Lecture 9

React

Acknowledgement: https://www.w3schools.com/REACT

https://reactjs.org/tutorial/tutorial.html

React

- React is a JavaScript library for building user interfaces
- Created by Facebook
- React is used to build single-page applications
- React allows us to create reusable UI components

How React works

- React creates a VIRTUAL DOM in memory
- Instead of manipulating the browser's DOM directly, React creates a virtual DOM in memory, where it does all the necessary manipulating, before making the changes in the browser DOM
- React finds out what changes have been made, and changes only what needs to be changed

Node.js

- JavaScript runtime environment that runs on the V8 engine
- Executes JavaScript code outside a web browser
- Open-source and cross-platform
- Can be used as a backend server so your application is in all one language
- Runs single-threaded, non-blocking, and asynchronous
- Download here: https://nodejs.org/en/download

npm and npx

- npm
  - Stands for node package manager
  - The dependency/package manager you get out of the box when you install Node.js
  - Provides a way for developers to install Node.js packages
- npx
  - Stands for node package execute
  - An npm package runner that can execute any package that you want from the npm registry without even installing that package

Create React App

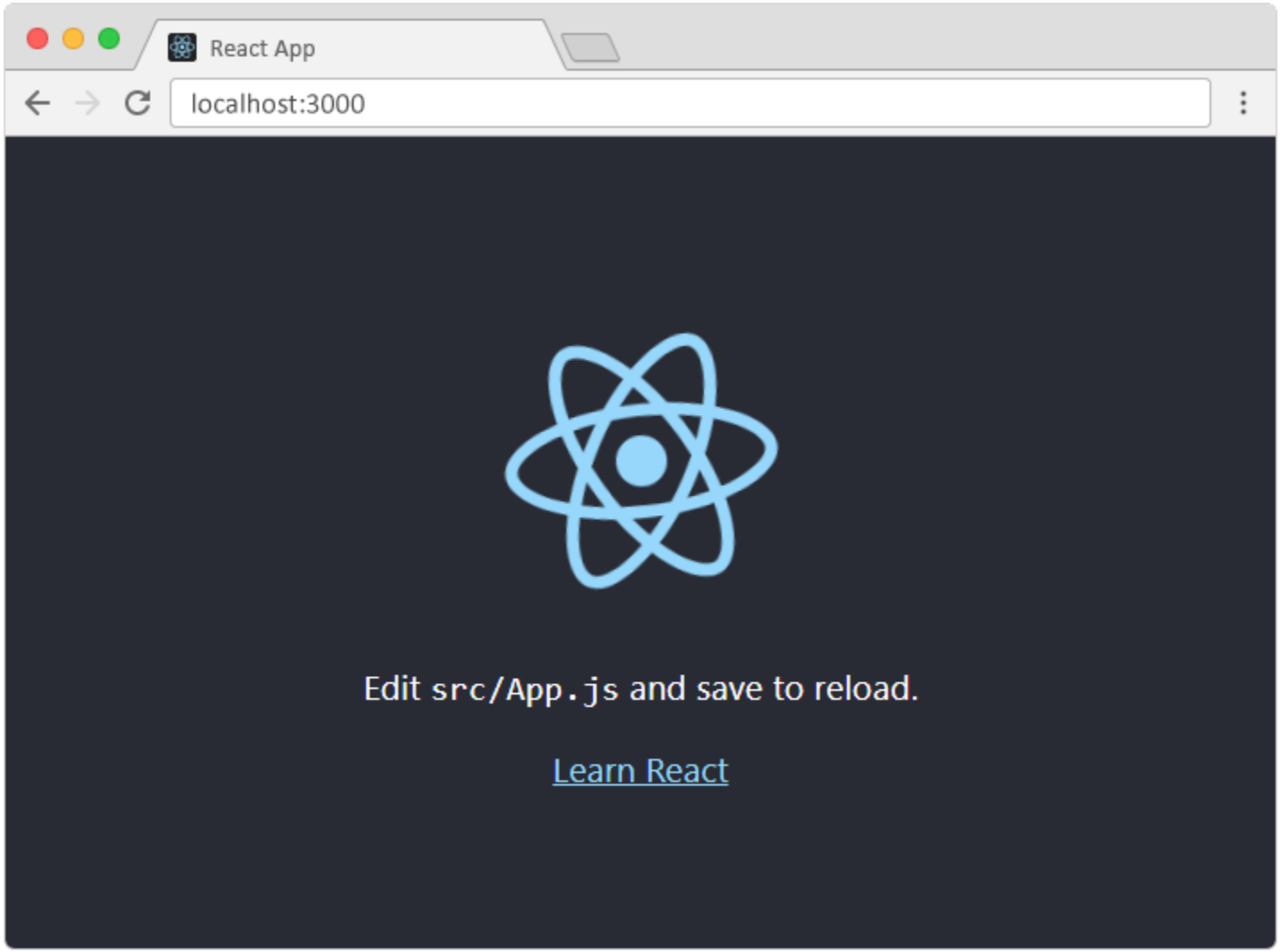
- The create-react-app tool is an officially supported way to create React applications
- Node.js is required to use create-react-app
- Open terminal in the directory you want to create your application and run this to create a React application named my-react-app:

```
npx create-react-app my-react-app
```

