**React Js**

2. A 7m Overview of Web Development and React

All of this relates to react since where react is a javascript

framework for web applications. As a web app framework reacts optimizes how javascript is used on the

Web browser to make the app dynamic.

HTML CSS JS

|  |  |  |
| --- | --- | --- |
| **Skeleton** | **CLoths** | **Magical body** |

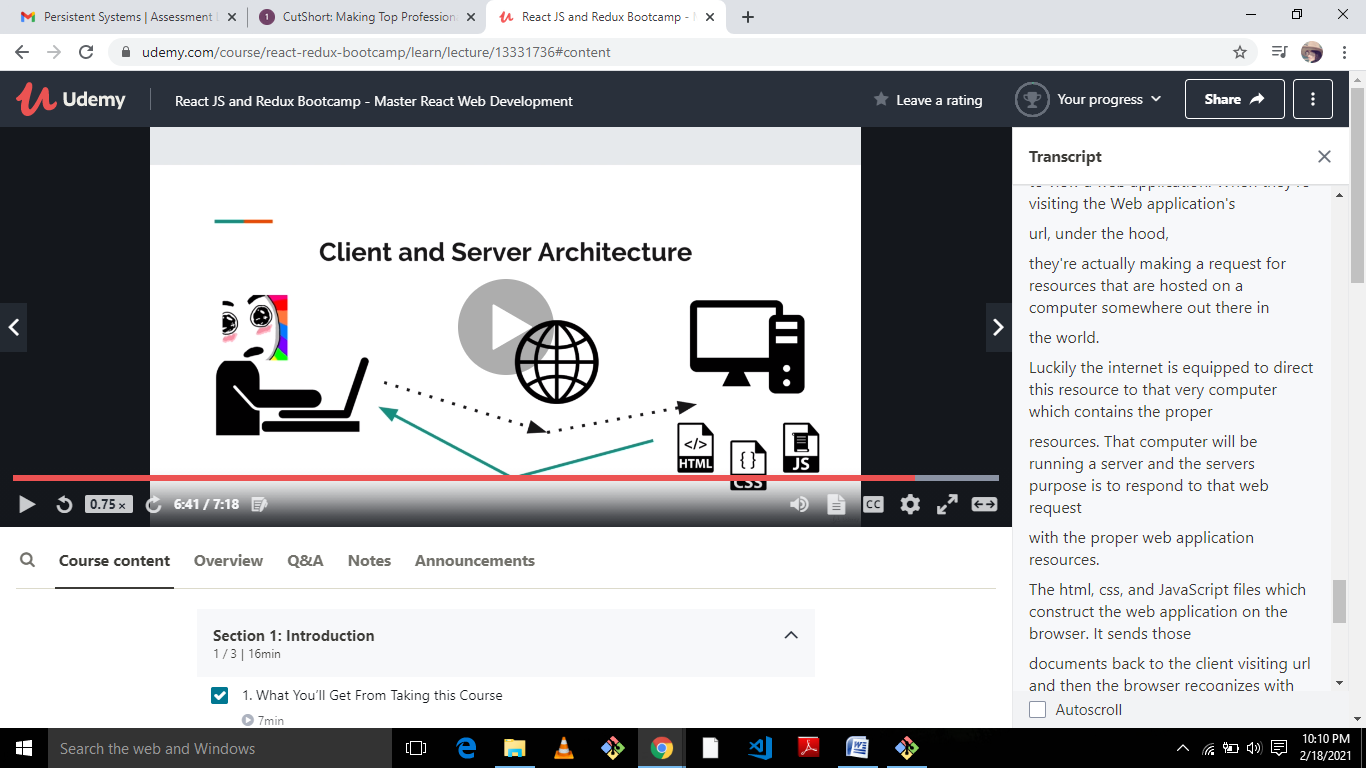
**Client-Server Architecture:**

First the user of the web application is the client. As a client,

they use their computer's web browser to view a web application. When they're visiting the Web application's url, under the hood, they're actually making a request for resources that are hosted on a computer somewhere out there in the world.

