HyperionDev

Full-Stack Web Development

HyperionDev

An introduction to React

Full Stack Web Development Session Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: **Questions**

Full Stack Web Development Session Housekeeping cont.

- For all **non-academic questions**, please submit a query: **www.hyperiondev.com/support**
- Report a **safeguarding** incident: **www.hyperiondev.com/safeguardreporting**
- We would love your **feedback** on lectures: **Feedback on Lectures**

Objective
s

- Explain the purpose and function of components in React.
- Create and render their first React Component
- Write and render simple JSX elements.
- Implement a good folder structure and best practices meant for React

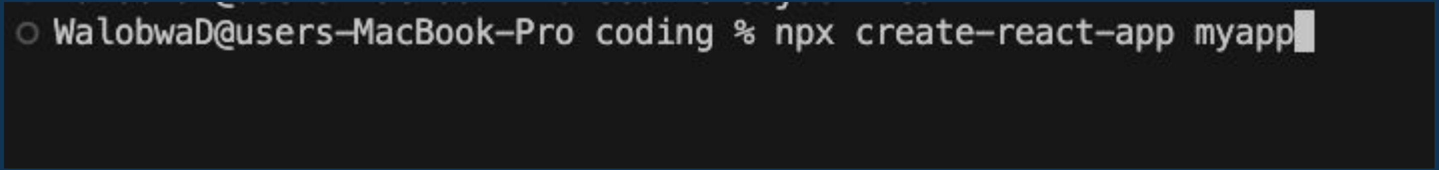
Introduction to React

What is React?

- ReactJS is a declarative, efficient, and flexible JavaScript library for building reusable UI components.
- It is an open-source, component-based front end library responsible only for the view layer of the application.
- A ReactJS application is made up of multiple components, each component responsible for outputting a small, reusable piece of code.
- The components are the heart of all React applications. These Components can be nested with other components to allow complex applications to be built of simple building blocks.

Setting up your first React app

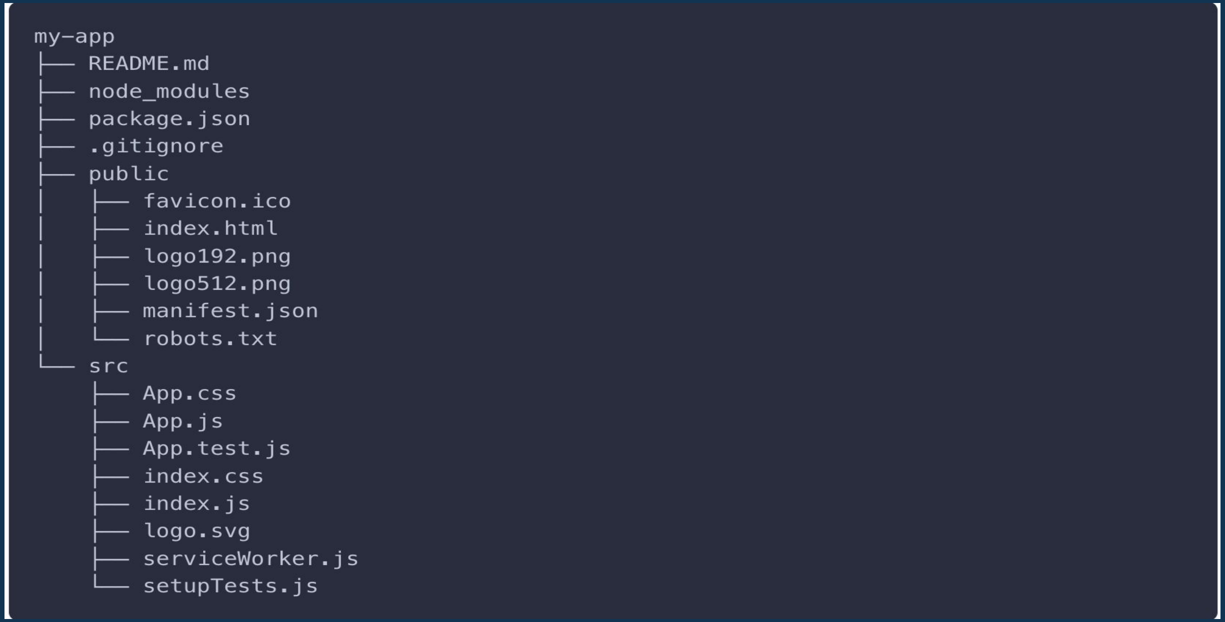
- **Requirements**
  - Nodejs v20 and above installed
  - VS Code as your editor
  - Brower(chrome/edge/firefox/safari)
- **Command**
  - In your terminal (in a preferred folder/desktop) insert the command npx create-react-app myapp
  - In this case (myapp) will be the name of the application.



```
○ WalobwaD@users-MacBook-Pro coding % npx create-react-app myapp
```