- Run this command in my-react-app directory to start app

```
npm start
```

Create React App



React Components

- Components are the building blocks of React
- They are independent and reusable bits of code
- They serve the same purpose as JavaScript functions, but work in isolation and return HTML
- Components come in two types:
  - Class components
  - Function components
- Function components are now more commonly used with Hooks

Class Component

- A class component must include the extends React.Component statement. This statement creates an inheritance to React.Component, and gives your component access to React.Component's functions
- The component also requires a render() method, this method returns JSX

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

Function Component

- A Function component also returns JSX, and behaves much the same way as a Class component, but Function components can be written using much less code and are easier to understand
- Same example but using a Function component instead:

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

Rendering a Component

- To use this component in your application, use the component as a tag <Car />

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

function Car() {
  return <h2>Hi, I am a car!</h2>;
}
ReactDOM.render(<Car />, document.getElementById('root'));
```

Props

- Components can be passed as props, which stands for properties
- Props are like function arguments, and you send them into the component as attributes

```
function Car(props) {
  return <h2>I am a {props.color} car!</h2>;
}

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

Arrow Function Syntax

- An abbreviated syntax for defining functions, very common in React
- No need for the **function** keyword, instead define a variable and use () => {}

```
const Car = (props) => {
    return <h2>I am a {props.color} car!</h2>;
}

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

Components in Components

- We can refer to components inside other components as shown here:

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

Components in Separate Files

- It is recommended to split your components into separate files

```
// Car.js
function Car() {
  return <h2>Hi, I am a Car!</h2>;
}
export default Car;

// Project.js
import React from 'react’;
import ReactDOM from 'react-dom’;
import Car from './Car.js’;

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

React Events

- Can perform actions based on user events, like HTML
- Same events as HTML, but in camel case: onClick, onChange, etc.

```
function Football() {
  const shoot = () => {
    alert("Great Shot!");
  }
  return (
    <button onClick={shoot}>Take the shot!</button> );
  }
const root = ReactDOM.createRoot(document.getElementById('root’));
root.render(<Football />);
```

Wake-up!

- https://youtu.be/8K_Tc-R7qZA

React Hooks

- Hooks were added to React in version 16.8.
- Hooks allow function components to have access to state and other React features
- Because of this, class components are generally no longer needed
- Hooks allow us to "hook" into React features such as state and lifecycle methods
- There are 3 rules for hooks:
  - Hooks can only be called inside React function components
  - Hooks can only be called at the top level of a component
  - Hooks cannot be conditional

React useState Hook

- The React useState Hook allows us to track state in a function component
- State generally refers to data or properties that need to be tracked in an application
- To use the useState Hook, we first need to import it into our component

```
import { useState } from "react";
```

Initialize useState

- We initialize our state by calling useState in our function component
- useState accepts an initial state and returns two values:
  - The current state
  - A function that updates the state

```
import { useState } from "react";
function FavoriteColor() {
  const [color, setColor] = useState("");
}
```

Read State

- Use the state variable in the rendered component

```
import { useState } from "react";
import ReactDOM from "react-dom";

function FavoriteColor() {
  const [color, setColor] = useState("red");

  return <h1>My favorite color is {color}!</h1>
}

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

Update State

- To update our state, we use our state updater function
- We should never directly update state. Ex: color = "red" is not allowed

