
2. It triggers with render update.
3. React manage the component state by calling `setState(data,callback)`. This method will merge data into `this.state` and re-renders the component to keep our UI up-to-date. For example the state of the dropdown menu (visible or hidden)
4. React component props – short for “properties” that don't change over the time. For example Dropdown menu items Sometimes components only take some data with `this.props` method and render it that makes your component stateless.

5. Using props and state together to help you to make interactive app.



Refer this live example of Chapter 3, *React JS – JSX* you will have better working understanding of state and properties(props)

In this example we are managing the state of toggle (show or Hide) and and text of toggle button as a properties.

Form components

In react form components differ from other native components because they can be modified via user interaction such as `<input>`, `<textarea>`, and `<option>`

Here is the list of supported events: `onChange`, `onInput`, `onSubmit`, `onClick`, `onContextMenu`, `onDoubleClick`, `onDrag`, `onDragEnd`, `onDragEnter`, `onDragExit`, `onDragLeave`, `onDragOver`, `onDragStart`, `onDrop`, `onMouseDown`, `onMouseEnter`, `onMouseLeave`, `onMouseMove`, `onMouseOut`, `onMouseOver`, and `onMouseUp`.

To know full list of supported events can be found in the official documentation at <https://facebook.github.io/react/docs/events.html#supported-events>

Props in form components

As we know ReactJS components have own props and state like forms that support a few props that are affected via user interaction.

`<input>` and `<textarea>`

Components	Supported Props
<code><input></code> and <code><textarea></code>	<code>Value</code> , <code>defaultValue</code>
<code><input></code> type of checkbox or radio	<code>checked</code> , <code>defaultChecked</code>
<code><select></code>	<code>Selected</code> , <code>defaultValue</code>



Note: In HTML `<textarea>` component value set via children but in react it can be set by `value`. `onChange` prop is supported by all native components like other DOM events and can be listen to all bubble change events.

`onChange` prop works across the browser when user interact and change:

The `value` of `<input>` and `<textarea>`

The `checked` state of `<input>` type of radio and checkbox

The `selected` state of `<option>` component

Throughout the chapter, we'll demonstrate that how we can controlled a component with properties(prop) and state we've just looked at. We'll then take a look at how we can apply the same from the component to control the behavior.

Controlled component

The first component we're going to look at is one that control the user input into textarea that prevent to user input when character reached the max length it will also update the remaining character when user input.

```
render: function() {  
  return <textarea className="form-control" value="fdgdfgd" />;  
}
```

In above code we have declared the value of `textarea` so when user input will have no effect to change the value of `textarea`. To controlled this we need to use `onChange` event:

```
var style = {color: "#ffaada"};  
var max_Char='140';  
var Teaxtaarea = React.createClass({  
  getInitialState: function() {  
    return {value: 'Controlled!!!', char_Left: max_Char};  
  },  
  handleChange: function(event) {  
    var input = event.target.value;  
    this.setState({value: input});  
  },  
  render: function() {  
    return (  
      <form>  
        <div className="form-group">  
          <label htmlFor="comments">Comments <span style=  
            {style}>*</span></label><span>{this.state.char_Left}  
          </span> characters left)  
          <textarea className="form-control" value={this.state.value}  
            maxLength={max_Char} onChange={this.handleChange} />  
        </div>  
      </form>  
    );  
  }  
});
```


Controlled Component

Comments *(140 characters left)

Controlled Componentnt

See the preceding image now we are accepting and controlled the value provided by the user and updating the value prop of the `<textarea>` component.



Note: `this.state()` should only contain the minimal amount of data needed to represent your UI's state.

But now we also want to update the remaining characters of textarea in ``

```
this.setState({
  value: input.substr(0, max_Char), char_Left:
  max_Char - input.length
});
```

In above code this would be control the remaining value of textarea and update the remaining character when user input.

Uncontrolled component

As we've seen in ReactJS using value property we can control the user input so `<textarea>` without value property is an **uncontrolled component**.

```
render: function() {
  return <textarea className="form-control"/>
}
```

This will render the textarea with empty value and User can allow to input the value that would be reflected immediately by the rendered element because uncontrolled component have own internal state or if you want to initialize the default value we need to use `defaultValue` prop.

```
render: function() {  
  return <textarea className="form-control" defaultValue="Lorem  
  ipsum"/>  
}
```

It look's like controlled component which we have seen before.

