Intro to Functional Programming with ReactJS

Week 4

16 Feb 24

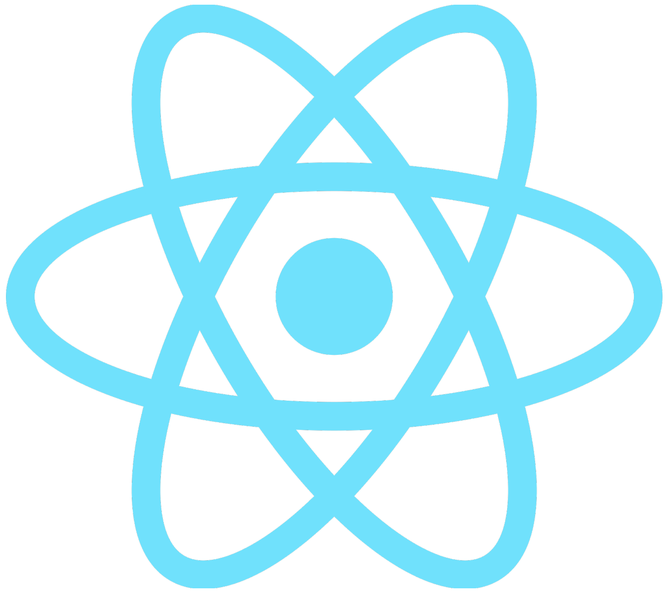
Functional Programming is a declarative programming paradigm where the program is constructed using functions

It helps in writing cleaner, and more maintainable code

Object-oriented Programming (OOP) is the other most notable paradigm in modern programming

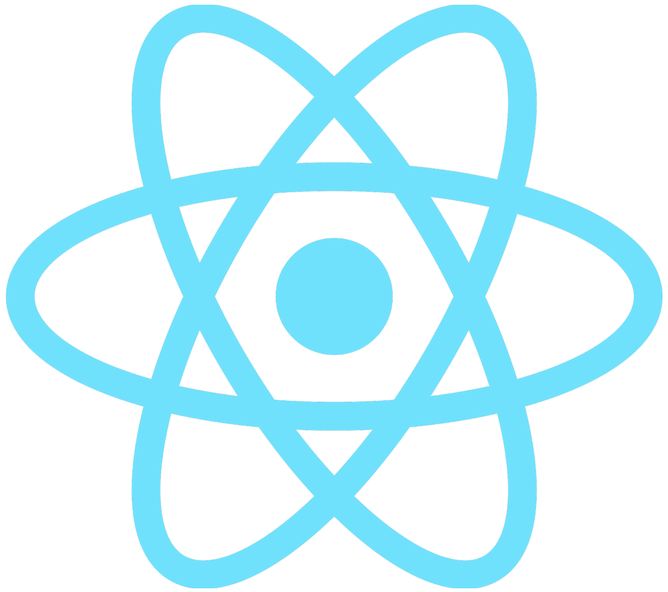
ReactJS

- ReactJS is an open-source JavaScript project developed and maintained by Meta
- React is a front-end library (and not a framework)
- React focuses on declarative syntax and famously utilises virtual DOM to render components on the screen



Who uses React?

- Facebook
- Instagram
- Shopify
- AirBnB
- Dropbox
- Netflix
- BBC
- and so many more…



