**Introduction to React**

**Learning Objectives**

* Understand the benefits of using a framework
* Know what React is
* Explain some of the key features of React
* Be able to create an application using Create React App

**Introduction**

**Frameworks**

Both back-end and front-end frameworks provide our applications with structure and often handle common tasks for us. Frameworks are designed to make building an application quicker and easier, and help us to achieve a result which is easier to reason about and therefore easier to maintain. However, in order to use any framework, we have to learn how to learn how it wants us to do things.

Each framework has its own way of doing things; some are more opinionated than others but once we’ve learned how to talk in a framework’s language, we can leverage its functionality, focus on the core of our application and the logic we need to write.

What is the difference between a framework and a library?

* A framework is a type of library; it is an organisational library, and will often contain methods that we can use, but don’t always call ourselves.
* A library is a set of methods and objects that we can use to help us complete programming tasks.

**JavaScript Front-end Frameworks**

Writing well-structured vanilla JavaScript applications, with a clear separation of concerns, it difficult to achieve. There are also common tasks that we want to carry out, such as manipulating the DOM, which become repetitive and time consuming. For these reasons, developers over the years have created libraries and frameworks that help with web development. We are going to be using the front-end framework React to help us with our front-end development.

**React**

**Popularity**

React says that it is ‘A JavaScript library for building user interfaces’. It is a front-end framework, created and used by Facebook. Making applications that are fast and easier to develop has made React a very popular choice in the web community and it is widely used as shown by the following:

* [A list of companies that use React](https://stackshare.io/)
* [State of JavaScript 2019 - Front-end](https://2019.stateofjs.com/front-end-frameworks/)

**Key Features**

If we go to the [React the website](https://facebook.github.io/react/), we can see there are three main characteristics of React:

1. One Way Data Flow
2. Virtual DOM
3. Component based UI - ‘HTML’ in JS with JSX

This week we are going to be using React to help us build well-structured view logic, using its DOM manipulation methods to make reading and writing to the DOM more efficient than we have previously been able to achieve with vanilla JavaScript.

**One way flow**

React takes a one-way flow approach. All of the data your app uses - its state - is kept in one place and then passed down through a hierarchy/tree of components for them to use as needed.

When a change happens, we alter the total state of our app and then just do the whole initial display again. We don’t need to worry about how each little change alters our UI, we just render the whole thing again, passing down the new data.

You might be thinking this incredibly inefficient. And it would be if it wasn’t for the middle item on that list, the virtual DOM.

**Virtual DOM**

Updating the DOM using JS is an expensive operation, it tasks a huge amount of time compared to us multiplying numbers, or searching a reasonably sized array. If after every little change our React application re-rendered the whole DOM, our app would be very inefficient and very slow.

React applications are known for being fast. So, how does it achieve this? React keeps a virtual version of the DOM in memory. When we make a change it updates this and compares it to the real DOM (a diff). It only updates the real DOM where necessary.

It is this step which allow us to write fast efficient applications by only having to describe a simple render.

**Component based UI**

React encourages us to break down our user interface into smaller parts called components. (By doing this we can make sure each part is only looking after one thing… which software principle does this relate to? Single Responsibility principle!) These can be reusable and are really helpful in separating the functionality of your app into independent parts.

These components may (but don’t have to) use a React-specific syntax called JSX. JSX allows us to use HTML-like syntax when describing new elements, whether HTML elements or React components, which saves us time and can make our React code more readable. Some frameworks use a templating language for a similar purpose, writing HTML which can pull in information from a separate JavaScript file, but React allows us to do all of this in one place. (Some people prefer to have those things separate, it depends on the nature of your app as to what’s most appropriate.)

**Create React App**

When creating a React application we can configure our application manually, setting up our own server, installing React and any other packages we might need to get it up and running. However, Facebook have produced a package called ‘Create React App’ that allow us to create a boilerplate application with all the build configuration already complete. It comes with a number of useful features already configured:

* A built-in web server (so we don’t need to set up Express)
* A build script that includes a pre-configured version of Webpack
* Babel (Transpiles our code at the point of bundling, so we can use all the latest language features without worrying if they are supported by our users’ browsers.)
* The ability to import CSS and images
* Hot reloading (Injects changes to your source code into the current state of the page, so we don’t need to refresh the browser each time you make a change.)
* Easily deployable to many hosting providers, for example Github Pages
* [Jest](https://facebook.github.io/jest/) - a testing framework that lets us test our app in various ways

…and lots more.

**Creating an Application with Create React App**

Let’s make use of these great features and create a “Hello World!” application with Create React App.

