ReactJS Fundamentals

What is ReactJS?

- A JavaScript library for building user interfaces (built by Facebook)
- Declarative: The state of the application determines what the UI looks like.
- Composable: create building blocks that are used to make bigger and more complicated structures

Why ReactJS?

There are so many javascript frameworks. Why do we need another?

- Speed: React is fast!
- Declarative: easier to read code and prevent bugs.
- Composable: easier to build more modular and reusable code.
- Learning how to write a React web app gives you the experience to write an app for another platform
  - iOS
  - Android
  - Windows 10

Community

- has a large community backing
- tons of support
- lots of smart people working on it
- if you have a problem, it's probably on StackOverflow

Hello World

```
<!DOCTYPE html>
<html>
<head>
    <title>Hello World</title>
    <script src="https://unpkg.com/react@15.3.2/dist/react.js"></script>
    <script src="https://unpkg.com/react-dom@15.3.2/dist/react-dom.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.24/browser.min.js"></script>
</head>
<body>
    <div id="container"></div>

    <script type="text/babel">
        var HelloWorld = React.createClass({
            render: function() {
                return <div> Hello World </div>
            }
        });

        ReactDOM.render(<HelloWorld />, document.getElementById('container'));
    </script>
</body>
</html>
```

What's Going On Here?

- createClass: used to create a React Component (More on this to come...)
- render function: minimal property that needs to be defined on an object passed to the createClass.
- HTML in JavaScript? What?!
  ** This is not actually HTML, but something similar called JSX

React with ES6 (setup)

We're not too focused on build processes today, let's build our app with the `create-react-app` node module.

Let's build a new React app!

```
npm install -g create-react-app
create-react-app my-app
cd my-app/
npm start
```

Hello World Example using ES6.

```
//App.js
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class HelloWorld extends Component {
  render() {
    return <h1>Hello World!</h1>
  }
}

class App extends Component {
  render() {
    return <HelloWorld />
  }
}

export default App;
```

How does it all work?

- Virtual DOM
- Diffing

Intro to Virtual DOM

- React uses what you return in the 'render' method to create what's called a virtual DOM.
- A virtual DOM is an in memory representation of a DOM.
- Any time the value of one of your state variables change, React will update it's Virtual DOM.
- When defining the render function, we are not trying to create the actual DOM, but the virtual DOM.
    - React will take care of rendering the actual DOM.
    - JSX is the language we use to generate the virtual DOM.

Diffing

- When React determines a change has been made it does what's called a diff on the virtual DOM.
- A diff is a comparison between to things where and how they are different.
- Once it determines where the changes have been made it updates the actual DOM of your page.
- Doing this in the virtual DOM is cheap. This is what makes React so fast!

JSX

React uses a special syntax called JSX.

- JSX is a language written by Facebook that simplifies the amount of code we have to write.
- JSX allows us to use XML inside of our JavaScript. Like XML, JSX has tags. Tags can have attributes and children.

JSX

- JSX is not something that browsers understand so we have to transpile it to JavaScript before serving it to the browser
- We previously transpiled ES6 code into ES5 code using Babel
- We can also use Babel to transpile our JSX code into JavaScript (more on this to come...)

JSX

JSX is optional. You can use React without it but JSX will make our React code much easier to read and write.

```
// React without JSX
render() {
  return React.createElement(
    "div",
    {className: "foo"},
    "Hello ",
    this.props.name
  );
}

// React with JSX
render() {
  return <div className="foo">Hello {this.props.name}</div>;
}
```

JSX

- React can render HTML tags and React Components. Below is an example of rendering a div tag.
- HTML tags are lowercase. React distinguishes between HTML tags and React components by their casing.

```
var exampleDivElement = <div>
        <h1> Welcome </h1>
        <MyComponent className="foo">Hello World</MyComponent>
    </div>
```

JSX

Why "className" instead of class?

- Since JSX is written in JavaScript, identifiers such as `class` and `for` are discouraged as XML attribute names.
- Instead, React DOM components expect DOM property names like className and htmlFor, respectively

JSX

You can use JavaScript expressions in JSX using curly braces.

```
<h1>Hello {this.user.name}</h1>
```

and also

```
<h1>Total Amount {2 + 2}</h1>
```

Exercise

- Get yourselves setup with 'create-react-app'
- 'create-react-app' to make a sample application
- Get the Hello World application working.

Component Based Architecture

- Declare a state and tie it to a specific UI component
- Think small to build big! Single purpose components and isolated scope make it easier to debug and maintain
- Small components are re-usable through your app (DRY - don't repeat yourself)
- You might have a component for different features on your site such as:
  - Navbar
  - Search form
  - Create comment form
  - Slider

What does a component look like?

Minimum requirements

- extends the Component class
- defines the 'render' function

```
class HelloWorld extends Component {
  render() {
    return <h1>Hello World!</h1>
  }
};
```

Gotchas

- `render` should return a *single* HTML element.
- Every self-closing tag in JSX needs to have an ending `/`: `<br>` needs to be `<br/>`, etc.

What about the data

- We have two ways of getting data into components
  - Props
  - State

Props (aka: Properties)

- props are the data that is passed into your component
- props are immutable - they can't be changed!
- think of props as being the default data for your component
- can't pass objects/arrays into props!
- Component Docs
- Does not take objects

Example

```
class HelloWorld extends Component {
  render() {
    return <h1>Hello {this.props.name}!</h1>
  }
};

ReactDOM.render(<HelloWorld name="Tom" />)
```

State

- But how do you make UI changes if your data is immutable? With state!
- An object defined on the Component that can be used in the render function to define data that will change.

Example

```
class HelloFriend extends Component {
    constructor(props) {
      // props.name is default data; in our first example we passed in 'David'
      // need to call super(props) when using 'this' in constructor
      super(props);
      this.state = {
        name: props.name
      }

      setTimeout(this.updateName.bind(this), 2000)
    }

    updateName() {
        this.setState({name: 'Jeff'});
    }

    render() {
      return <h1>Hello {this.state.name}!</h1>
    }
};
```

Proptypes

- Defined as the 'propTypes' property on the Component
- Essentially a dictionary where you define what props your component needs
- and what type(s) they should be
  - a type is a `Number`, `Object`, `String`, etc.
- `id` prop should of type `Number`
- `id` is also required!
- `message` is not required, and it is of type `String`

```
MyCoolComponent.propTypes = {
  id: React.PropTypes.number.isRequired,
  message: React.PropTypes.string
}
```

Example

```
import React, { Component, PropTypes } from 'react';

class MyCoolComponent extends Component {
  constructor(props) {

  }

  render() {
    return <div id={this.props.id}>{this.props.message}</h1>
  }
};

MyCoolComponent.propTypes = {
  id: PropTypes.number.isRequired,
  message: PropTypes.string
}
```

Exercise: Contact Cards

Props and Proptypes

- Define a 'ContactCard' component that takes in a contact name, mobile number, work phone number and email as properties and displays the results in a visually appealing way.
- The name, mobile number and email are required, but the work phone number is optional.
- Create a page that display at least 3 different contact cards with different information.

Exercise: Decrement

State

- Define a 'Decrement' component that will display a number and an "Increment" button next to it.
- The start number should come from the props.
- The number should have a `number` prop type and be required on the component.
- Clicking on the "Decrement" button should subtract 1 to the number.
- When the number reaches zero, clicking on "Decrement" should show an alert to the user "Cannot be less than zero" and not decrement the number any more.