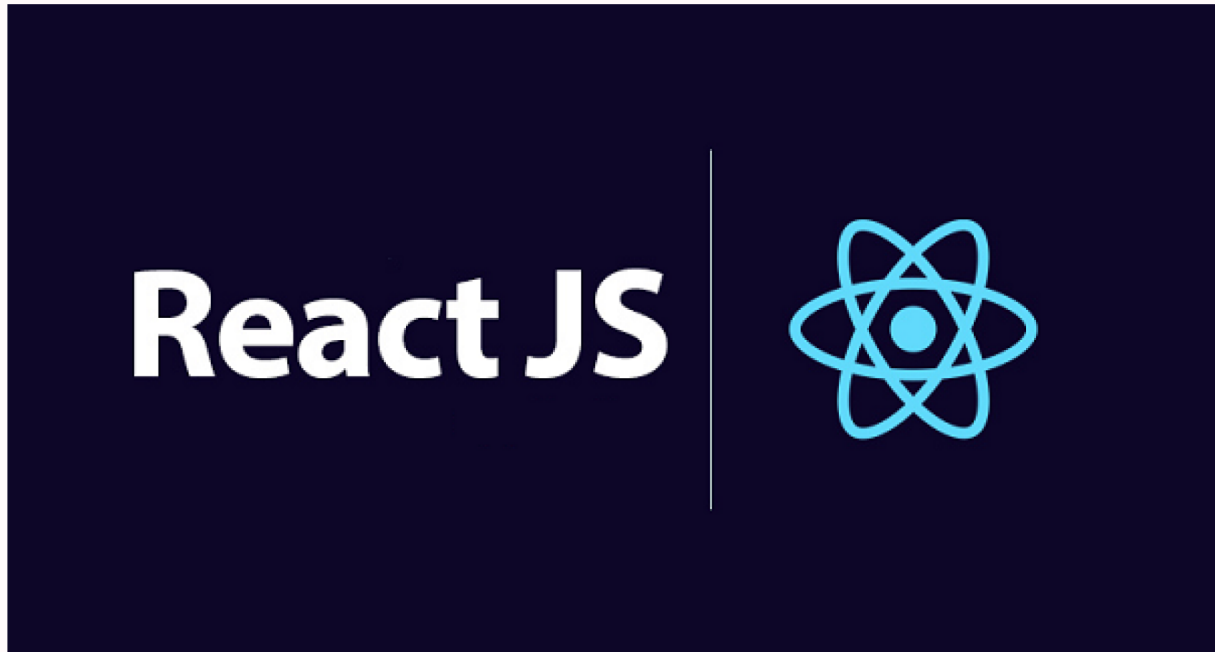


React Manual



www.anchorsoftacademy.com

Module 1. Getting Started

- Introduction
- What is React?
- Development Environment Setup

Module 2. React Basics and JSX

- Create Your First React App
- Initializing a React App
- JSX Fundamentals and Workflow
- Working with List in JSX
- Conditionals in JSX

Module 3. Components, Props and State

- Creating Components

- Props
- Props Defaults and Props Types
- Components Styles
- State and useState hook
- The Student Attendance Rating App
- The StudentItem, StudentList components
- Card component
- Event and prop drilling
- The StudentSummary component

Module 4. Working with Forms

- Form input state
- Custom Button component
- Form validation
- The Rating component
- Add new rating event

Module 5. Routes and Links

- Creating routes
- Replacing anchor tag with the Link component
- NavLink and useParams hook
- Navigate and Nested Routes

Module 6. Managing State with the Context API

- Create a Context and a Provider
- Fetch State data with useContext hook
- Move App component event handlers to the Context API
- Edit handler with useEffect hook
- Deploy to Vercel

Module 7. Backend Communication - API & http Request

- Introduction to API and http requests
- Mock backend with JSON Server
- Postman and http requests
- Run Frontend and API with concurrently
- Set API base as Proxy on Package.json
- Implement Add, Delete and Update event handlers with Fetch API

Module 1. Getting Started with React

Introduction

React is the last technology will cover in the frontend development section of this course. Everything we have covered so far in HTML and CSS will be brought together and used in React development.