Getting the form values on submit

As we've seen, the state and prop will give you the control to alter the value of component and handle the state for that component.

Ok now let's add some advanced features in our AddTicket form which can validate the user input and display the tickets it on the UI.

Ref attribute

React provide ref Non-DOM attributes to access the react component. Ref attribute can be a callback function and it will execute immediate after the component mounted.

So we will attach ref attribute in our form element to fetch the values

```
var AddTicket = React.createClass({  
  handleSubmitEvent: function (event) {  
    event.preventDefault();  
    console.log("Email--"+this.refs.email.value.trim());  
    console.log("Issue Type--"+this.refs.issueType.value.trim());  
    console.log("Department--"+this.refs.department.value.trim());  
    console.log("Comments--"+this.refs.comment.value.trim());  
  },  
  render: function() {  
    return (  
      <form onSubmit={this.handleSubmitEvent}>  
        <div className="form-group">  
          <label htmlFor="email">Email <span  
style={style}>*</span></label>  
          <input type="text" id="email" className="form-control"  
placeholder="Enter email" required ref="email"/>  
        </div>  
        <div className="form-group">  
          <label htmlFor="issueType">Issue Type <span style=  
{style}>*</span></label>  
          <select className="form-control" id="issueType"  
required ref="issueType">
```

```
        <option value="">-----Select-----</option>
        <option value="Access Related Issue">Access Related
Issue</option>
        <option value="Email Related Issues">Email Related
Issues</option>
        <option value="Hardware Request">Hardware Request</option>
        <option value="Health & Safety">Health & Safety</option>
        <option value="Network">Network</option>
        <option value="Intranet">Intranet</option>
        <option value="Other">Other</option>
      </select>
    </div>

    <div className="form-group">
      <label htmlFor="department">Assign Department
      <span style={style}>*</span></label>
      <select className="form-control" id="department"
required ref="department">
        <option value="">-----Select-----</option>
        <option value="Admin">Admin</option>
        <option value="HR">HR</option>
        <option value="IT">IT</option>
        <option value="Development">Development</option>
      </select>
    </div>

    <div className="form-group">
      <label htmlFor="comments">Comments <span style={style}>*</span></label>(<span id="maxlength">200</span> characters left)
      <textarea className="form-control" rows="3" id="comments"
required ref="comment"></textarea>
    </div>
    <div className="btn-group">
      <button type="submit" className="btn btn-primary">Submit</button>
      <button type="reset" className="btn btn-link">cancel</button>
    </div>
  </form>
  );
}
});
```

Se above code I have added the ref attribute on our form elements and onSubmit calling the function name handleSubmitEvent. Inside this function we are fetching the values with this.refs.

Now, open your browser and Let's see the output of our code

The screenshot shows a web browser at `localhost:8181/react/chapter4/advance-form.html`. The page has a title "Add Ticket". Below the title is a form with four fields: "Email *" with the value "harmeetsingh090@gmail.com", "Issue Type *" with the value "Email Related Issues", "Assign Department *" with the value "IT", and "Comments *(200 characters left)" with the value "Email is not working". At the bottom of the form are "Submit" and "cancel" buttons. Below the form is a Chrome DevTools console window showing the following log entries:

Log Entry
Email--harmeetsingh090@gmail.com
Issue Type--Email Related Issues
Department--IT
Comments--Email is not working

We are successfully getting the values for our component. It's very clear how data is flowing in our component. In console we can see the values of form when user submit the button.

Now, let display this ticket info in our UI.