React Folder structure

❖ After initialization, you will have a folder structure similar to this



```
my-app
├── README.md
├── node_modules
├── package.json
├── .gitignore
├── public
│   ├── favicon.ico
│   ├── index.html
│   ├── logo192.png
│   ├── logo512.png
│   ├── manifest.json
│   └── robots.txt
└── src
    ├── App.css
    ├── App.js
    ├── App.test.js
    ├── index.css
    ├── index.js
    ├── logo.svg
    ├── serviceWorker.js
    └── setupTests.js
```

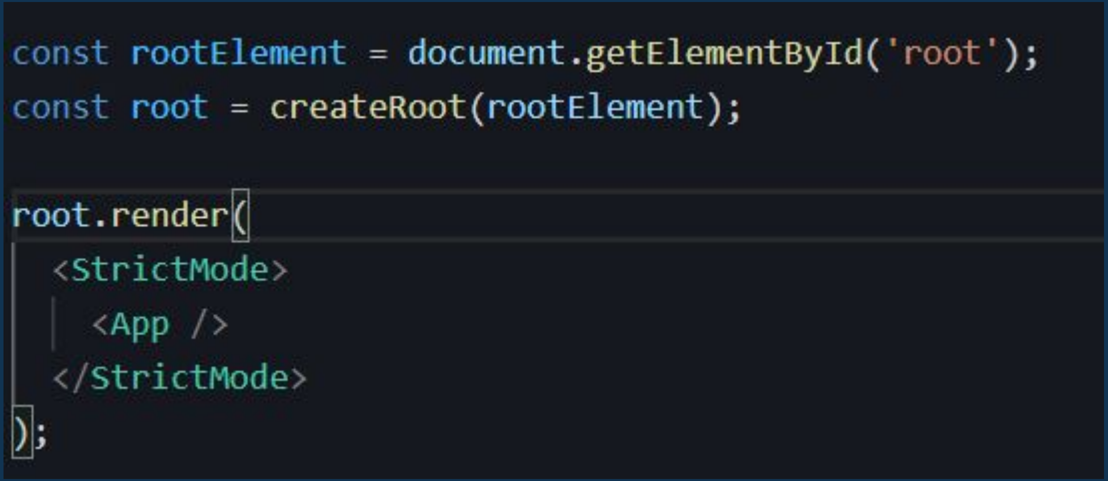
Running your React application

- Once your React app is created, you'd want to see it go live(run it on the browser). We use npm start as the command for running the app and a port will open for us to see the application's results.
- Commands
  - cd myapp
  - npm start

Rendering in React

- React renders HTML to the web page by using a function called **createRoot()** and its method **render()**.
- The **createRoot()** function takes one argument, an HTML element. The purpose of the function is to define the HTML element where a React component should be displayed.
- The **render()** method is then called to define the React component that should be rendered.