What is React?

React is a JavaScript library used for building user interfaces. It was created at Facebook by Jordan Walke in 2013, and is maintained as an open-source project. React's primary purpose is to simplify the process of developing interactive and dynamic web applications by breaking down the user interface into reusable components.

Key concepts in React:

- **Components:** Components are the building blocks of React applications. They encapsulate UI elements and their behavior, making it easier to manage and reuse code.
- **Props:** Props (short for properties) are a way to pass data from parent to child components. They make components dynamic by allowing them to receive and render different data.
- **State:** State is used to manage data that can change over time within a component. It enables components to respond to user interactions and re-render when necessary.
- **Virtual DOM:** React uses a virtual representation of the DOM to efficiently update the actual DOM, improving performance.
- **React Router:** React Router is a popular library for handling routing in React applications, enabling navigation between different views or pages.

Development Environment Setup

Before you can start building React applications, you need to set up your development environment. Here are the steps to get started:

1. **Node.js and npm Installation:** React development typically relies on Node.js and npm (Node Package Manager). You can download and install them from the official website: [Node.js](https://nodejs.org/).
2. **VS Code:** While you can use any code editor of your choice, we strongly recommend vs code. To get to this level of the program, you should already have vs code installed. You can download vs code here: <https://code.visualstudio.com/>.
3. **Git:** Every react project also automatically creates a git repo for you. So it is important to also have git installed: <https://git-scm.com/>.

Module 2. React Basics and JSX

2.1 Create Your First React App

To get started with React, you'll want to create your first React application. Follow these steps:

1. **Setup Node.js and npm:** Ensure you have Node.js and npm installed on your system. You can download them from the official website: [Node.js](https://nodejs.org/).

2. **Create a New React App:** One of the easiest ways to start a React project is by using the "Create React App" tool. Open your terminal and run the following command to create a new React app (replace "my-app" with your desired project name):

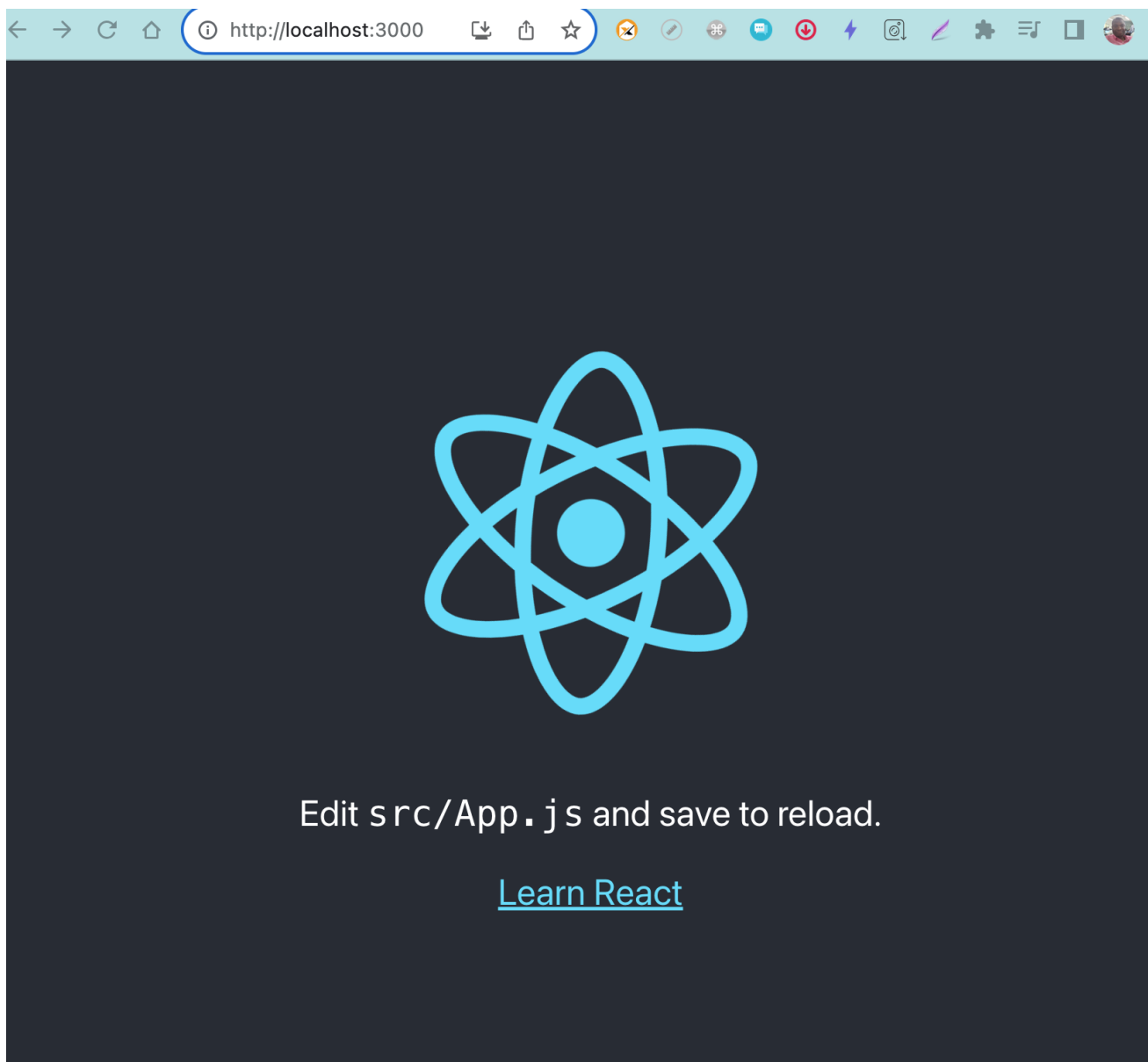
```
npx create-react-app my-app --use-npm
```