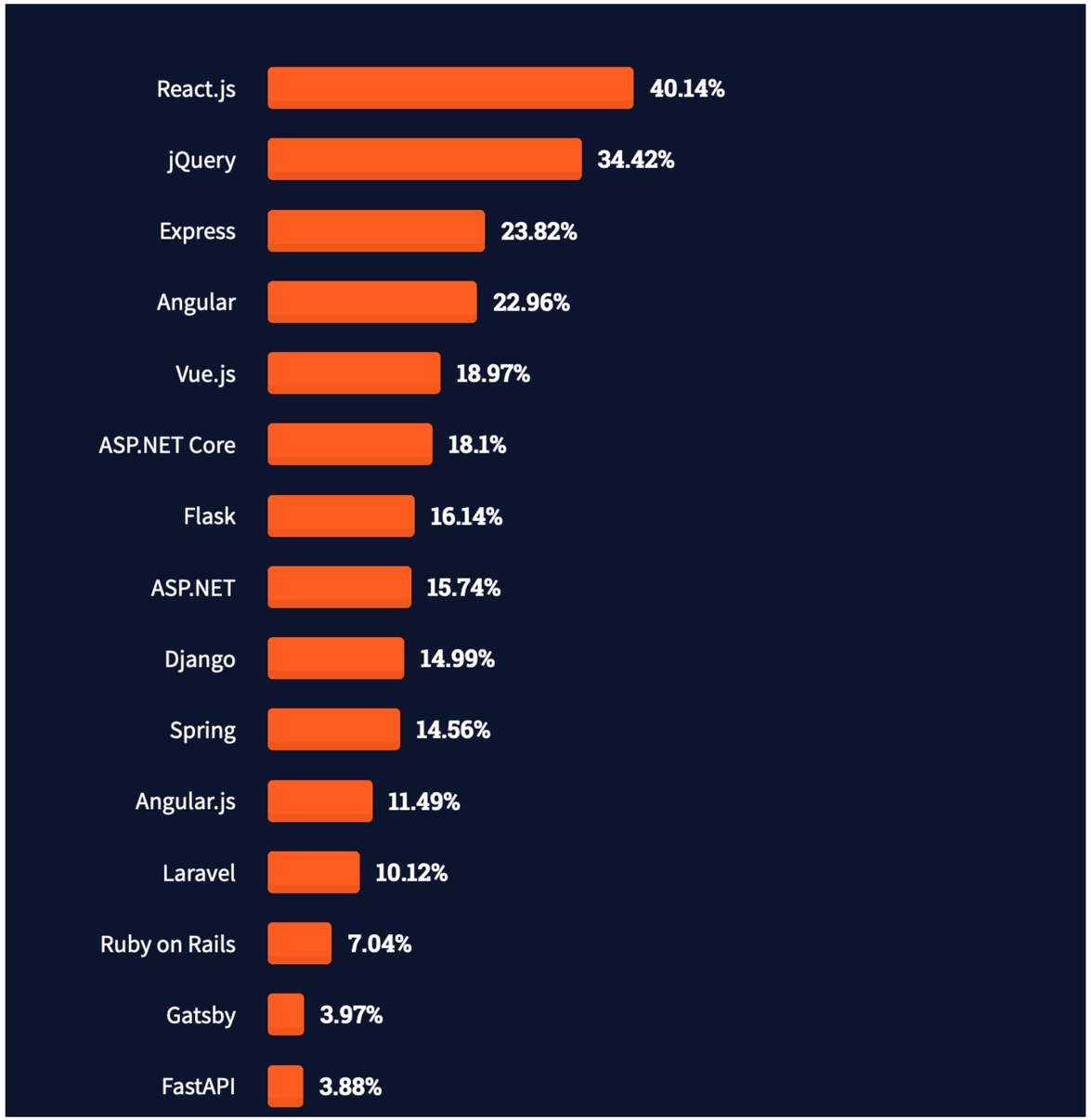
Popularity

Total downloads over the years

Popularity

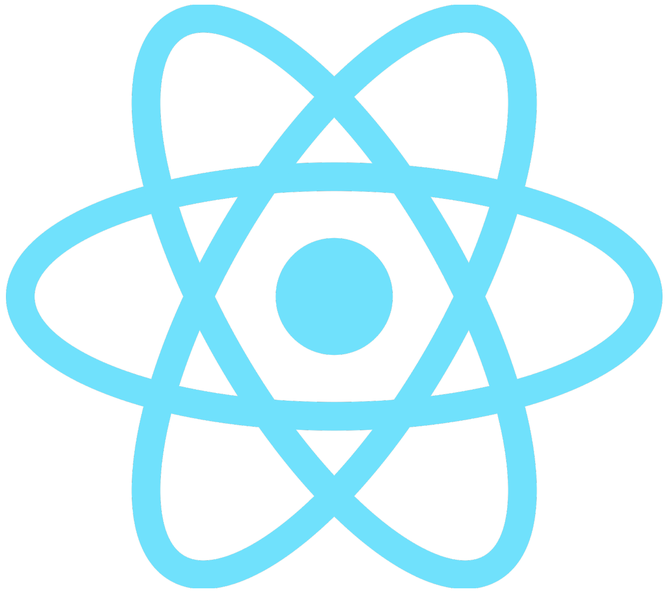
Popularity amongst developers

ReactJS is ranked consistently as the most loved web technology for years in the Stack Overflow Developer Surveys



Why use React?

- Less opinionated
- Declarative code
- A large community for support and third-party libraries
- Scalability
- Portability to iOS and Android apps with React-native



React Components

Building Blocks of React

- Components are independent blocks of code
- A group of components creates a page
- Components enable reusability, and maintainability

- At the end, components are just JavaScript Functions that return JSX

Example React Component returning JSX

```
function Welcome() {
  let name = "World"
  return (
    <h1>Hello, {name}</h1>
  );
}
```

Component names must start with an uppercase letter to distinguish from native HTML elements

Example React Component returning JSX

```
function Welcome() {
  let name = "World"
  return (
    <h1>Hello, {name}</h1>
  );
}
```

Logic

JSX

Use curly braces inside JSX to <u>escape</u> JavaScript

What is JSX?

- JSX is a templating language that extends JavaScript's features into HTML
- Looks very similar to HTML
- But we can use JavaScript inside JSX within curly braces

```
function Welcome() {
  let name = "World"
  return (
    <h1>Hello, {name}</h1>
  );
}
```