Rendering in React



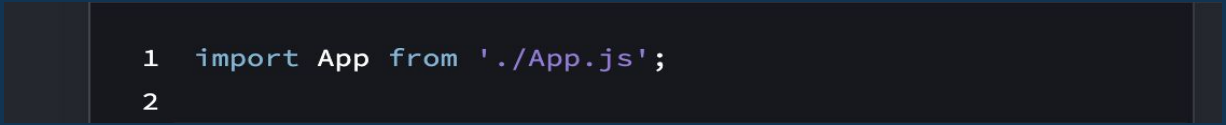
```
const rootElement = document.getElementById('root');
const root = createRoot(rootElement);

root.render(
  <StrictMode>
    <App />
  </StrictMode>
);
```

React Components

- ❖ A Component is considered as the core building blocks of a React application.
- ❖ It makes the task of building UIs much easier.
- ❖ Each component exists in the same space, but they work independently from one another and merge all in a parent component, which will be the final UI of your application.
- ❖ All React components have their own structure, methods as well as APIs. They can be reusable as per your need.

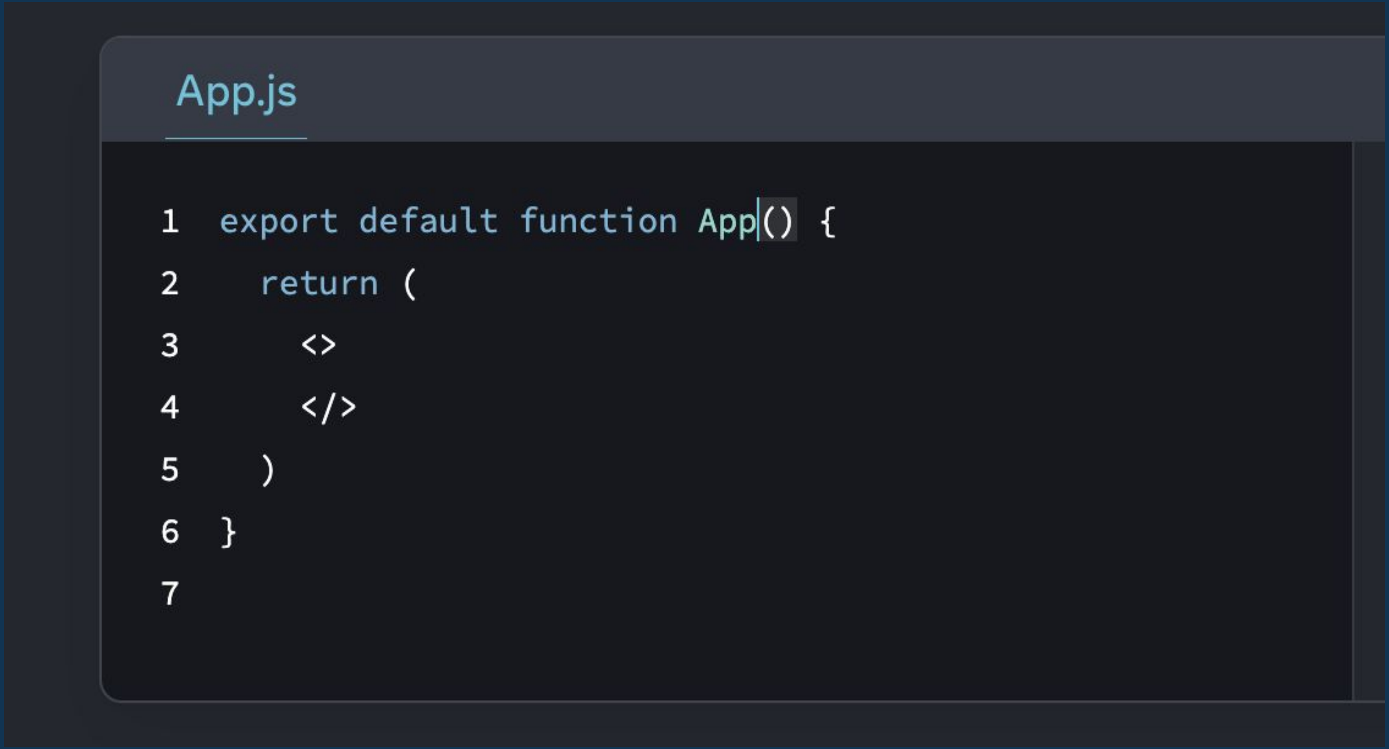
Functional Components

- ❖ A functional component is a simple Javascript function that takes in props as arguments and return JSX.
- ❖ A functional component should be exported and rendered on the main ui for it to display on the browser. We use the keyword export.
- ❖ For importing components, you do so in the parent component using the keyword import



```
import App from './App.js';