3. **Navigate to the Project Folder:** Once the app is created, navigate to the project folder using the command:

```
cd my-app
```

4. **Start the Development Server:** Start the development server by running:

```
npm start
```

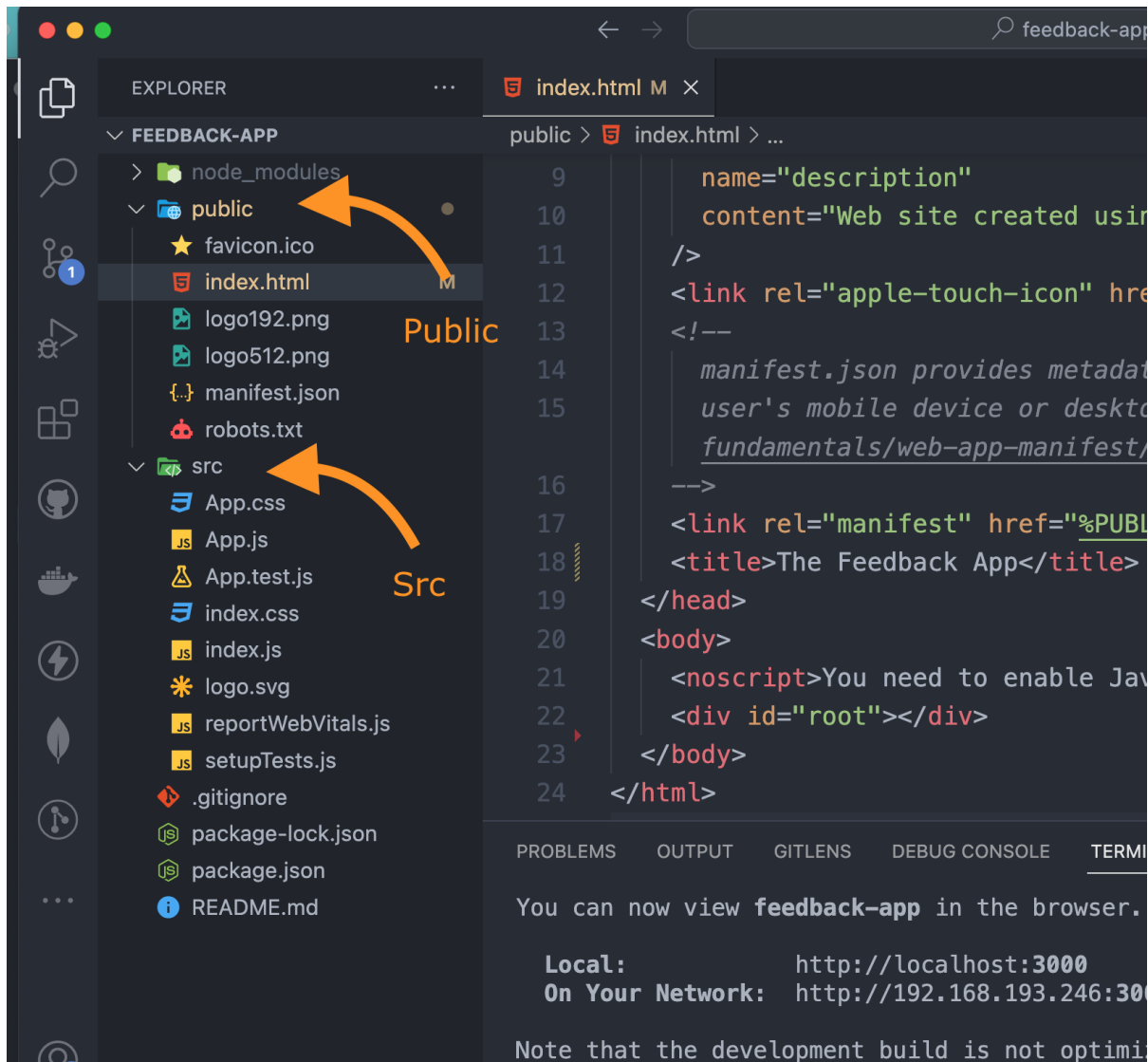


start and stop the web server

After creating the app, you can stop the server with `ctrl + c` and start it again with `npm start`.

5. **Open Your App:** Your React app will open in a web browser, and you'll see a basic React template. You're now ready to start working with React!

Folder structure



When you create a new React application using a tool like `create-react-app`, it generates an initial codebase and project structure to help you get started quickly as shown in the above shot. This is an overview of the typical project structure and some of the important files and directories:

- node_modules:** This directory contains all the third-party packages and dependencies your React app uses. You don't need to manage this directory manually; infact you should never tinker with it at all, npm handles package installation or removal for you.
- public:** This directory contains files that are publicly accessible, such as the HTML file that serves as the entry point for your app. Some important files and directories in the `public` directory include:
 - index.html:** The main HTML file that is loaded by the browser. It usually includes a `<div>` element with an `id` where your React app will be rendered.

- **favicon.ico**: The app's favicon icon.
- **manifest.json**: A JSON file that provides metadata about the web application.
- Other assets like images, fonts, or other static files can be placed in this directory.

C. **src**: This is your workbench! This is where you will do virtually all your work. This directory contains the source code for your React application. Some of the key files and directories within the **src** directory include:

