Schedule

ReactJS:

- React app – how it works
  - React app – folder structure
- Thinking in JSX
- React Components
  - Class vs Functional component
  - Using state
- Selected topic:
  - ES6 class inheritance
  - ES6 arrow function

[code demo]

React app – How it works

React app – folder structure

- **build**: final, production-ready build (wont exist until npm build)
- **node_modules**: packages by npm or *yarn*
- **public**: static files,
  - NOT imported by app &
  - must maintain its file name (images, index.html…)

  → Cached by browser, never download again
- **src**: dynamic files
  - Imported by app
  - Change contents
  - Never worry about the browser using outdated copy

  e.g. The default `src/App.js`

React app – How it works

- JS files (components) in `src/` are translate into pure JS → inject into `public/index.html`

- `Run src/index.js`
  - the entry point, just like `main()`
  - Render components (input: props → output: HTML)

  → append into **`div#root`** & display

[code demo]

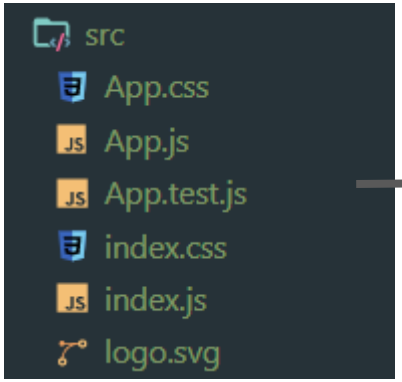
React app – How it works

[public/index.html]

```
<head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
            me="viewport" content="width=device-width, initial-scale=1" />
            "theme-color" content="#000000" />

            scription"
            nt="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <title>React App</title>
  </head>
```

No <script>



compile

inject into

public/index.html

React app – How it works

[public/index.html]

```
<body>
    <noscript>You need to
to run this app.</noscript>
    <div id="root"></div>
</body>
```

No content

[src/index.js]

```
import React from 'react';
import ReactDOM from 'react-dom';

class Test extends React.Component {
  render() {
    return <h1>Hello World!</h1>;
  }
}

ReactDOM.render(<Test />, document.querySelector('#root'));
```

src/index.js

HTML string

public/index.html

React app – How it works

[src/index.js]

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

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

[src/App.js]

```
function App() {
  return (
    <div className="App">
      <h1>Hello World!</h1>
    </div>
  );
}

export default App;
```

JSX

- export default? ES6 module (just like Node module):
  - `module.exports` → `require()`
  - `export ….` → `import … from …`

Thinking in JSX

JSX

- JavaScript XML
- HTML in JavaScript
- Easier to write & add HTML in React
- Shortcut for `React.createElement()`
  - Recall: `document.createElement()`

```
const div = document.createElement('div');
outer.classList.add('App');
const h1 = document.createElement('h1');
h1.textContent = 'Hello World!';

div.appendChild(h1);
```

```
function App() {
  return (
    <div className="App">
      <h1>Hello World!</h1>
    </div>
  );
}

export default App;
```

Coding JSX

- JSX allow to write HTML elements in JavaScript and place them in the DOM
  - without any `createElement()` and/or `appendChild()` methods.

- JSX **converts** HTML tags into react elements.

You are not required to use JSX, but .. why not? :D

```
const div = document.createElement('div');
outer.classList.add('App');
const h1 = document.createElement('h1');
h1.textContent = 'Hello World!';

div.appendChild(h1);
```

```
function App() {
  return (
    <div className="App">
      <h1>Hello World!</h1>
    </div>
  );
}

export default App;
```

JSX syntax

- Expressions
  - Written inside { }
  - Expression can be variable, property or any valid JS expression

```
const myelement = <h1>React is {5 + 5} times better with JSX</h1>;
```

- Multiple lines HTML
  - Put inside ()

```
const myelement = (
    <ul>
        <li>Apples</li>
        <li>Bananas</li>
        <li>Cherries</li>
    </ul>
);
```

JSX syntax

- **Note**: One top level element
  - The HTML code MUST be wrapped in ONE top level element

  e.g. wrap 2 headers inside one DIV element

```
const myelement = (
    <div>
        <h1>I am a Header.</h1>
        <h1>I am a Header too.</h1>
    </div>
);
```

- **Note**: Elements Must be Closed
  - JSX follows XML rules → HTML elements MUST be properly closed
  - Close empty elements with />

```
const myelement = <input type="text" />;
```

Selected topic:
ES6 class inheritance

Recall: ES6

- ES6?
  - ECMAScript 6 (ECMAScript 2015) – standard of JavaScript

- Class in ES6:

```
class Car {
    constructor(name) {
        this.brand = name;
    }

    present() {
        return 'I have a ' + this.brand;
    }
}

mycar = new Car("Ford");
mycar.present();
```