Using Components

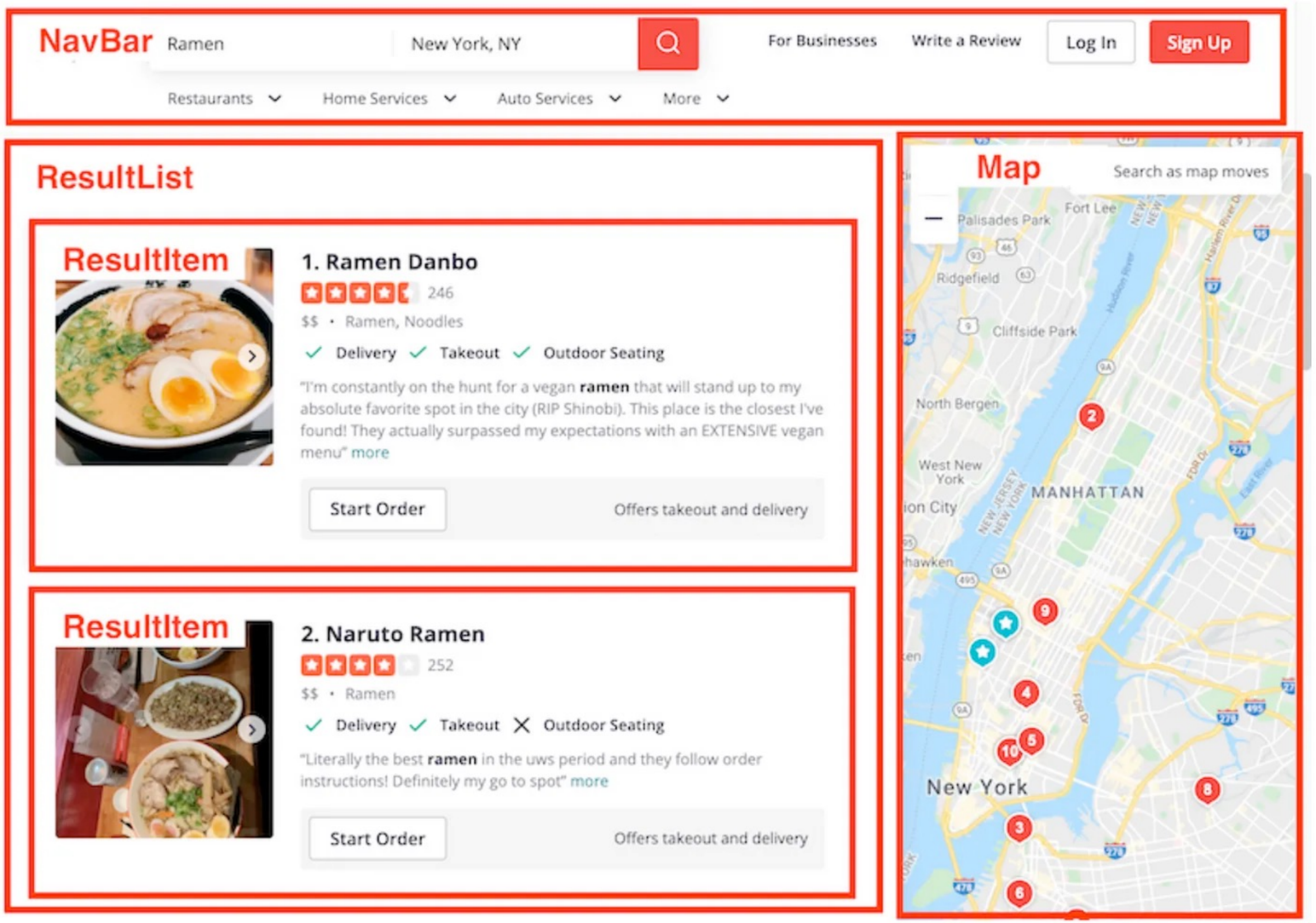
- Components can be used inside the code with <ComponentName /> syntax

```
function CatImage() {
  let imagePath = "/images/cat.jpg";

  return (
    <img src={imagePath} alt="Cat looking at the camera" />
  )
}
```

```
function Welcome() {
  let name = "World";

  return (
    <div>
        <h1>Hello, {name}</h1>
        <CatImage />
    </div>
  )
}
```

Working with Components

Component-based Designs



Component Props

Passing Data to Components

- `props` enable passing data from a parent component to a child component
- Data can be accessed in the child component with the `props` argument

```
function Welcome() {
  let name = "World";
  let catImgPath = "images/cat.jpg"
  let catImgDescription = "A cat looking at the camera"

  return (
    <div>
        <h1>Hello, {name}</h1>
        <CatImage path={catImgPath} description={catImgDescrition}/>
    </div>
  )
}
```

```
function CatImage(props) {
  return (
    <img src={props.imgPath} alt={props.imgDescription} />
  )
}
```

Parent Component

```
function Welcome() {
  let name = "World";
  let catImgPath = "images/cat.jpg"
  let catImgDescription = "A cat looking at the camera"

  return (
    <div>
        <h1>Hello, {name}</h1>
        <MyImage path={catImgPath} description={catImgDescrition}/>
    </div>
  )
}
```

Child Component

```
function MyImage(props) {
  return (
    <img src={props.path} alt={props.description} />
  )
}
```

Passing Props from Child to Parent

Parent Component

```
import Child from './Child';

function Parent() {

  const getData = (data) => {
    console.log(data); // LOGS DATA FROM CHILD
  }

  return (
    <div className='App'>
      <Child
        func={pull_data}
      />
    </div>
  );
}
```

Child Component

```
const Child = (props) => {

  props.func('Hello World');

  return (
    <>
      <h1>I am the Child Component!</h1>
    </>
  );
}

export default Child;
```