- **index.js**: The entry point for your React application. It typically imports the necessary dependencies and renders your app into the DOM.
- **App.js**: The main component of your application. This is where you define the structure and behavior of your app.
- **index.css**: Global CSS styles for your application.
- **components/**: This is not created for you by default, you will add it yourself. This is where you can organize your React components. You can create additional subdirectories for better organization.
- **App.test.js** and other ***.test.js** files: These are unit test files for your components and can be found throughout your project. Though we are not going to cover testing in this course.

d. **package.json**: This is the file you will find in every node project. This file contains metadata about your project and its dependencies. It also includes scripts for common development tasks like starting the development server, building your app, or running tests. e. **package-lock.json**: This file locks the version of each dependency in your project to ensure consistency across different development environments.

f. **README.md**: A documentation file that provides information about your project, how to set it up, and how to use it.

g. **.gitignore**: You are definitely already familiar with this as covered in the second section of this course. The file specifies which files and directories should be ignored by git. It typically includes entries for the **node_modules** directory and build artifacts.

h. **public/favicon.ico**: The favicon icon for your app that appears in the browser tab.

i. **public/manifest.json**: A JSON file that provides metadata about your web application, such as its name and icons for home screen installation on mobile devices.

j. **public/service-worker.js**: A service worker file for Progressive Web App (PWA) functionality. It enables features like offline caching.