First we need to get the value of the form and manage the state of the form

```
var AddTicket = React.createClass({  
  handleSubmitEvent: function (event) {
```

```
        event.preventDefault();
var values = {
    date: new Date(),
    email: this.refs.email.value.trim(),
    issueType: this.refs.issueType.value.trim(),
    department: this.refs.department.value.trim(),
    comment: this.refs.comment.value.trim()
};
this.props.addTicketList(values);
},
});
```

Now we will create component AddTicketsForm that will be the responsible of manage and hold the state of addTicketList(values).

```
var AddTicketsForm = React.createClass({
    getInitialState: function () {
        return {
            list: {}
        };
    },
    updateList: function (newList) {
        this.setState({
            list: newList
        });
    },
    addTicketList: function (item) {
        var list = this.state.list;

        list[item] = item;
        //pass the item.id in array if we are using react key attribute.
        this.updateList(list);
    },
    render: function () {
        var items = this.state.list;
        return (
            <div className="container">
                <div className="row">
                    <div className="col-sm-6">
                        <List items={items} />
                        <AddTicket addTicketList={this.addTicketList} />
                    </div>
                </div>
            </div>
        );
    }
});
```

Let's take a look on the preceeding code

- `getInitialState`: initialized the default state for `<List />` component
- `addTicketList`: Hold the the value and passing into the **updateList** with state
- `updateList`: For updating the list of the tickets to make our UI in sync

Now we need to create `<List items={items} />` component that iterate the list when we submit the form

```
var List = React.createClass({

  getListOfIds: function (items) {
    return Object.keys(items);
  },

  createListElements: function (items) {
    var item;

    return (
      this
      .getListOfIds(items)
      .map(function createListItemElement(itemId) {
        item = items[itemId];
        return (<ListPanel item={item} />); //key={item.id}
      }).bind(this)
      .reverse()
    );
  },

  render: function () {
    var items = this.props.items;
    var listItemElements = this.createListElements(items);

    return (
      <div className="bg-info">
        {listItemElements}
      </div>
    );
  }
});
```

- `getListOfIds`: It will iterate through all the keys in item and return the list which we have map with the `<ListPanel item={item}/>` component
- `.bind(this)`: this keyword will be passed as a second argument which gives the appropriate value when function called

In render method we are just rendering the list of elements.

In addition we can also add condition based on the length inside the render method.

```
<p className={listItemElements.length > 0 ? "":"bg-info"}>
  {listItemElements.length > 0 ? listItemElements :
    "You have not raised any ticket yet. Fill this form
    to submit the ticket"}
</p>
```

It will validate the length and based on the return value TRUE : FALSE, will display the message or apply the bootstrap class .bg-info

Now we need to create a `<ListPanel />` component of displaying the list of tickets in UI.

```
var ListPanel = React.createClass({
  render: function () {
    var item = this.props.item;
    return (
      <div className="panel panel-default">
        <div className="panel-body">
          {item.issueType}<br/>
          {item.email}<br/>
          {item.comment}
        </div>
        <div className="panel-footer">
          {item.date.toString()}
        </div>
      </div>
    );
  }
});
```

Now, let's combine our code out and see the result in browser

```
var style = {color: "#ffa000"};
var AddTicketsForm = React.createClass({

  getInitialState: function () {
    return {
      list: {}
    };
  },
  updateList: function (newList) {
    this.setState({
      list: newList
    });
  },

  addTicketList: function (item) {
```

```
        var list = this.state.list;
        list[item] = item;
        this.updateList(list);
    },
    render: function () {
        var items = this.state.list;
        return (
            <div className="container">
                <div className="row">
                    <div className="col-sm-12">
                        <List items={items} />
                        <AddTicket addTicketList={this.addTicketList} />
                    </div>
                </div>
            </div>
        );
    }
});

var ListPanel = React.createClass({
    render: function () {
        var item = this.props.item;
        return (
            <div className="panel panel-default">
                <div className="panel-body">
                    {item.issueType}<br/>
                    {item.email}<br/>
                    {item.comment}
                </div>
                <div className="panel-footer">
                    {item.date.toString()}
                </div>
            </div>
        );
    }
});

var List = React.createClass({

    getListOfIds: function (items) {
        return Object.keys(items);
    },
    createListElements: function (items) {
        var item;

        return (
            this
                .getListOfIds(items)
                .map(function createListItemElement(itemId) {
```



```
        item = items[itemId];
        return (<ListPanel item={item} />); //key={item.id}
    }.bind(this))
    .reverse()
  );
},
render: function () {
  var items = this.props.items;
  var listItemElements = this.createListElements(items);

  return (
    <p className={listItemElements.length > 0 ? "":"bg-info"}>
      {listItemElements.length > 0 ? listItemElements : "You have
      not raised any ticket yet. Fill this form to submit the ticket"}
    </p>
  );
}
});

var AddTicket = React.createClass({
  handleSubmitEvent: function (event) {
    event.preventDefault();
    var values = {
      date: new Date(),
      email: this.refs.email.value.trim(),
      issueType: this.refs.issueType.value.trim(),
      department: this.refs.department.value.trim(),
      comment: this.refs.comment.value.trim()
    };
    this.props.addTicketList(values);
  },
  render: function() {
    return (
```