Luckily the internet is equipped to direct this resource to that very computer which contains the proper resources. That computer will be running a server and the servers purpose is to respond to that web request with the proper web application resources.

The html, css, and JavaScript files which construct the web application on the browser. It sends those documents back to the client visiting url and then the browser recognizes with the html, css and js that is now viewing a web application and it should present something amazing for the user.



[IMPORTANT] Course Repo and Software Installations

**Important! This is required reading to be successful in the course.**

**Here is the official course repository:**

<https://github.com/15Dkatz/react-bootcamp>

**Software Installations:**

**1. A Code Editor**

* For a code editor, I recommend Visual Studio Code: https://code.visualstudio.com/. I also recommend configuring it to open via the command line:
  + CMD-SHIFT-P
  + Enter “Shell Command”, and choose “Install ‘code’ command in path.
  + Then you can run $ code /folder to open folders in VSCode from your shell.

**2. A Command Line Application**

* For the command line application, on MacOS or Linux, I recommend the native Terminal application (or Iterm2)..

**WINDOWS USERS: I absolutely recommend downloading Git Bash. This course was recorded using a bash environment. Using Git Bash will make doing this course a smooth process. Here’s a great stackoverflow thread for configuring Git Bash on on Windows:**<https://stackoverflow.com/questions/42606837/how-do-i-use-bash-on-windows-from-the-visual-studio-code-integrated-terminal>

**3. Node.js.**

If you don’t already have Node.js installed, here is the download page: https://nodejs.org/en/download/. Once you have Node.js installed, run the following commands on the command line

* $ node -v
  + See a value like `v8.8.1` or higher
* $ npm -v
  + See a value like `v5.4.2` or higher

\*\*\*\*\*

With a Code Editor, Command Line App, and Node.js installed, you should be ready to go!

**Set up portfoilio app**

1.mkdir react-bootcamp

Cd react-bootcamp

Npx –version

npx create-react-app portfolio //create react app

goto command-pallete in visual studio and type “shell command”- add path

Run React app and project Structure

1.goto portfolio

npm run start :open localhost:3000

|  |  |  |
| --- | --- | --- |
| Directory/file name | | Description |
| Public/index.html | | The browser receives this document and then  uses it to display the content on the page. |
| Index.js | Import react | import sets up the file to write react code  by importing global values and methods that are defined within the react library. |
| Import react-dom | library takes care of actual application rendering meaning displaying the application on the browser  itself. |
| Sericeworker | | Use offline application |
| ReactDOM.render | | ReactDOM.render is a function which takes your react code and then inserts it into  the html. |
| App.js | | ReactDOM.render inserts this components JSX as HTML elements within the root div in the H.HTML. |
| Manifest.json | | Information about the project |
| Src/app.js,index.css,app.css,gitignore,  package.json | | React-script-build: The build script builds a production |

ReactDOM, Elements, and JSX

But we can create an element rather quickly. A react element is the smallest object within a react application.

They're pretty much copies of html elements.

So think of div tags, h1 tags, image tags etc. Those are all each simple elements, and a react element is

very much the same.

They represent an object at that level, like a div, or an H1, or an image in react.

We represent those elements with a syntax called JSX. Meaning javascript, JS,

and XML-like syntax, X. So XML is this other markup language like html which is based off the

pattern of using arrow like tags. For our first element let's add a div, and this will say react element.

So that becomes what we want to render within the application.

now we have the second argument of the render method, and in the second argument we specify where we want

this react element to get inserted. To do that we can access the core javascript document object, this

document object is provided to any javascript file running in the browser and there's way more to this

document object than meets the eye.

Src/index.js--🡪index.html(id=root)

A React Component(app.js)

we import the react library from the react module. So import react with a capital R from the lowercase react module.