To use create-react-app to give us a start-point, we can use npx to get a temporary version of create-react-app to use to create the application.

npx create-react-app hello\_world

cd hello\_world

npm start

Note: An alternative to using the npx prefix would be to globally install create-react-app and then use npm create-react-app hello\_world

The npm start command has bundled the source code and started the server, so we don’t need to run webpack separately anymore. It should have also opened the application in a new browser tab going to http://localhost:3000.

**Create-React-App Application Architecture**

Let’s take a look at the application architecture. Some of the files and directories should look familiar. There is the ‘package.json’, ‘.gitignore’ and ‘node\_modules’ in the top level directory and there are also the ‘public’ and ‘src’ directories. The ‘src’ directory is where our source code lives, and this is where we will be putting our React components.

**index.js: The application’s entry point**

The entry point to the application is index.js, which renders the top level React component to the page. The ReactDOM’s render method takes the React component to be rendered, and the DOM element it should be appended to. The result of the render method is very similar to when we appended elements to the DOM by using element.appendChild.

ReactDOM.render(**<**App **/>**, document.getElementById('root'));

index.js accesses the element with an id of ‘root’ using getElementById from the DOM. (getElementById is just another method for querySelector). The ‘root’ element is defined in the public/index.html.

In most cases, the is the only time we will be directly accessing an element from the DOM in our React applications. Instead, we use a component rendering hierarchy, rendering the top level component to the page and then letting React handle the rest of the DOM manipulation for us. Therefore we will rarely (if ever) be using the document method createElement, or the element methods appendChild or textContent.

**Components**

So we know that the component defined in App.js is being rendered to the page by index.js. Let’s have a look at it. It is created as a function. This is actually a special kind of React component called a Stateless Functional Component which we will cover later.

Every React Component must implement a render method that returns a DOM element, an array of DOM elements or null. (We will be looking more at this new syntax in a moment.) As our App component is an SFC the function itself takes the place of the render method.

**Deleting Unneeded Boilerplate Code**

Create React App comes with some boilerplate code that we don’t need (you can see some of it being used to render information on the page). Let’s start by removing some of the unnecessary code and files. We don’t need the React logo file so we can start by deleting that.

rm src/logo.svg

Then is App.js we can delete the import of the logo file.

*// src/App.js*

**import** React **from** 'react';

*// DELETED*

**import** './App.css';

If we look at what the component returns from its function, we can see there is some new syntax there. It looks like HTML, but we know that it’s not because it is inside a JavaScript file. This syntax is called JSX and is what we can use in React to describe what we want to be rendered to the page. We are going to be looking more at JSX next. For now let’s delete the boilerplate code, so we can write our own.

**import** React **from** 'react';

**import** './App.css';

**function** App() {

**return** (

*// DELETED*

);

}

**export** **default** App;

**render**

As mentioned before, the Component’s function must return either a DOM element, an array of DOM elements or null. We could use React’s method to create a DOM element, createElement that takes 3 arguments:

1. the element tag
2. an object containing any classes or ids (in this case null)
3. the content

**function** App() {

**return** (

React.createElement('h1', **null**, 'Hello World!')

);

}

If you check the browser, you should now see ‘Hello World!’ instead of the React logo.

**JSX**

React provides us with an alternative syntax, preventing us from having to create elements in this way. The syntax is called JSX. It allows us to describe elements in a way that is very similar to HTML. JSX allows us to construct our user interfaces in a much quicker and more natural fashion, declaring our JSX rather than using JavaScript to manually create, manipulate and append DOM elements.

Let’s refactor our ‘h1’ using JSX. JSX looks a lot like HTML, but there are some key differences. For example, giving an element a class attribute, we use className="my\_class", instead of using class="my\_class" as we do in HTML. (This is because class is a reserved keyword in JavaScript.)

**function** App() {

**return** (

**<**h1 className**=**"title"**>**Hello World**!<**/h1>

);

}

Note: React and JSX are two independent technologies, but JSX was built with React in mind. Create React App has Babel already configured and Babel is transpiling the JSX into JavaScript, making it compatible with browsers.

**JSX Syntax Highlighting**

The JSX syntax highlighting might be broken as it’s being interpreted as JavaScript. To fix this you will need to get the Babel package for your editor. For Atom you can use Language Babel: <https://atom.io/packages/language-babel>

**Conclusion**

We have seen how to use Create React App to create a start point for our application and render to the screen using “Hello World”.

**Additional Resources**

There is also a React implementation for native mobile applications: <https://facebook.github.io/react-native/>