// Form template

```
ReactDOM.render(
  <AddTicketsForm />,
  document.getElementById('form')
);
```

Here is markup of our HTML page.

```
<link rel="stylesheet" href="css/bootstrap.min.css">
<style type="text/css">
div.bg-info {
  padding: 15px;
}
```

```
    </style>
  </head>
  <body>
    <div class="container">
      <div class="row">
        <div class="col-sm-6">
          <h2>Add Ticket</h2>
          <hr/>
        </div>
      </div>
    </div>
    <div id="form">
  </div>
  <script type="text/javascript" src="js/react.js"></script>
  <script type="text/javascript" src="js/react-dom.js"></script>
  <script src="js/browser.min.js"></script>
  <script src="component/advance-form.js" type="text/babel"></script>
</body>
```

Open your browser and Let's see the output of our form before submit

Dynamic form with JSX

localhost:8181/react/chapter4/advance-form.html

Add Ticket

You have not raised any ticket yet. Fill this form to submit the ticket

Email *

Enter email

Issue Type *

-----Select-----

Assign Department *

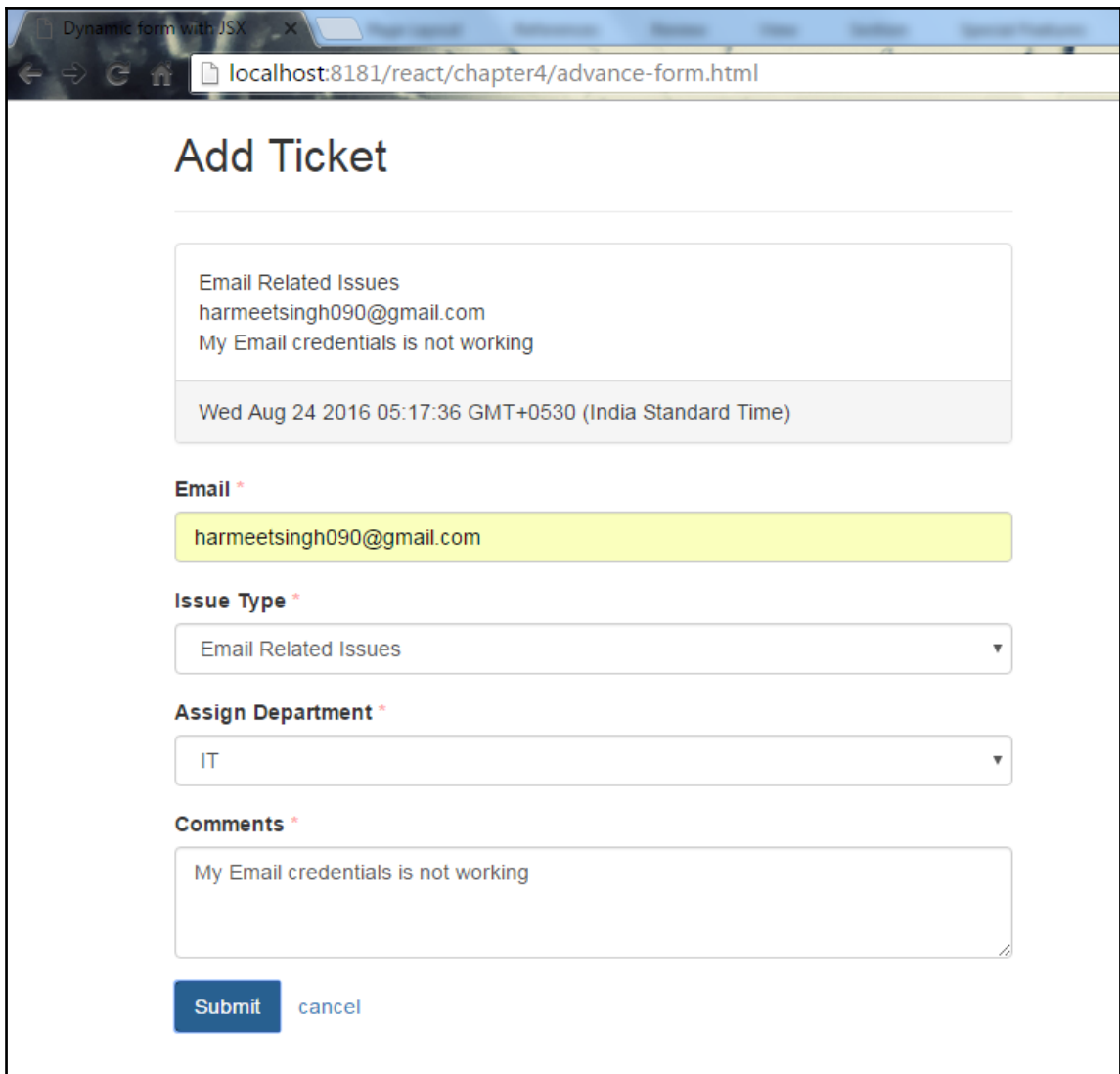
-----Select-----

Comments *

Submit

cancel

After submit the form



The screenshot shows a web browser window with the address bar displaying 'localhost:8181/react/chapter4/advance-form.html'. The page title is 'Dynamic form with JSX'. The main content is a form titled 'Add Ticket'. The form contains a text area with the text 'Email Related Issues', 'harmeetsingh090@gmail.com', and 'My Email credentials is not working'. Below this is a timestamp 'Wed Aug 24 2016 05:17:36 GMT+0530 (India Standard Time)'. The form has several fields: 'Email *' with the value 'harmeetsingh090@gmail.com', 'Issue Type *' with a dropdown menu showing 'Email Related Issues', 'Assign Department *' with a dropdown menu showing 'IT', and 'Comments *' with the text 'My Email credentials is not working'. At the bottom are 'Submit' and 'cancel' buttons.

Dynamic form with JSX x

localhost:8181/react/chapter4/advance-form.html

Add Ticket

Email Related Issues
harmeetsingh090@gmail.com
My Email credentials is not working

Wed Aug 24 2016 05:17:36 GMT+0530 (India Standard Time)