Note: Object Destructuring

- Objects in JavaScript can be destructured using the following syntax

```
let location = {
  "council": "wandsworth",
  "city": "london",
  "country": "united kingdom"
}
```

```
let {council, city, country} = location

console.log(council) // wandsworth
console.log(city) // london
console.log(country) // united kingdom
```

Note: Object Destructuring

- So the previous component code can be written as

```
function MyImage(props) {

  const {path, description} = props

  return (
    <img src={path} alt={description} />
  )
}
```

Or

```
function MyImage({path, description}) {

  return (
    <img src={path} alt={description} />
  )
}
```

Events in React

- Events in React can be called directly on the HTML elements

```
function MyButton() {
  const clickHandler = (e) => {
    alert('Clicked')
  }

  return (
    <button onClick={clickHandler}>Click here</button>
  )
}
```

Event object

Access to all event details

Event listener

All JS event names are prefixed with on in JSX. E.g. onClick, onMouseOver etc.

Bringing it all together

Setting up a React project

React (and most other libraries/frameworks) are installed via npm using the terminal

NPM is the official package manager for NodeJS applications

Search for available packages on npmjs.com

create-react-app

Create-react-app is the officially recommended way to setup your React SPA.

It relies on Webpack bundling system to setup the project

Update 2023:

You can also setup your own react projects using other bundlers like Vite

Vite commands

npm create vite // creates a react app in my-app directory

npm run dev // starts a development server

npm run build // build the project for production

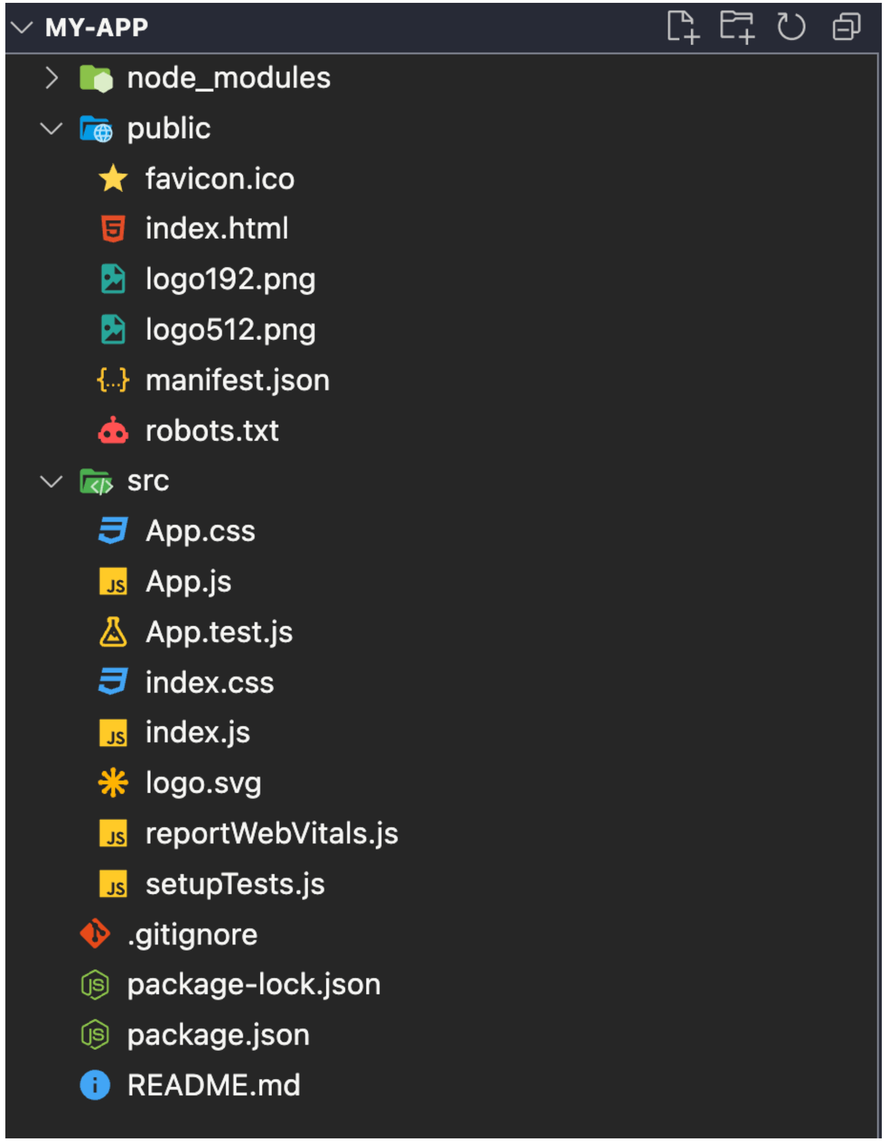
CRA commands

npx create-react-app my-app // creates a react app in my-app directory

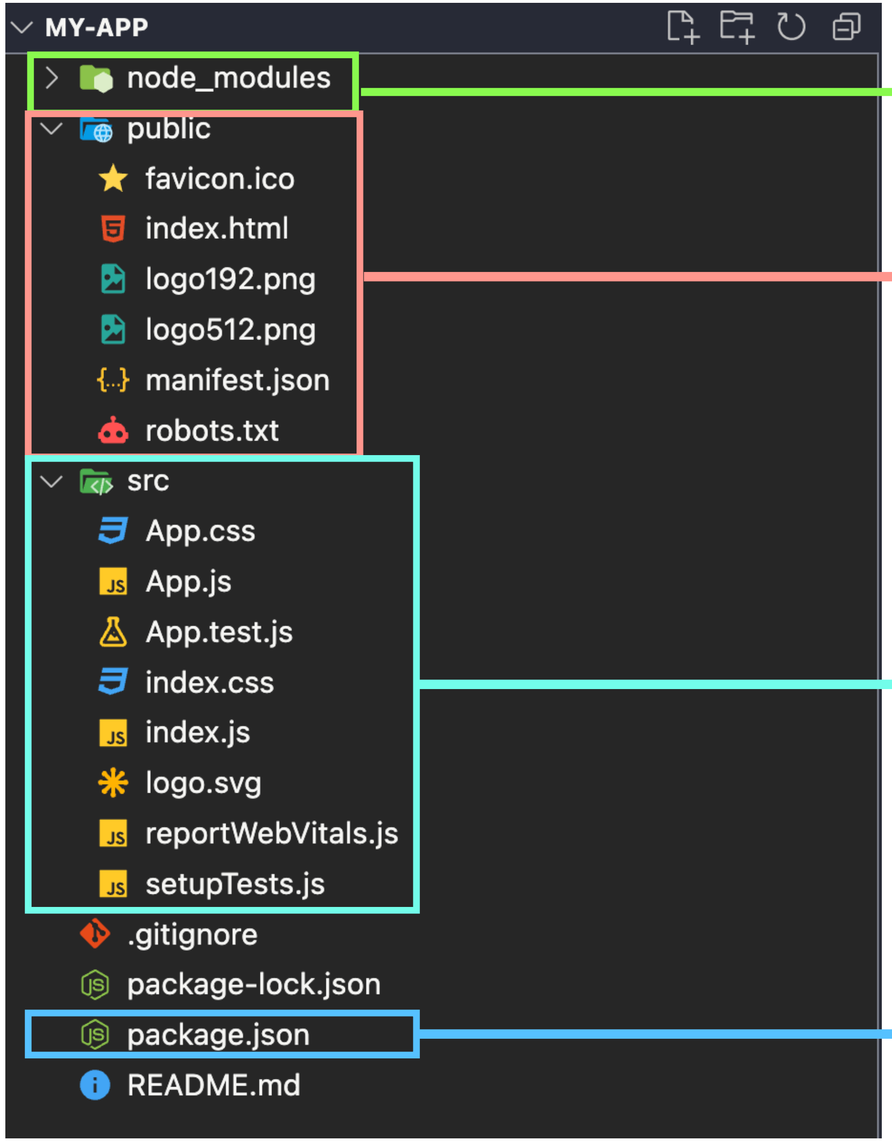
npm start // starts a development server

npm run build // build the project for production

CRA Project Structure



CRA Project Structure



All the project dependency packages and files

Public Directory

Source Directory

Where we write our code

package file

React Rendering

How create-react-app renders elements on the DOM?