This is a high-level overview of the typical project structure and files you'll encounter when creating a React app using a tool like **create-react-app**. Keep in mind that the specific structure may vary depending on your own preferred component structure, the tools and configurations you use, or even subsequent updates of the react library.

Delete all the files in src folder

Trying to explain the code inside each file is not an effective way to learn react. A better way is to delete all the files and start creating them one after the other

2.2 Initializing React

Index.js, App.js and Index.css

1. Index.js: This is the first compulsory file in the generated react project. It is this `index.js` file that loads the `public/index.html`. It is also this file that loads the main react App component into the html through the `createRoot()` method.

src/index.js

```
import { createRoot } from "react-dom/client";

const root = createRoot(document.querySelector("#root"));

root.render(<h1>Hello world React!</h1>);
```

```
// React 17 Code
//=====
// import React from "react";
// import ReactDOM from "react-dom";
// import './index.css';
// import App from './App'

// ReactDOM.render(
//   //   <React.StrictMode>
//   //   <App />
//   //   </React.StrictMode>
//   ,document.getElementById('root'))

// React 18 Code
//=====
import { createRoot } from "react-dom/client";
import './index.css' <===== the main css
import App from './App'; <===== the root component

const root = createRoot(document.getElementById("root")); // createRoot
root.render(<App />);
```

2. App.js: This is the main component. It could be a function or class(old-school) based component. The react world has moved away from class-based component to function-based component. This `App.js` main/root component is imported into the `index.js` file:

src/App.js

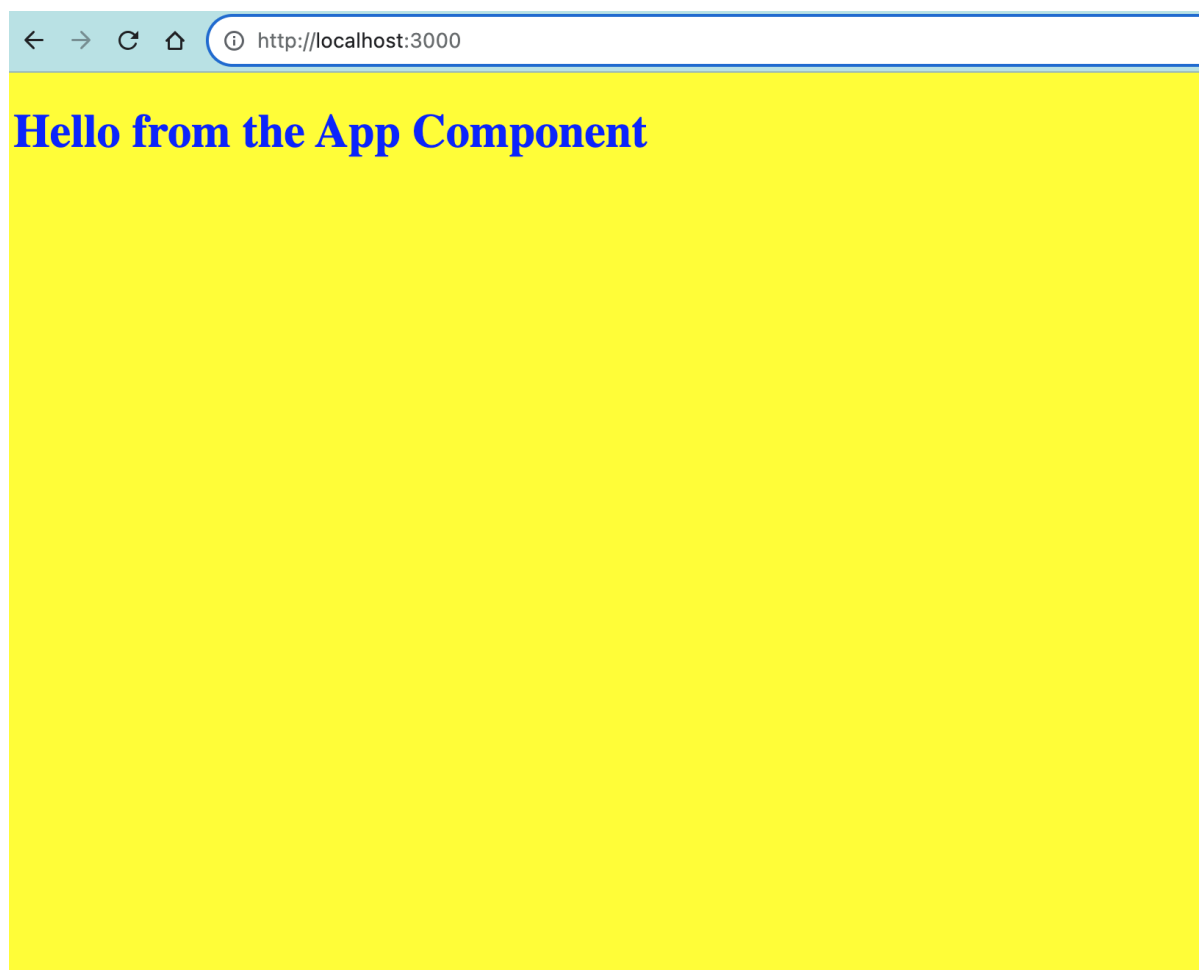
```
function App() {
  return (
    <div>
      <h1>Hello from the App Component</h1>
    </div>
  );
}
```

```
}  
  
export default App;
```

3. index.css(Styling): This is called the **global css file**. There are different ways to add css in a react project, one of them is this global css file. Apart from global style, you can also create inline style for a given component, which will be covered later in the course.

src/index.css

```
body {  
  background-color: yellow;  
}  
  
h1 {  
  color: blue;  
}
```



Strict Mode

When rendering the main component inside `index.js`, we also enforce what is called **strict mode**. This mode adds more checks to our react app, leading to a better code base quality:


```
import React from "react";
import { createRoot } from "react-dom/client";
import { App } from "./App";

const root = createRoot(document.querySelector("#root"));

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

You will not see any change in the output at the moment, but enforcing the strict mode has the benefits of preventing potential exceptions in your codebase.

Global css file

There are different ways to add css to element in a react app, we'll explore these as we go along. The easiest is to use global css. Create `index.css` inside the src folder and import it into the main `index.js` file:

~/index.css

```
body {
  background-color: yellow;
}

h1 {
  color: red;
}
```

~/index.js

```
import React from "react";
import { createRoot } from "react-dom/client";
import { App } from "./App";
import "./index.css"; <=====

const root = createRoot(document.querySelector("#root"));

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

2.3. JSX Fundamentals and Workflow

JSX (JavaScript XML) is a Javascript extension to html. It is a language you can describe as a javascript flavour of html.

It looks a lot like html, but it is actually javascript, and that is the language we use to build react components. Also note that jsx is no string. Don't make the mistake of putting a piece of jsx code into quotes.

