**REACT**

ReactJS is an open-source, component based front end library responsible only for the **view layer** of the application. It is maintained by Facebook.

NOTE: You can install React using-

* npm **install** --save react react-dom
* yarn add react react-dom

**React components**

* React components are ES6 classes that extend the class React.Component.
* A React application is made up of multiple components, each responsible for *outputting a small, reusable piece of HTML.* Components can be nested within other components to allow complex applications to be built out of simple building blocks. A component may also maintain internal state - for example, a TabList component may store a variable corresponding to the currently open tab.
* React allows us to write components using a domain-specific language called JSX. JSX allows us to write our components using HTML, whilst mixing in JavaScript events. React will internally convert this into a virtual DOM, and will ultimately output our HTML for us. The virtual DOM is an in-memory representation of an actual DOM.
* The most basic type of react component is one without state
* React components that are pure functions of their props and do not require any internal state management can be written as simple JavaScript functions. These are said to be Stateless Functional Components because they are a function only of props, without having any state to keep.

Here is a simple example to illustrate the concept of a Stateless Functional Component:

*// In HTML*

<div id="element"></div>

*// In React*

**const** MyComponent = props => {

**return** <h1>Hello, {props.name}!</h1>;

};

ReactDOM.render(<MyComponent name="Arun" />, element);

*// Will render <h1>Hello, Arun!</h1>*

Note that all that this component does is render an h1 element containing the name prop. This component doesn't keep track of any state.

There are two ways to declare React components:

(1) As ES6 classes.

(2) Function components

An example of using an ES6 class:

**class** HelloWorld **extends** React.Component {

render() { **return** <p>Hello, world!</p>; }

}

The same component written in a “functional component” style:

**function** HelloWorld() {

**return** <p>Hello, world!</p>;

}

* The syntax of the return value doesn’t look like traditional JavaScript. We’re using

**JSX** (JavaScript eXtension syntax), a syntax extension for JavaScript written by

Facebook.

* Using JSX enables us to write the markup for our component views in a

familiar, HTML-like syntax. In the end, this JSX code compiles to vanilla JavaScript.

**JSX**

* React components ultimately render HTML which is displayed in the browser. As such, the render() method of a component needs to describe how the view should be represented as HTML.
* React builds our apps with a fake representation of the Document Object Model (DOM). React calls this the virtual DOM.
* The Document Object Model (DOM) refers to the browser’s HTML tree that makes up a web page.
* JSX was created to make this JavaScript representation of HTML more HTML-like. To understand the difference between HTML and JSX, consider this JavaScript syntax:
* Remember, the JSX components return is not actually the HTML that gets rendered, but is the representation that we want React to render in the DOM.

React.createElement('div', {className: 'ui items'},

'Hello, friend! I am a basic React component.'

)

Which can be represented in JSX as:

<div className='ui items'>

Hello, friend! I am a basic React component.

</div>

**Babel**

* Most browsers in use today do not fully support ES6.
* Babel is a JavaScript **transpiler**. **Babel turns ES6 code into ES5 code**. We call this process **transpiling**.
* Babel compiles our JSX into vanilla ES5 JS that our browser can then interpret and execute.

<**head**>

*<!-- ... -->*

<**script** src="vendor/babel-standalone.js"></**script**>

*<!-- ... -->*

</

<script src="./js/seed.js"></script>

**<script**

**type="text/babel"**

**data-plugins="transform-class-properties"**

**src="./js/app.js"**

**></script>**

* The attribute type="**text/babel**" indicates to Babel that we would like it to handle the loading of this script. The attribute **data-plugins** specifies a special Babel plugin we use.

**ReactDOM.render()**

* We need to instruct React to render this ProductList inside a specific DOM node.
* ReactDOM is from the react-dom library that we also include in index.html. We pass in two arguments to the ReactDOM.render() method. The first argument is ***what*** we’d like to render. The second argument is ***where*** to render it:

ReactDOM.render([what], [where]);

**ReactDOM.render(**

**<ProductList />,**

**document.getElementById('content')**

**);**

* NOTE:

To recap, we wrote a React component using an ES6 class as well as JSX. We specified

that we wanted Babel to transpile this code to ES5. We then used ReactDOM.render()

to write this component to the DOM.

* In React, native HTML elements always start with a lowercase letter whereas React component names always start with an uppercase letter.