Lecture 11

Brief Introduction to React.js

Contents

JavaScript ES6

JavaScript revisions

- Since its introduction in 1995, JavaScript has gone through many revisions, two of them are considered "major" revisions
- ECMAScript 2009 was the first major revision (a.k.a ES5)
  - `"strict mode"`, JSON support
  - `String.trim()`, `Array` functions
  - Many other features
- ECMAScript 2015 was the 2nd major revision (a.k.a ECMAScript 6 / **ES6**)
  - `let` and `const`, arrow functions, `for...of`, etc.
  - Classes
  - Promises
  - **ES6 Modules**
  - Many other features

ES6 Classes

- A class is a type of function, but instead of using the keyword `function` to initiate it, we use the keyword `class`, and the properties are assigned inside a `constructor()` method.

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

- Then you can create objects of this `Car` class:

```
const mycar = new Car("Ford");
```

Methods in ES6 Classes

- You can add methods in a class.
- Example, to create a method named `present`:

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

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

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

ES6 Class Inheritance

- To create a subclass, use the `extends` keyword.
- A subclass inherits all the methods from the superclass.

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

ES6 `Array.map()`

- The `map()` method transforms an array's items with a function and returns the transformed array.

```
const fruits = ['apple', 'banana', 'orange'];

const fruitList = fruits.map(
    fruit => "<li>" + fruit + "</li>"
);

console.log(fruitList);
```

- Result:

```
['<li>apple</li>', '<li>banana</li>',
          '<li>orange</li>']
```

ES6 Object Destructuring

- Similar to array destructuring, object properties can also be destructured.

```
const vehicle = {
    brand: 'Ford',
    model: 'Mustang',
}

function myVehicle({ brand, model }) {
    console.log('My ' + brand + ' ' + model + ' is awesome!');
}

myVehicle(vehicle);
```

ES6 Spread Operator for Arrays

- The spread operator (`...`) allows us to quickly copy all or part of an existing array or object into another array or object.

```
const numbersOne = [1, 2, 3];
const numbersTwo = [4, 5, 6];
const numbersCombined = [...numbersOne, ...numbersTwo];
console.log(numbersCombined);
```

- Result: `[ 1, 2, 3, 4, 5, 6 ]`

- Assign the 2 items from array `numbers` to variables and put the rest in an array:

```
const numbers = [1, 2, 3, 4, 5, 6];
const [one, two, ...rest] = numbers;
console.log(rest);
```

- Result: `[ 3, 4, 5, 6 ]`

ES6 Spread Operator or Objects

- We can use the spread operator with objects, too.
- Example of combining two objects:

```
const obj1 = {
    brand: 'Ford',
    model: 'Mustang',
    color: 'red'
}
const obj2 = {
    year: 2021, color: 'yellow'
}
const obj3 = { ...obj1, ...obj2 }
console.log(obj3);
```

- Result: `{ brand: 'Ford', model: 'Mustang', color: 'yellow', year: 2021 }`
- **Note:** the properties that match (i.e. `color`) are overwritten by the later object.

ES6 Modules

- ES Modules rely on the `import` and `export` statements.
- Named exports can be done in two ways.
- In-line, individual named exports:

```
// person.js
export const name = "Jesse";
export const age = 40;
```

- Or multiple variables at once:

```
// person.js
const name = "Jesse";
const age = 40;

export { name, age };
```

ES6 Modules

- You can have one (and ONLY one) `default export` in a file (module).

```
// message.js
const message = () => {
    const name = "Jesse";
    const age = 40;
    return name + ' is ' + age + 'years old.';
};

export default message;
```

ES6 Modules

- Import named exports from `person.js`:

```
import { name, age } from "./person.js";
```

- Import a default export from `message.js`:

```
import message from "./message.js";
```

Introduction to React.js

What is SPA vs MPA?

- **Single-Page Applications (SPA)**

  SPAs load the entire application at once, offering a seamless experience with dynamic updates without reloading the entire page.

  - Loads a single HTML page.
  - Updates content dynamically.
  - Faster experience (no full page reloads).