src/index.js

```
import ReactDOM from 'react-dom/client';
```

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

public/index.html

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

package.json

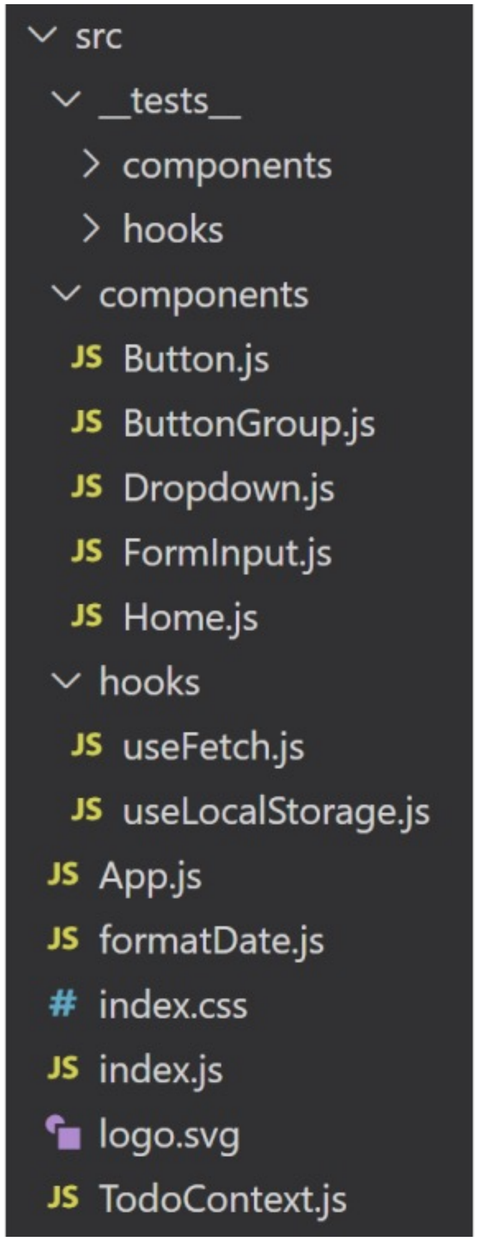
- package.json is the core of all Node project
- Contains all the information about the project - including configuration, version tracking, scripts, dependency packages

{} package.json > ...

```
1  {
2    "name": "myfirstreactapp",
3    "version": "0.1.0",
4    "private": true,
5    "dependencies": {
6      "@testing-library/jest-dom": "^4.2.4",
7      "@testing-library/react": "^9.5.0",
8      "@testing-library/user-event": "^7.2.1",
9      "react": "^16.13.1",
10     "react-dom": "^16.13.1",
11     "react-scripts": "3.4.3"
12   },
13   "scripts": {
14     "start": "react-scripts start",
15     "build": "react-scripts build",
16     "test": "react-scripts test",
17     "eject": "react-scripts eject"
18   },
19   "eslintConfig": {
20     "extends": "react-app"
21   },
22   "browserslist": {
23     "production": [
24       ">0.2%",
25       "not dead",
26       "not op_mini all"
27     ],
28     "development": [
29       "last 1 chrome version",
30       "last 1 firefox version",
31       "last 1 safari version"
32     ]
33   }
34 }
35
```

File Structuring

- All our code can be split into multiple files for each component, hooks, and other utility functions
- They can be nested into their own directories within the src folder



