ReactJS Components and JSX

DEVELOPING APPLICATIONS USING REACTJS

Objectives

- To understand what a ReactJS component is
- To understand what JSX is and why it is used
- To be able to create, export and import components using JSX

Components

- ReactJS is fundamentally about Components
  - Developers need to think in Components when working with ReactJS!
- Components, according to the official ReactJS documentation:

  "React is a library for building composable user interfaces. It encourages the creation of reusable UI components which present data that changes over time."
- Can be created using raw JavaScript
  - Readability suffers!
- Can be a Class or a Function
- React.Component is an abstract base class
  - React components are typically subclasses that have a `render()` method (which is required)
- Component names should always begin with a capital letter (by convention)
  - Allows identification of native HTML tags and component tags
- Facebook created JSX to help integrate HTML and JavaScript specifically for ReactJS

3

Components as Classes

- If a Component is set up as a class then it must have a **`render`** method
- Required in all components and must return:
  - a representation of native DOM component (e.g. **`<div />`**) OR
  - a custom composite component defined by the developer OR
  - null or false to indicate that the component should not be rendered
- **`render`** method should be kept pure
  - Does not modify component's state
  - Returns same result each time it is invoked
  - Does not directly interact with the browser

```
import react from 'react';


default export class MyComponent
extends React.Component {
        render() {
                <h1>Hello World</h1>
        }
}
```

4

Components as Functions

- Not every component needs to *or should be* a class
- Functional components are useful when the component does not have state
- Like a render method in a Component Class, must return:
  - a representation of native DOM component (e.g. ** **) OR
  - a custom composite component defined by the developer OR
  - null or false to indicate that the component should not be rendered

```
import react from 'react';


const myComponent = (props) => {
       return (
              <h1>Hello World</h1>
       );
};
export default myComponent;
```

5

App.jsx file

- Simple examples of file *App.jsx*

App.jsx with class component within it:

```
import React from 'react';
class App extends React.Component {
    render() {
        return (
            <React.Fragment>
                <h1>Hello, World!</h1>
                <p>I am a paragraph</p>
            </React.Fragment>
        );
    }
}
export default App;
```

App.jsx with functional component within it:

```
import React from 'react';
const App = () => {
        return (
            <React.Fragment>
                <h1>Hello, World!</h1>
                <p>I am a paragraph</p>
            </React.Fragment>
        );
}
export default App;
```

- ES2015+ syntax used for imports/exports, class declarations and class inheritance
- `return()` must have a wrapper element for multiple lines of HTML – React.Fragment can be used if there is no natural (or required) structural wrapper.

export default is used to export a single class from a JSX file as the default export. As most Components in React are created in their own JS module, it is most common to find this. Modules can have other classes and functions exported from them.

Changes to main.js

- Simple example of a *main.js* file to use a file called *App.jsx* and a component within it:

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

ReactDOM.render(
      <App />, document.querySelector('#app')
);
```

- Additional import to allow uses of classes from *.jsx* file
- `ReactDOM.render()` now contains `<App />` which is the Component defined in the *App.jsx* file Multiple exports from a JSX require a slightly different syntax, which is covered later in this guide.

JSX – The Language for React

- JavaScript Syntax Extension
- Not necessary to use but has some advantages:
  - Makes it easier and quicker to create templates for developers familiar with HTML
  - Faster as optimisation occurs when compiling code to JavaScript
- Looks like normal HTML code in most cases
- Can use JavaScript expressions within it

8

Example of a Component with JSX and expressions

```
import React from 'react';
const App = () => {
    let x = 10;
    let y = 20;
    return (
        <React.Fragment>
            <h1>Hello, World!</h1>
            <p>10 + 20 = {x + y}</p>
        </React.Fragment>
    );
}
export default App;
```

- **<React.Fragment>** lets you group a list of children without adding extra nodes to the DOM
- The use of { } in the return allows JavaScript expressions to be inserted as content into the HTML
  - These can be used to perform transforms of data to produce multiple elements from an array or array-like object
    - E.g. though the use of the array.map function

9

TypeScript and TSX – Type Safe React

- TypeScript (a superset of the JavaScript language) can be used in conjunction with React
- Requires some additional project dependencies to have the TSX files compiled
  - This will not be covered in this course
- Allows React applications to be type safe at development
  - Errors will be given on compilation of the code

- More information can be found on TypeScript and Flow (an alternative) at:

https://reactjs.org/docs/static-type-checking.html#file-extensions

10

More on JSX files

- Can contain more than one component
  - Means that components can be split down into more manageable 'chunks'
- Any components defined can be declared and used within the exported component
- Example:
  - *App.jsx* contains 3 components: App, Header, Content
  - App component contains a Header component and a Content component

```
// Render method inside class App…

      render() {
          return (
            <div>
                  <Header />
                  <Content />
            </div>
        )
    }
```

11

More on JSX files

- `export` does not have to be default
  - Components can be exported individually – start the class declaration with **`export`**