```

Functional Components


App.js

```
export default function App() {
  return (
    <>
    </>
  )
}

```

React JSX

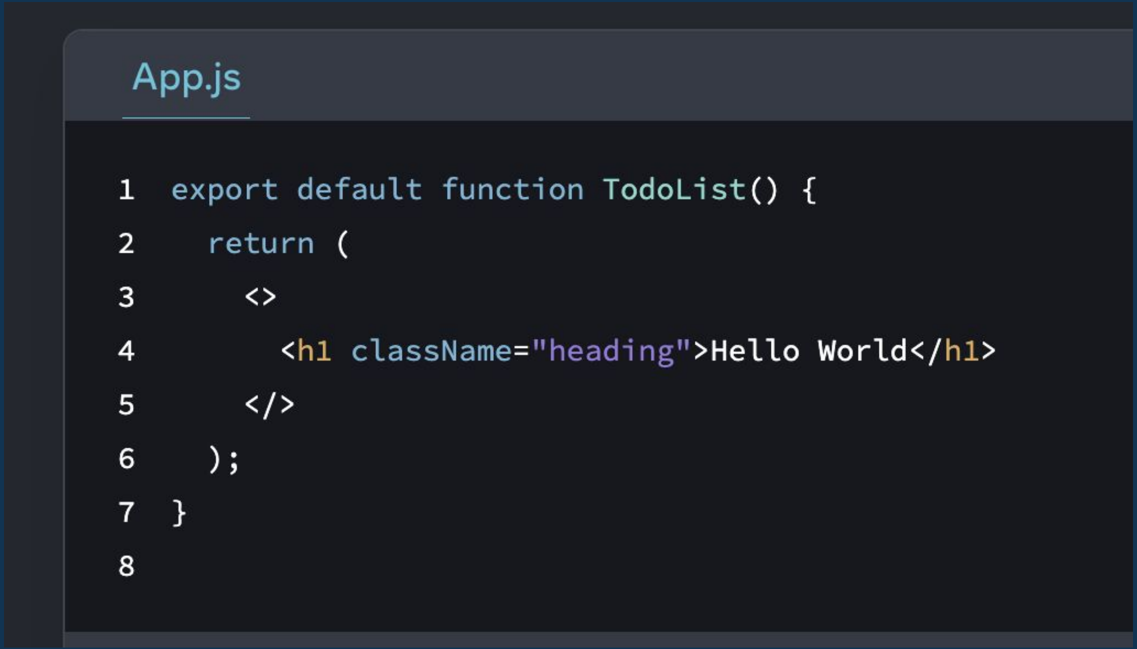
- ❖ JSX(JavaScript Extension), is a React extension which allows writing JavaScript code that looks like HTML.
- ❖ In other words, JSX is an HTML-like syntax used by React that extends ECMAScript so that HTML-like syntax can co-exist with JavaScript/React code.
- ❖ JSX allows you to write HTML/XML-like structures (e.g., DOM-like tree structures) in the same file where you write JavaScript code, then preprocessor will transform these expressions into actual JavaScript code.

Why use JSX?

- ❖ It is faster than regular JavaScript because it performs optimisation while translating the code to JavaScript.
- ❖ Instead of separating technologies by putting markup and logic in separate files, React uses components that contain both.
- ❖ It is type-safe, and most of the errors can be found at compilation time.
- ❖ It makes easier to create templates.

JSX Rules

- Always return a single root element
- Always ensure all tags are closed
- Use camelCase in most of the attributes naming/variables



App.js

```
export default function TodoList() {
  return (
    <>
      <h1 className="heading">Hello World</h1>
    </>
  );
}

```

Hyperiondev

Questions and Answers

HyperionDev

Thank You for attending!