- **Multi-Page Applications (MPA)**

  MPAs consist of multiple HTML pages, where each page is loaded independently, leading to page reloads for every navigation.

  - Loads multiple HTML pages.
  - Slower due to full page reloads.

Basics of ReactJS

- **Declarative UI**

  React embraces a declarative approach to UI development, where you describe what your UI should look like rather than how to update it.

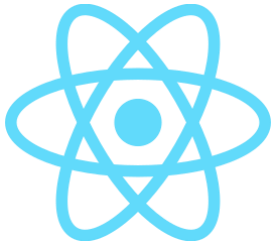
- **Component-Based**

  React promotes building user interfaces by composing reusable components that encapsulate specific functionality and structure.

- **Virtual DOM**

  React utilizes a virtual DOM, which is a lightweight representation of the actual DOM, for efficient UI updates.

What is React?



- React (a.k.a. ReactJS) is an open-source JS library maintained by Facebook
  - Used for creating *interactive* web pages (front-end) in a web application.

EJS (server-side rendering)

- To view a different web page, user has to load a different URL.
- Client sends an HTTP request, server responses with HTML.

React (client-side rendering)

- Client-side JS uses fetch to request data from server, then renders HTML based on server's response.
- Browser doesn't need to load another URL.
  - URL on address bar may change, but the page is not reloading.

Setting up React

- Install `create-react-app` globally

```
npm install -g create-react-app
```

(You only need to run this command once)

- Create a React application (this will create the project folder for you)

```
npx create-react-app myfirstreact
```

A folder named `myfirstreact` will be created under the current folder.

React project structure

- A React project is essentially a Node.js project
  - But we're not going to run the `.js` files using `node` command
- `public` folder contains static assets which can be accessed from `http://localhost:PORT/`
  - React actually uses `express` as a web server and serve static files in this `public` folder
- `src` folder contains React source files
- After creating the project, go to the project folder:

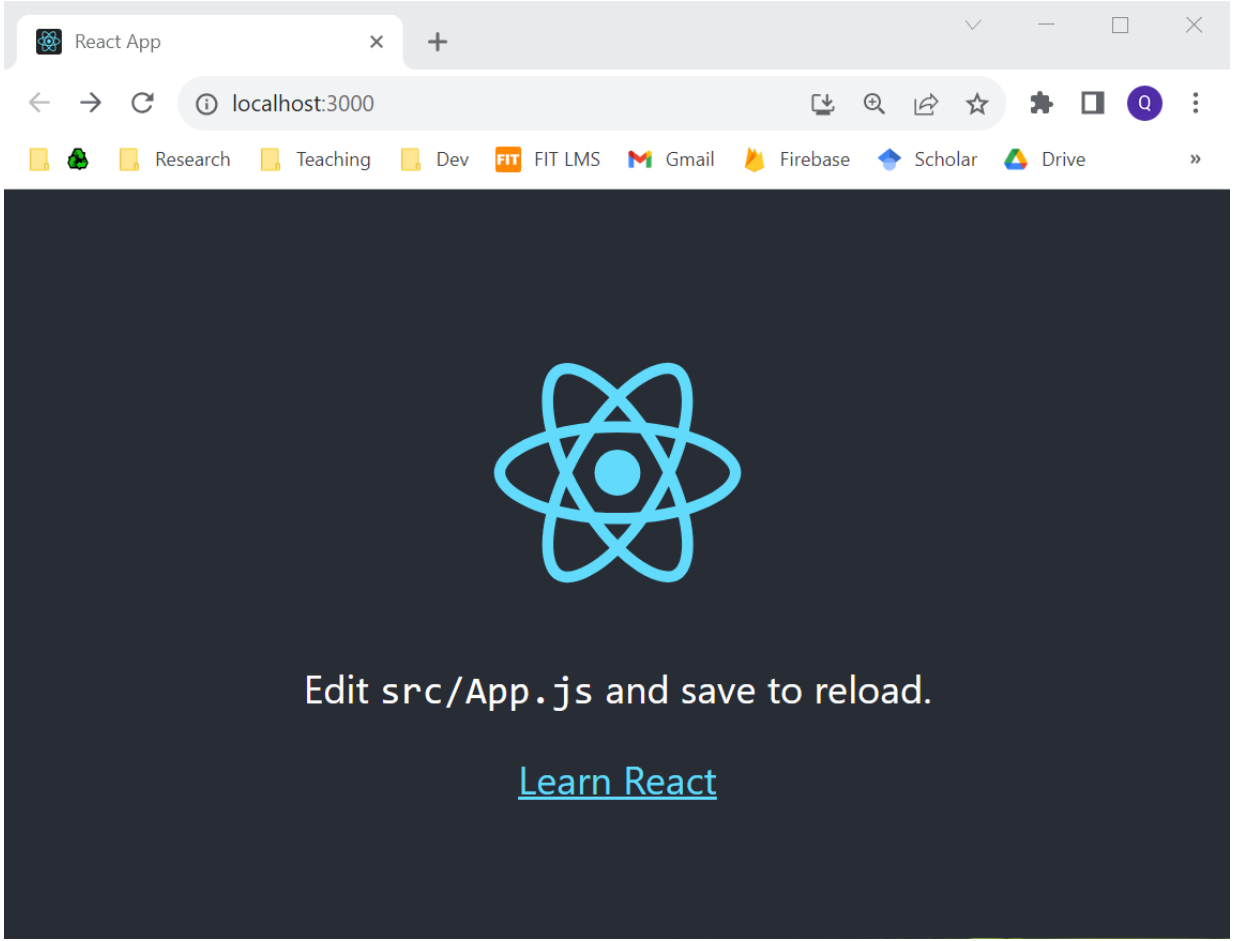
```
cd myfirstreact
```

- Start the application:

```
npm start
```

- A browser tap will be opened automatically.

What you'll see...



React's Key Concepts

1. Don’t touch the DOM. I’ll do it!

- Libraries & frameworks before:
  - Listen to user events
  - Directly change individual parts of the web page
- Problems:
  - Hard to see relationship between events & DOM changes
  - DOM manipulation takes long time ==> slow
- React solution:
  - User events affect the app's State. State controls what the page looks like
  - Manipulating a Virtual DOM before finally rendering the actual DOM => faster

React's Key Concepts

2. Build website like LEGO blocks

- Reusable components
  - e.g. Button, List, Product, Footer...
- Small components put together ==> bigger component!
- Can move components to other projects

React's Key Concepts

3. Unidirectional data flow

- Data only flow from the top-level component to child components
  - Data never move up
  - All the changes can only flow down from parent component to child components
- Anytime we want to change the webpage, we change the state

React's Key Concepts

4. **React builds UI only** (the rest is up to you)

- Unlike AngularJS, which is a MVC framework, React is just a UI **library**
  - React only provides the "view" part of the web application (front-end).
  - So we need some kind of back-end (can be Node.js back-end, can be others)
- React everywhere principle:
  - React project can build cross-platform UI
  - Web, Mobile, Desktop, VR...
  - `react-dom`: specific library to build for the web platform

How a React app works?