```
export class Header extends React.Component {…}
```

  - Exported components have to be imported individually into *main.js*

```
import {Header, Content} from './App';
```

- Components can then be used in further `ReactDOM.render()` methods to populate other elements in the *index.html* file

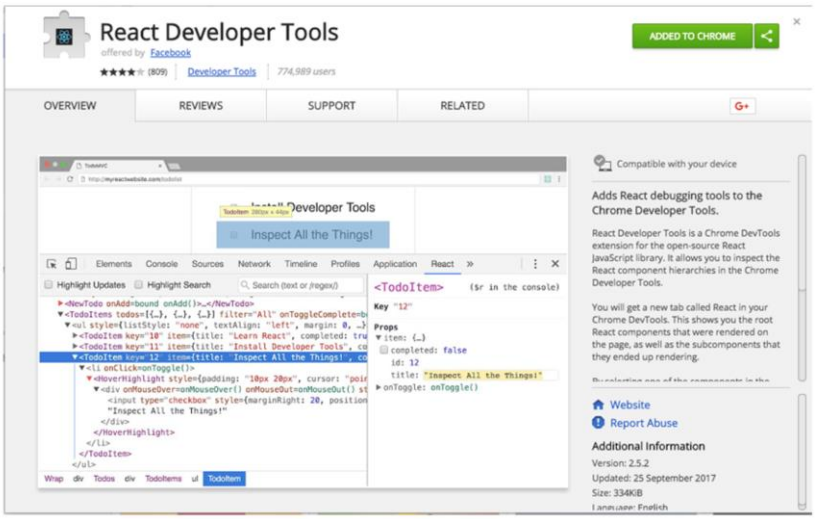
```
ReactDOM.render(
      <Header />, document.querySelector('header'));

ReactDOM.render(
      <Content />, document.querySelector('#content'));
```

12

Completing Exercise 3 and looking at the different solutions provided will lead to a thorough understanding of the use of multiple classes in a JSX file.

CHROME REACT DEVELOPER TOOLS

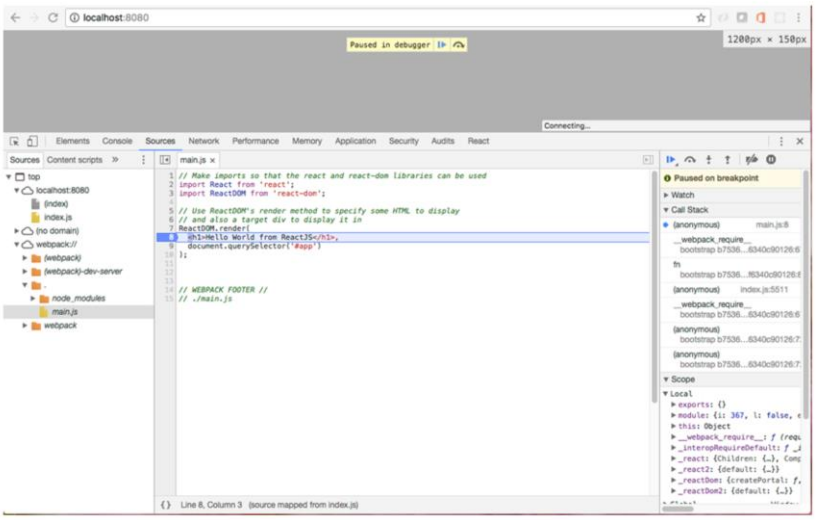


- Allows developers to see information in the Developer Tools for React
- It allows to access to how Components are rendered and also to their State, Props and other information, depending on its definition

13

(Image from https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi)

DEBUGGING USING THE SOURCE MAP



Using the devtool option of 'source-map' in the webpack.config.js file enables Chrome (and other browsers with advanced debuggers) to allow you to debug your JS and JSX files. They will appear in the . folder as shown on the image above. After selecting the appropriate file, you can set breakpoints in your React code and step and watch to your heart's content.

14

Objectives

- To understand what a ReactJS component is
- To understand what JSX is and why it is used
- To be able to create, export and import components using JSX

Exercise Time

- Complete EG04 – Thinking in React - Create Components

16

Appendix – Defining Components pre-ES2015

- Before ES2015 introduced classes to JavaScript, components were defined as follows:

```
MyComponent = React.createClass ({
    render: function() {
      return <div>Some Content</div>
    }
}
```

- This would produce a Component called MyComponent that could be rendered to the browser in the same way as using ES2015 classes:

```
class MyComponent extends React.Component {
    render() {
      return <div>Some Content</div>;
    }
}
```

17

If you are looking at other examples and tutorials, be wary if they are using pre-ES2015 syntax. This is a good indication that the material is old and therefore it may suggest practices, syntax and methodology that have been since deprecated.