Importing/Exporting Components and Functions

- React (and most other common UI libraries and frameworks) uses JavaScript modules convention to export and import modules
- All component functions <u>must be exported</u> so we can import them where we need
- Components can be exported by adding the export keyword in front of the function

Importing/Exporting Components and Functions

Exporting

```
1  export function MyComponent() {
2    const name = "World";
3
4    return <p>Hello {name}</p>;
5  }
6
```

Importing

```
import { MyComponent } from './MyComponent';
```

Named Exports vs Default Exports

Export Statements:

```
export default function Button() {} // default export
export function Button() {} // named export
```

Import Statements:

```
import Button from './button.js'; // default export
import { Button } from './button.js'; // Named export
```

Important npm commands

npm init // initialise a node app

npm install package-name // install a package in the directory

npm uninstall package-name // uninstall the package directory

Global flag with -g

npm install -g package-name // install the package globally

npm uninstall -g package-name // install the package globally

Useful VSCode Extensions for React

- ES7+ React/Redux/React-Native snippets
- Prettier
- Path Intellisense
- ESLint

Exercise

- Setup a ReactJS Project
  - Create a **Card** Component that accepts a title, image path, and description as props
  - Use the Card Component on your App.jsx with varying prop data

React Hooks

- Hooks are utility functions that are part of React
- Hooks exposes APIs to local state, lifecycle events, reference to JSX elements etc
- Hooks follow a naming pattern. All hooks are prefixed with `use{hook_name}`. E.g. useState, useRef, useEffect etc

Local State

With useState

- State is an object to store values in the component
- Components (and all children components) are automatically re-rendered whenever the state changes
- In React, we can store state data with useState hook

Local State

With useState

```
function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Local State

With useState

```
function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Initial value

Getter

Get the state value

Setter

Set/update the state value

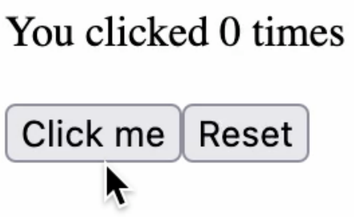
Updating state

Using the Setter function on event listener

Local State

With useState

```
export default function App() {
  const [count, setCount] = useState(0)
  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
      <button onClick={() => setCount(0)}>Reset</button>
    </div>
  );
}
```



Updating Previous Value

With useState

```
const [counter, setCounter] = useState(0);

const incrementHandler = () => {
  setCounter(counter + 1);
}
```

```
const [counter, setCounter] = useState(0);

const incrementHandler = () => {
  setCounter((prevValue) => prevValue + 1);
}
```

State Setter function accepts a value <u>or</u> a function that returns a value

Conditional Rendering

Showing/hiding content conditionally using State value or Props

```
function LogButton({isLoggedIn}) {
  if (isLoggedIn == true) {
    return (
        <button>Logout</button>
    )
  } else {
    return (
        <button>Login</button>
    )
  }
}
```

Conditional Rendering

Showing/hiding content conditionally using State value or Props

```
function LogButton({isLoggedIn}) {
  if (isLoggedIn == true) {
    return (
        <button>Logout</button>
    )
  } else {
    return (
        <button>Login</button>
    )
  }
}
```

isLoggedIn prop passed into LogButton component

If user is logged in, show “Logout”,
Else, show “Login”

Conditional Rendering

New Syntax

```
function LogButton({isLoggedIn}) {
    return (
        <button>{isLoggedIn ? "Logout" : "Login"}</button>
    )
}
```

Conditional Rendering

JavaScript Ternary Operator

```
name = (inUpperCase === true) ? name.toUpperCase() : name.toLowerCase();
```

1 CONDITIONAL TO CHECK

2 EXECUTED IF THE CONDITION IS TRUE

3 EXECUTED IF THE CONDITION IS FALSE

Exercise

- Setup a ReactJS Project
  - Create A Counter App with
  - Increment, Decrement and Increment by `n` buttons