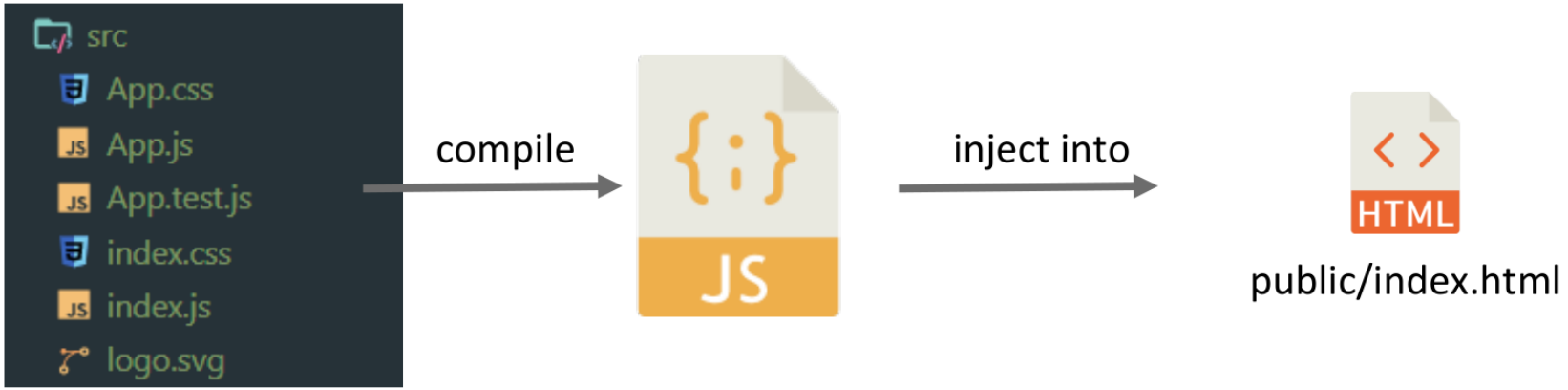
```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

- JS files (components) in `src/` are translate into pure JS
  - Then injected into `public/index.html`
- Run `src/index.js`
  - The entry point, just like `main()` in Java
- Render components
- Append into `div#root` & display on browser

How a React app works?

index.html

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



A first look at `App.js`

```
function App() {
  return (
    <div className="App">
      <h1>Hello, world!</h1>
    </div>
  );
}
export default App;
```

JSX
We'll learn about it now

- React uses the ES6 module pattern:
  - `require('...')` → `import ... from ...`
  - `module.exports` → `export default`

JSX

What is JSX?

- Stands for **JavaScript XML**
- Used to write 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;
```

What is 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.

```
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 - Expression support

- Embed JS expressions in curly braces { }

```
const intro = (
    <div>
        I'm {thisYear - user.bYear} years old.
    </div>
);
```

JSX - Attribute values

- Do not add quotes for expressions used as attribute value.

```
// right
const sqr = <Square value={i} />;

// wrong
const sqr = <Square value="{i}"
/>;
```

JSX - Single-line and multi-line elements

- Parentheses are NOT needed in multi-line elements
  - But recommended

```
// single-line
const myElement = <h1>I Love JSX!</h1>;

// multi-line
const listElement = (
    <ul>
        <li>PR1</li>
        <li>WPR</li>
        <li>MPR</li>
    </ul>
);
```

JSX's rule - Only <u>one</u> top-level element

```
// the problem
const myElement = (
    <h1>I Love JSX!</h1>
    <h1>Me too!</h1>
);
```

- Solutions:
  1. Wrap them in a parent `<div>` element
  2. Wrap them in a *fragment*. Like so:

```
// use fragment
const myElement = (
    <>
        <h1>I Love JSX!</h1>
        <h1>Me too!</h1>
    </>
);
```

JSX Elements Must be Closed

- JSX follows XML rules
  - HTML elements MUST be properly closed
- Close empty elements with `/>`

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

JSX - The `className` attribute

- Use `className` attribute instead of the `class` attribute in HTML.
- This is an exeption because `class` is a JS reserved keyword.

```
const myElement = <button className="square"></button>;
```

JSX - `if` statement support

- Cannot put `if` statement inside `{ }`
  - An `if` statement is not an *expression* anyway.
- → Workaround: use *ternary expression*

```
<h1>Good {h < 12 ? "morning" : "afternoon"}!</h1>
```

JSX - loop/collection support

- Cannot put `for` loops inside `{ }`
- → Workaround: prepare a collection of components in advance