Rules for writing JSX

JSX (JavaScript XML) in React follows a syntax that blends JavaScript with HTML-like elements to describe the structure of your user interface. Here are some key rules and examples for writing JSX:

1. Use Capitalized Component Names:

- JSX element names that start with a lowercase letter are assumed to be HTML tags, which causes errors. JSX element names that start with an uppercase letter are assumed to be custom React components.
- Example:

```
// Valid JSX for an HTML element
const element = <div>Hello, world!</div>;

// Valid JSX for a React component
const MyComponent = <MyCustomComponent />;
```

2. Self-Closing Tags:

- HTML-like self-closing tags should be closed with a trailing slash `/`, or you can simply close them without the slash.
- Example:

```
// Valid self-closing tags
const imgTag = ;
const inputTag = <input type="text" />;
```

3. JavaScript Expressions Inside Curly Braces:

- You can embed JavaScript expressions within JSX elements using curly braces `{}`.
- Example:

```
const greeting = "Hello, world!";
const element = <div>{greeting}</div>;
```

4. Single Root Element:

- JSX expressions must have a single root element. If you need to render multiple elements, wrap them in a parent element.
- Example:

```
// Valid JSX with a single root element
const element = (
  <div>
    <h1>Hello</h1>
    <p>World</p>
  </div>
);

// Invalid JSX without a single root element
const elements = <h1>Hello</h1><p>World</p>;
```

5. HTML Attributes Are Written in CamelCase:

- HTML attributes in JSX should be written using camelCase, not kebab-case. For example, `class` becomes `className`, and `for` becomes `htmlFor`.
- Example:

```
const myElement = <div className="my-class">Content</div>;
```

6. Comments:

- You can include comments in JSX using curly braces with `/* */` syntax.
- Example:

```
const element = (
  <div>
    {/* This is a comment */}
    <p>Some text</p>
  </div>
);
```

7. JSX in JavaScript Functions:

- JSX can be used in JavaScript functions. Typically, React components are defined as functions that return JSX.
- Example:

```
function MyComponent() {  
  return <div>Hello from MyComponent</div>;  
}
```

These are the fundamental rules for writing JSX in React. JSX is a powerful tool for building user interfaces, and understanding these rules is essential for creating React applications effectively.

2.4. Working with lists in JSX

To output dynamic content in react, we use the braces `{}`. You can also write other valid javascript codes within the braces. The sample below shows writing different expressions types with variables and an array declared within the component.

NB: Those variables and the array will usually come from the state and not declared directly inside the component. This is just to demonstrate the use of the braces:

```
function App() {  
  const title = "Student List";  
  const students = [  
    { id: 1, name: "aminat" },  
    { id: 2, name: "kunle" },  
    { id: 3, name: "zion" },  
    { id: 4, name: "ife" },  
    { id: 5, name: "james" },  
    { id: 6, name: "ayoade" },  
    { id: 7, name: "nelson" },  
    { id: 8, name: "david" },  
    { id: 9, name: "gbenga" },  
    { id: 10, name: "treasure" },  
    { id: 11, name: "dominion" },  
    { id: 12, name: "hammed" },  
  ];  
  
  return (  
    <div>  
      <h1 className="dojo">{title.toUpperCase()}</h1>  
      <ul>  
        {students.map((astudent, index) => (  
          <li key={index}>  
            {astudent.name}, ID: {astudent.id}  
          </li>  
        ))}  
      </ul>  
    </div>  
  );  
}  
  
export default App;
```

2.5. Conditionals

There are different ways to use conditional inside a react component.

Different JSX for Different Conditions

Lets say you want to return specific jsx based on a condition. You might truncate the main return statement midway and return a different jsx. This is useful for situation where you are fetching data from an external resource:

```
function App() {
  const title = "Cool product";
  const body = "This is a product to enhance your efficiency";
  const students = [
    { id: 1, name: "aminat" },
    { id: 2, name: "kunle" },
    { id: 3, name: "zion" },
    { id: 4, name: "ife" },
    { id: 5, name: "james" },
    { id: 6, name: "ayoade" },
    { id: 7, name: "nelson" },
    { id: 8, name: "david" },
    { id: 9, name: "gbenga" },
    { id: 10, name: "treasure" },
    { id: 11, name: "dominion" },
    { id: 12, name: "hammed" },
  ];

  const loading = true;

  if (loading) return <h2>content loading ...</h2>; <=====

  return (
    <div>
      <h1 className="dojo">{title.toUpperCase()}</h1>
      <p>{body}</p>
      <h2>{23 + 90}</h2>
      <h3>Some Students</h3>
      <ul>
        {students.map((astudent, index) => (
          <li key={index}>
            {astudent.name}, ID: {astudent.id}
          </li>
        ))}
      </ul>
    </div>
  );
}
```

```
export default App;
```



content loading ...