But now since we're creating a component we're going to import a second variable from the react module:

the Component class. Now the Component class is not the default export of the react module.

Therefore in order to grab the component class you specify it as an optional import with a comma and

a pair of curly braces

after the main react import. So have the Component class with a capital C after the react library

now we can create the react component. A React component is defined by declaring a javascript class

with the class keyword. We're gonna call this class App since by convention a react component has the

same name as its file. And then we'll open up a class body with a pair of curly braces immediately.

We're gonna say that this class extends the component class from react with the extends keyword by extending

the component class.

We're making this our class a subclass of the react component.

The effect is that this App class as a react component now will have a bunch of methods and properties

that were originally declared in the react Component

class. So the first method that we want to take advantage of is the render method.

So add a render method to the app component. In the render method,

we define the structure of the component itself. The structure of a component, like an element, is defined

through JSX, except a component doesn't have to be just one element.

It can consist of many elements nested together.

So that's the distinction between an element and a component. An element is that smallest block.

o one div, one paragraph element, or one header. But the component can be many of these elements together

in order to structure an entire piece of UI for this app component.

Let's return an overall div. It's very important to have this return keyword with the parentheses.

It's a common mistake to be missing the return statement in the render method and then get confused

why the react app isn't showing anything so make sure to have that return keyword again.

Export App.js-> import in index.js

Classes, Inheritance, and a Closer Component Look

1. add the constructor method right away, This method allows us to define the properties the instances

of this class we'll have for the animal class that was name and age and to attach those properties.

We use this special this keyword.

So have this.name as well as this.age.

This keyword is a powerful one.

It's a special object that gets recreated uniquely for every instance of the class.

So one animal instance is going to have its own this object.

class Animal{

constructor(name,age){

this.name=name;

this.age=age

}

speak(){

console.log("I am",this.name,"I am",this.age,"years old")

}

}

const animal1=new Animal("Lion",40);

animal1.speak();

Inheritance:

one benefit is that we now have a lion class with all the properties and methods from the original animal

class but then it can define its own properties and methods

in addition to that. The one catch though is that the constructor for this needs to have the same starting

parameters as the original class so it needs a name and age but after that we can declare new inputs,

for the lion like furColor and speed.

If we're extending the animal but that code already exists in the constructor of the animal class.

So in this situation to invoke the constructor of the parent class we use these super keyword so we

can pass in the name any age to the constructor of our parent class which will take care of setting

this.name and this.age. Already within the line.

And after that we're gonna do similar code to set the furColor property to the incoming furColor value

and the speed to the incoming speed value.

class Lion extends Animal{

constructor(name,age,furColor,speed){

super(name,age);

this.furColor=furColor;

this.speed=speed;

}

roar(){

console.log("Rooooor I have ",this.furColor,"fur,and I can run",this.speed,"miler per hour");

}

}

const lion=new Lion("Mufasa",20,"golder",40);

lion.speak();

lion.roar();

console.log(lion);

States:

The state is in updatable structure that contains the data for a component.

setState:

The setState method is used by calling this.setState and its input is an object.The

keys in this object are whatever keys we want to update in the state.

So the challenge is that we still want the methods to have access to this object of the component.

Luckily in JavaScript we have a special .bind method for methods that allow us to pass this object

from one object to another.

That way we can bind the this object of the component itself to the helper method and it goes like this.

We'll call this.readMore.bind(this).

Remember it's a rule to never modify the state directly within react.

You should call setState to handle situations where you want values in the state to change.