```
import { useState } from "react";
import ReactDOM from "react-dom";
function FavoriteColor() {
 const [color, setColor] = useState("red");
 return (
  <>
    <h1>My favorite color is {color}!</h1>
    <button type="button" onClick={() => setColor("blue")} >Blue</button>
  </>
) }

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

Update State

- The useState Hook can be used to keep track of strings, numbers, booleans, arrays, and objects

```
import { useState } from "react";
import ReactDOM from "react-dom";
function Car() {
  const [brand, setBrand] = useState("Ford");
  const [model, setModel] = useState("Mustang");
  const [year, setYear] = useState("1964");
  const [color, setColor] = useState("red");
  return ( <>
    <h1>My {brand}</h1>
    <p> It is a {color} {model} from {year}. </p> </>
  )
}

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

Update State

- Or, just use one state and include an object instead:

```
import { useState } from "react";
import ReactDOM from "react-dom";
function Car() {
const [car, setCar] = useState({ brand: "Ford", model: "Mustang", year: "1964", color: "red" });
return (
  <>
    <h1>My {car.brand}</h1>
    <p> It is a {car.color} {car.model} from {car.year}. </p>
  </>
)}
ReactDOM.render(<Car />, document.getElementById('root'));
```

Updating Objects and Arrays in State

- When state is updated, the entire state gets overwritten
- What if we only want to update the color of our car?
- If we only called setCar({color: "blue"}), this would remove the brand, model, and year from our state
- We can use the JavaScript spread operator (…) to help us

Updating Objects and Arrays in State

```
function Car() {
  const [car, setCar] = useState({  brand: "Ford", model: "Mustang", year: "1964", color: "red"});
  const updateColor = () => {
    setCar(previousState => {
      return { ...previousState, color: "blue" }
    });
  }
  return (<>
      <h1>My {car.brand}</h1>
      <p>
        It is a {car.color} {car.model} from {car.year}.
      </p>
      <button type="button“ onClick={updateColor}>Blue</button>
    </>
  )}
```

React useEffect Hooks

- The useEffect Hook allows you to perform side effects in your components
- Some examples of side effects are: fetching data, directly updating the DOM, and timers
- useEffect accepts two arguments, the second argument is optional
- useEffect(<function>, <dependency>)

React useEffect Hooks

1. No dependency passed:

```
useEffect(() => {
  //Runs on every render
});
```

2. An empty array:

```
useEffect(() => {
  //Runs only on the first render
}, []);
```

3. Props or state values:

```
useEffect(() => {
  //Runs on the first render
  //And any time any dependency value changes
}, [prop, state]);
```

React useEffect Hooks

```
import { useState, useEffect } from "react";
import ReactDOM from "react-dom";

function Counter() {
  const [count, setCount] = useState(0);
  const [calculation, setCalculation] = useState(0);

  useEffect(() => {
    setCalculation(() => count * 2);
  }, [count]); // <- add the count variable here

  return (
    <>
      <p>Count: {count}</p>
      <button onClick={() => setCount((c) => c + 1)}>+</button>
      <p>Calculation: {calculation}</p>
    </>
  );
}

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

CSS Styling with React - Inline

- Style an element with the **style** attribute by setting it to a JavaScript object
- Properties must be camelCase (background-color -> backgroundColor)

```
const Header = () => {
  return (
    <>
      <h1 style={{backgroundColor: "lightblue"}} >Hello Style!</h1>
      <p>Add a little style!</p> </>
  );
}
```

CSS Styling with React – JavaScript Object

- Create an object with styling information, and refer to it in the style attribute

```
const Header = () => {
  const myStyle = {
    color: "white",
    backgroundColor: "DodgerBlue",
    fontFamily: "Sans-Serif"
  };
  return (<><h1 style={myStyle}>Hello Style!</h1> <p>Add a little style!</p></>);
}
```

CSS Styling with React – CSS Stylesheet

- Define your css in another file and import it

App.css

```
body {
    color: "white";
    background-color: "DodgerBlue“;
  }
```

Index.js

```
import './App.css’;
const Header = () => {
  return ( <> <h1>Hello Style!</h1> <p>Add a little style!.</p> </> );
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Header />);
```

Material UI

- The React UI library that provides components to build your own design system and develop React applications faster
- https://mui.com/components