```
const cars = [
    <Car brand="Toyota" />,
    <Car brand="Vinfast" />,
    <Car brand="Mercedes" />
];

return <ul>{cars}</ul>;
```

```
const cars = [
    'Toyota', 'Vinfast', 'Mercedes'
];

return (<ul>
    {cars.map(e => <Car brand={e} />)}
</ul>);
```

React components

- Independent and reusable bits of code
  - Components are what appear on the UI
  - Components return HTML
- Two types:
  - **Class component:** extends `React.Component` and has the `render()` method
  - **Function component:** a function which returns JSX, shorter code, behaves mostly like a Class component
- Difference:
  - You can add properties, methods, etc. to a Class component

Class components

- Constructor
  - Constructor inherited from `React.Component` receive HTML attributes as an argument (usually) called `props`
  - Can be overriden but not overloaded (no overloading in JS)
- State
  - `state` attribute and `setState()` method inherited from `React.Component`
  - React monitors the `state` object
  - When the `state` object changes, the component re-renders.

Creating a Class Component

- Create a component by extending `React.Component`. A component's properties should be kept in an object called `state`.
  - The `state` property is a special property.
  - A component re-renders if its `state` changes.

```
import React from 'react';

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

React Component `props`

- Use an attribute to pass a color to a component, and use it in the JSX of that component.

```
import React from 'react';
class Car extends React.Component {
    render() {
        return <h2>I am a {this.props.color} Car!</h2>;
    }
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Car color="red" />);
```

`props` in the Constructor

- If a component has a constructor function, the `props` should always be passed to the constructor and also to the `React.Component` via the `super()` method.

```
import React from 'react';

class Car extends React.Component {
    constructor(props) {
        super(props);
    }
    render() {
        return <h2>I am a {this.props.model}!</h2>;
    }
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Car model="Mustang" />);
```

React Function Component

- Here is the same component from previous examples, but created using a *Function component* instead.

```
function Car(props) {
    return <h2>I am a {props.color} Car!</h2>;
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Car color="red" />);
```

Nesting Components

- We can refer to components inside other components:

```
function Car() {
    return <h2>I am a Car!</h2>;
}
function Garage() {
    return (
        <>
            <h1>Who lives in my Garage?</h1>
            <Car />
        </>
    );
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Garage />);
```

React events

- React has the same events as HTML
  - onLoad, onClick, onMouseOver, onChange, onSubmit…
  - These events can only be attached to synthetic elements (HTML elements, not components)

- Reference: https://reactjs.org/docs/events.html

Handling Events

- React events are written in `camelCase` syntax
  - `onClick` instead of `onclick`
- React event handlers are written inside curly braces
  - `onClick={handleClick}` instead of `onClick="handleClick()"`

App.js

```
class App extends React.Component {
    handleClick = () => {
        this.setState({ count: this.state.count + 1 });
    };
    render() {
        return (
            <div>
                <MyComponent onClick={this.handleClick} />
                <p>Count: {this.state.count}</p>
            </div>);
    }
}
```

Handling Events

- In child component (`MyComponent.js`):

```
function MyComponent({ onClick }) {
    return (
        <button onClick={onClick}>Click me</button>
    );
}
```

- Clicking on this button will trigger the `handleClick` function in `App.js`, which modifies the state of the `App` component (the `count` variable).

React CSS styling

- Three ways:
  - Inline styling
  - CSS stylesheets
  - CSS Modules

- Reference: https://www.w3schools.com/REACT/react_css_styling.asp

Inline CSS styling

- Assign a JS object to the style attribute:

```
<h1 style={{color: "red"}}>Hello Style!</h1>
```

- You see double curly braces because there is an object inside the JSX expression.
- CSS properties are in `camelCase`
  - Similar to CSS properties in JavaScript
  - See https://www.w3schools.com/jsref/dom_obj_style.asp

React CSS stylesheets

- You can `import` an external `.css` file

```
import './App.css';
```

- ...then style the web page based on tag names, `className` and `id` attributes of the tags/components.

React CSS modules

- Create the CSS module with the `.module.css` extension.
  - Example: `my-style.module.css`

```
.bigblue {
    color: DodgerBlue;
    padding: 40px;
    font-family: Sans-Serif;
    text-align: center;
}
```

- ...then `import` it into the `.js` file of your component:

```
import styles from './my-style.module.css';
```

- ...and use it:

```
const Car = () => {
    return <h1 className={styles.bigblue}>Hello Car!</h1>;
}
```