Email *

harmeetsingh090@gmail.com

Issue Type *

Email Related Issues ▼

Assign Department *

IT ▼

Comments *

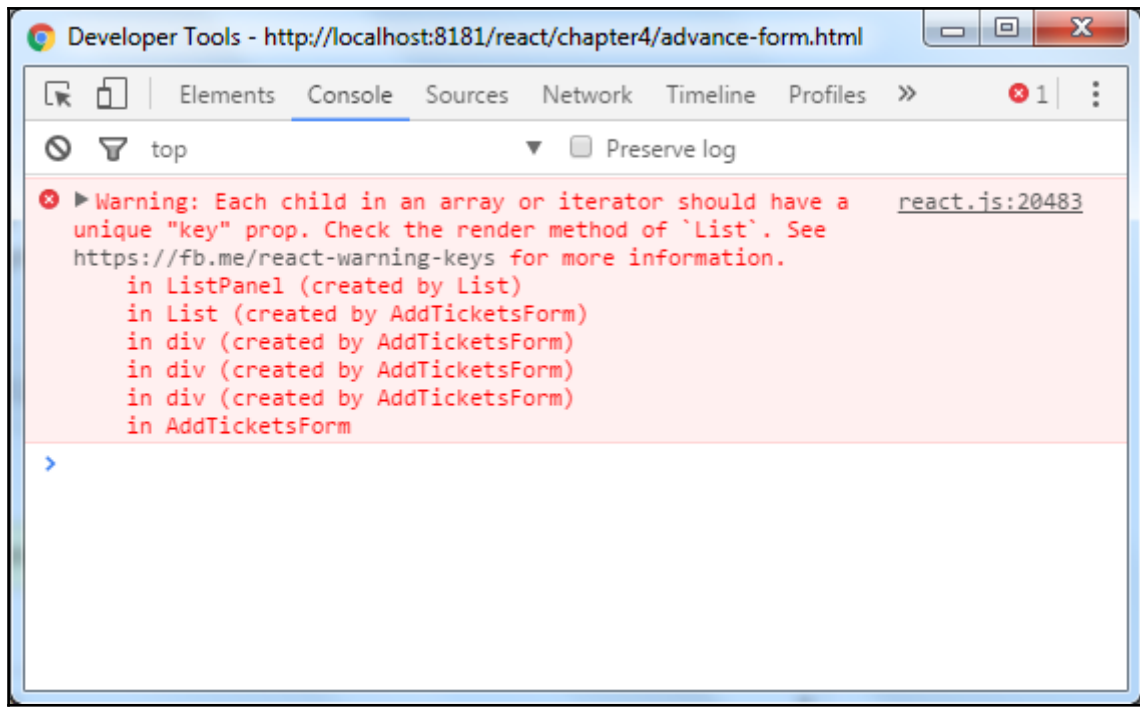
My Email credentials is not working

Submit cancel

This looks good. Our first fully functional react component is ready.



Note: Never access refs inside any component and it may not be attached to a stateless function



We are getting this warning message because of key (an optional) attribute of react that accept unique id everytime when we submit the form it will iterate the list component to update the UI.

React is providing Add-Ons module to solve this type of warning and generate the unique id but that are only available in npm. In further chapters will show you how we can work with react npm modules.

Here is the list of some popular add-ons:

- **TransitionGroup and CSSTransitionGroup:** For dealing with animations and transitions
- **LinkedListMixin:** To make it easy interaction user's form input data and the component's state.
- **cloneWithProps:** change the props of component and make a shallow copies
- **createFragment:** Use to create a set of externally keyed children
- **Update:** Helper function that makes you easy dealing with data in javascript.
- **PureRenderMixin:** A performance booster
- **shallowCompare:** A helper function to do shallow comparison for props and

state.

Bootstrap Helper Classes

Bootstrap will be providing some helper classes to give you the better User experience. In `AddTicketsForm` form component we have used bootstrap helper classes `*-info`, it helps you to convey the meaning of your message with color for screen readers.

`*-muted`, `*-primary` `-success` `*-info` `*-warning` `*-danger`

For changing the color of the text we can use `.text*`

```
<p class="text-info">...</p>
```

For changing the background color we can use `.bg*`

```
<p class="bg-info">...</p>
```

Caret

For displaying the caret that will indicate the direction of dropdown.

```
<span class="caret"></span>
```

Clearfix

Using clearfix on the parent element we can clear the float of child elements.

```
<div class="clearfix">...  
  <div class="pull-left"></div>  
  <div class="pull-right"></div>  
</div>
```

Summary

We have seen that how props and state plays important role for making component interactive as well as in DOM interaction. Refs is the great way to interact you DOM elements that would be inconvenient to do via streaming Reactive props and state With the help of Refs we can call any public method and send a message to our particular child instance.

In this chapter we have seen the props and their state, along with how it works. We take a look at the controlled and uncontrolled components. How react interacts with DOM elements is also a section in this chapter we have looked upon. Later on, we saw the Non-DOM attribute keys and ref. Lastly we see an example of a advance form with JSX.

The key examples shown in chapter will help you to understand or clear your concept about props, state, and DOM interaction.

Last example covers advance “Add ticket” form with multiple JSX components along with bootstrap which will give you more idea about creating react component and how we can interact with using refs. You can use it and instrument it as easy as you play with HTML.

I recommend, still if you are not sure how state, props works, and how react interacts with DOM then please go through this chapter again which will help you in the further chapters.

If you are done then let's move on to discuss about `Chapter 5`, which is all about Redux architecture in react.