You should always supply new objects as values to replace the current values in the state to reinforce

this rule.

class App extends Component{

constructor(){

super();

this.state={displayBio:false}; //change false to true then visible the state div on webpage

// this.readMore=()=>{

// console.log(this);

// }

//bind the readmore to the component of "this" object

this.toggleDisplayBio=this.toggleDisplayBio.bind(this);

}

toggleDisplayBio(){

this.setState({displayBio:!this.state.displayBio});

}

render(){

// const bio=this.state.displayBio?(

// <div>

// <p>I live in India.</p>

// <p>I like javascript, and ReactJs.</p>

// </div>

// ):null;

return(

<div>

<h1>Hello,</h1>

<p>My name is Monika.I'm a software engineer.</p>

{/\* {bio} \*/}

{

this.state.displayBio?(

<div>

<p>I live in India.</p>

<p>I like javascript, and ReactJs.</p>

<button onClick={this.toggleDisplayBio}>Show less</button>

</div>

):(

<div>

<button onClick={this.toggleDisplayBio}>Read more</button>

</div>

)

}

</div>

)

}

}

export default App

Component 2:Projects:

We can now use a map function to transform the project data items to individual pieces of JSX to

render.

So with that we have a callback function to pass into the map and the actual parameter to the callback

function is each project one at a time.

So it's gonna use the first project and then the second project and the third project firing this callback

function one at a time.

And for each item now let's have the return statement.

And this is going to return a div overall that renders the project object's title.

So have a pair of curly braces and then render the project title and this code is equivalent to what

we had before.

For each of the individual project items within the project array we're turning each one into a div

that consists of its title right.

Props:

In react, props are a way for parent components to pass down

data to child components.

{{}} -provide inline style

One for declare style and other brance for object

Styling the element:

1.import into js file:

import './index.css';

2.inside “index.html” file:

<link rel="stylesheet" href="index.css"/>

3.bootstrap: inside index.html:

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-BmbxuPwQa2lc/FVzBcNJ7UAyJxM6wuqIj61tLrc4wSX0szH/Ev+nYRRuWlolflfl" crossorigin="anonymous">

4.Fonts:

fonts.google.com

<link rel="preconnect" href="https://fonts.gstatic.com">  
<link href="https://fonts.googleapis.com/css2?family=**Akaya+Telivigala**&display=swap" rel="stylesheet">

**5.using class element:**

<img src={profile} alt="profile" className="profile" />

And inside .css file:

React from scratch:

Index.html

<!DOCTYPE html>

<html>

    <head></head>

    <body>

        <div id="root"></div>

        <script crossorigin src="https://unpkg.com/react@17/umd/react.development.js"></script>

        <script crossorigin src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>

        <script src="index.js"></script>

    </body>

</html>

Index.js

const element=React.createElement('div',null,'React element');

ReactDOM.render(element,document.getElementById("root"));

 Compiling, Bundling, and a Closer Look at JSX:

 babel is a technology

that does a translation job that we need. It compiles javascript code with newer syntax into the code

that the browser can actually understand

link: <https://babeljs.io/repl>

link: <https://parceljs.org/>

Parcel did its job of compiling the JSX react element code into a syntax that the browser can understand.

Goto the command line(project directory):

$npm init –y

$ npm install parcel-bundler --save

$npm i react react-dom –save

Package.json

{

  "name": "starter",

  "version": "1.0.0",

  "description": "",

  "main": "index.js",

  "scripts": {

    "start": "parcel index.html"

  },

  "keywords": [],

  "author": "",

  "license": "ISC",

  "dependencies": {

    "parcel-bundler": "^1.10.3"

}

Index.js

import React from 'react';

import ReactDOM from 'react-dom';

const element=React.createElement('div',null,'React element'); //instead of nulll we can use props name

ReactDOM.render(element,document.getElementById("root"));

Goto the <http://localhost:1234> on browser

Refactor the Portfolio to use Parcel

1.in portfolio app🡪

Package.json

{

  "name": "portfolio",

  "version": "0.1.0",

  "private": true,

  "dependencies": {

  },

  "scripts": {

  }

$ npm install parcel-bundler react react-dom –save

2. The reason that we're specifying this as the dev script and not the regular start script is a parcel

actually has a hot reloading capability in its server a feature that's very advantageous while developing

so as we make changes it will automatically reload those changes in the actual browser.