ES6 Class inheritance

- Class inheritance:
  - Keyword: `extends`
  - Child class inherits all the methods from the parent (super) class

```
class Model extends Car {
    constructor(name, mod) {
        super(name);
        this.model = mod;
    }
    show() {
        return this.present() + ', it is a ' + this.model
    }
}
mycar = new Model("Ford", "Mustang");
mycar.show();
```

- **constructor**: MUST invoke `super()`

  → Get access to parent properties & methods

- Use `this` to access parent's properties & methods

React components

- Independent and reusable bits of code
- are like **functions that return HTML** (required) via `render()`

TWO types of component:

- Class component
- Functional component

```
class Car extends React.Component {
    render() {
        return <h2>I am a Car!</h2>;
    }
}
```

```
function Car() {
    return <h2>Hi, I am also a Car!</h2>;
}
```

```
ReactDOM.render(<Car />, document.getElementById('root'));
```

** Component name MUST start with an uppercase letter*

Function vs Class component

- Function component
  - Function returns HTML

- Class component
  - A lot more functionality (lifecycle)
  - **State**

Component Constructor

- Called when the component gets initiated
  - initiate the **component's properties**
  - inherit parent component super()

- In React, component's properties should be kept in an object called **state**

  e.g. add color property & use it in render()

```
class Car extends React.Component {
    constructor() {
        super();
        this.state = { color: "red" };
    }
    render() {
        return <h2>I am a {this.state.color} Car!</h2>;
    }
}
```

Using the state object

- Refer to the **state** object anywhere in the component by using syntax:

```
this.state.propertyname
```

```
class Car extends React.Component {
    constructor() {
        super();
        this.state = { color: "red" };
    }
    render() {
        return <h2>I am a {this.state.color} Car!</h2>;
    }
}
```

Changing the state object

- Use `this.setState()` method.

```
class Car extends React.Component {
    constructor() {
        super();
        this.state = { color: 'red' };
    }
    changeColor = () => {
        this.setState({ color: 'blue' });
    }
    render() {
        return <>
            <h2>I am a {this.state.color} Car!</h2>
            <button onClick={this.changeColor}>Change color</button>
        </>
    }
}
```

Handling click event

- When state object changes → the component re-renders.

Important note on State

Always use the `setState()` method
to change the state object.

- it will ensure that the component knows its been updated

→calls the `render()` method

→(and all the **other lifecycle methods**) ???

Later

Handling click event

`addEventListener`? → No, React makes it easier

- Attribute: `onClick`

```
class Car extends React.Component {
    constructor() {
        super();
        this.state = { color: 'red' };
    }
    changeColor = () => {
        this.setState({ color: 'blue' });
    }
    render() {
        return <>
            <h2>I am a {this.state.color} Car!</h2>
            <button onClick={this.changeColor}>Change color</button>
        </>
    }
}
```

???

Selected topic:
ES6 arrow function

ES6 arrow function

- Shorter syntax

```
(param1, param2, … paramN) => {
    // statements
}
```

Problem with this?

```
function hello(name) {
    console.log('Hello ' + name);
}
```

```
const hello = (name) ⇒ {
    console.log('Hello ' + name);
}
```

Handling click event

`addEventListener`? → No, React makes it easier

- Attribute: `onClick`

```
class Car extends React.Component {
    constructor() {
        super();
        this.state = { color: 'red' };
    }
    changeColor = () => {
        this.setState({ color: 'blue' });
    }
    render() {
        return <>
            <h2>I am a {this.state.color} Car!</h2>
            <button onClick={this.changeColor}>Change color</button>
        </>
    }
}
```

Arrow function

Problem with this?

- **regular functions**: `this` = *the object that called the function*: the window, the document, a button or whatever

→ `bind()`

```
function hello(name) {
    console.log('Hello ' + name);
}
```

- **arrow functions**: `this` always = *the object that defined the arrow function*

```
const hello = (name) ⇒ {
    console.log('Hello ' + name);
}
```

Components in Files

- **Note**: the file HAS TO `export default function / class.`

```
class Car extends React.Component {
    render() {
        return <h2>Hi, I am a Car!</h2>;

    }
}

export default Car;
```

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

ReactDOM.render(<Car />, document.getElementById('root'));
```

- A file may contain more than one function / class components

```
export class Car {…}
export class Truck {…}
    → import {Car, Truck} from './App.js';
```

Components in Components

- Refer to components inside other components

```
class Garage extends React.Component {
    render() {
        return (
            <div>
                <h1>Who lives in my Garage?</h1>
                <Car />
            </div>
        );
    }
}

ReactDOM.render(<Garage />, document.getElementById('root'));
```

More next week!

References

ReactJS:

- W3schools React Tutorials: https://www.w3schools.com/react/default.asp

- Udemy - Complete React Developer in 2020 (w Redux, Hooks, GraphQL)