All right so let's try this out and see if it works.

Fire npm run dev in the command line.

So that is a new script and fire npm run dev and immediately you should see this error on the state

line, as the error says support for the experimental syntax class properties isn't currently enabled.

{

  "name": "portfolio",

  "version": "0.1.0",

  "private": true,

  "dependencies": {

    "@babel/plugin-proposal-class-properties": "^7.13.0",

    "parcel-bundler": "^1.12.5",

    "react": "^17.0.1",

    "react-dom": "^17.0.1"

  },

  "scripts": {

    "dev": "parcel src/index.html",

    "build": "parcel build src/index.html",

"start": "npm run build && live-server dist"

  },

  "devDependencies": {

    "@babel/core": "^7.13.10"

Npm run dev

3.npm install @babel/plugin-proposal-class-properties

4. with the live-server we

can tell this server module to load up this directory to anyone that requests it.

So the browser can actually get those contents as its sources.

Npm I live-server

The Necessity of Bundlers and Transpilers Review

A bundler's input is a project that consists of multiple JavaScript files.

The bundler will then package all of those files and build one overall JavaScript file.

This includes all custom javascript code in the project and all imported code from node modules like

react and react-dom for our bundler.

We're using parcel.

It's a great one with a minimal configuration and a very fast engine but there are other popular options

you might have heard of like webpack and browserify.

Bundlers are necessary because a browser doesn't support the import and export system.

 For our transpiler

We use a technology called babel. Babel does the crucial job of taking newer javascript and react syntax

and transforming it into a syntax that the browser is actually compatible with.

This allows us to use a technology like a babel to translate our clean modern JSX code into the

older equivalent JSX syntax that the browsers can handle.

And this applies to our React code.

The example we came across was JSX, a react specific syntax in order for the code to run on the browser.

This had to be transported to react.createElement the code under the hood that the browser can

understand.

The DOM and React’s Virtual DOM

* The Document Object Model is an API that allows us to manipulate html documents it takes an

html like structure and represents that same structure in a tree like structure, that is a hierarchy of

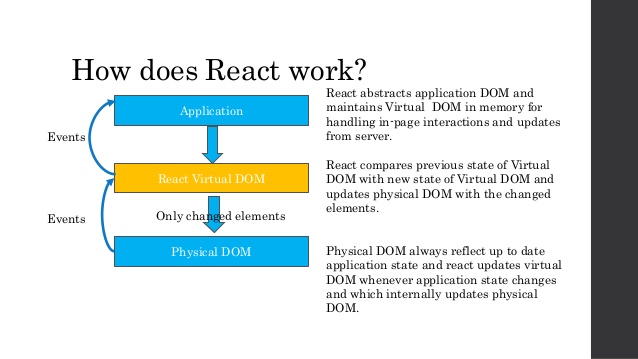
objects. Each node in the tree is an individual object.

Essentially there's the regular dom that the browser provides and then there's a virtual copy of this

DOM maintained internally in the react engine called the virtual DOM.

|  |  |
| --- | --- |
| Regular dom | Virtual dom |
| Slow and heavyweight | Lightweight and fast |

React use



Lifecycle of React:

lifecycle methods allow you to fire functionality at different

phases of a components life throughout the app.

1.fetch()

The fetch method allows you to make HTTP requests to external APIs. That way you have the ability

to load in data from any external service and really beef up your react application with more information

we'll also explore stateless functional components which are an alternative way to create components

without javascript classes.

lifecycle methods are based on the components

relationship with the document. Recall that react as a job of inserting a components JSX into the document.

|  |  |
| --- | --- |
|  |  |
| 1.componentDidMount(): | Fires when component inserted into DOM. |
| 2.componentWillUnMount(): | When component going to leave the DOM. |
|  |  |