The main jsx truncate midway

Different portion of a single jsx

For this, you will usually use a ternary operator and put different jsx into different variables to keep it neat:

src/App.jsx

```
function App() {
  const title = "Cool product";
  const body = "This is a product to enhance your efficiency";
  const students = [
    { id: 1, name: "aminat" },
    { id: 2, name: "kunle" },
    { id: 3, name: "zion" },
    { id: 4, name: "ife" },
    { id: 5, name: "james" },
    { id: 6, name: "ayoade" },
    { id: 7, name: "nelson" },
    { id: 8, name: "david" },
    { id: 9, name: "gbenga" },
    { id: 10, name: "treasure" },
    { id: 11, name: "dominion" },
    { id: 12, name: "hammed" },
  ];
  const loading = false;
  const showStudents = true;

  const studentBlock = ( <=====
  <>
```

```

    <h3>Some Students</h3>
    <ul>
      {students.map((astudent, index) => (
        <li key={index}>
          {astudent.name}, ID: {astudent.id}
        </li>
      ))}
    </ul>
  </>
);

if (loading) return <h2>content loading ...</h2>;

return (
  <div>
    <h1 className="dojo">{title.toUpperCase()}</h1>
    <p>{body}</p>
    <h2>{23 + 90}</h2>

    {showStudents ? studentBlock : null} <=====
  </div>
);
}

export default App;

```

Module 3. Components, Props and State

As mentioned repeatedly already, React is mainly used for building user interfaces, and at its core, it revolves around the concepts of components, props and state. These concepts are fundamental to understanding how React applications are structured and how data flows through them.

We will also talk about props types, props defaults, and styling components with inline styles.

3.1. Components

- **What are Components?**

- Components are the building blocks of React applications. They represent reusable, self-contained pieces of the user interface.
- Components can be functional (stateless) or class-based (stateful), depending on whether they manage internal state. We longer work with class components, we will conver function based component in this course.

- **Creating Components:**

- Functional components are JavaScript functions that return JSX elements.
- Example:

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}
```

3.2 Props

- **What are Props?**

- Props (short for "properties") are a way to pass data from a parent component to its child components.
- They are read-only and help make your components dynamic and reusable by allowing you to customize their behavior and appearance.

- **Passing Props:**

- To pass props to a child component, you simply include them as attributes when rendering the child component.

```
<ChildComponent prop1={value1} prop2={value2} />
```

- In the child component, you can access the props via the `props` object. Below is an example of a child component receiving props from the root `App.js` component:

`App.js` component:

```
// App.jsx  
  
import React from "react";  
import ChildComponent from "./ChildComponent";  
  
const App = () => {  
  const prop1Value = "Hello";  
  const prop2Value = "World";  
  
  return (  
    <div>  
      <h1>Parent App Component</h1>  
  
      {  
        /* Using ChildComponent with props */  
        <ChildComponent prop1={prop1Value} prop2={prop2Value} />  
      }  
    </div>  
  );  
};  
  
export default App;
```


And this is the Child component extracting the props:

```
// ChildComponent.jsx

import React from "react";

const ChildComponent = (props) => {
  // Without destructuring
  const prop1WithoutDestructuring = props.prop1;
  const prop2WithoutDestructuring = props.prop2;

  // With destructuring
  const { prop1, prop2 } = props;

  return (
    <div>
      <h2>Child Component</h2>

      { /* Without destructuring */ }
      <p>Prop1 without destructuring: {prop1WithoutDestructuring}</p>
      <p>Prop2 without destructuring: {prop2WithoutDestructuring}</p>

      { /* With destructuring */ }
      <p>Prop1 with destructuring: {prop1}</p>
      <p>Prop2 with destructuring: {prop2}</p>
    </div>
  );
};

export default ChildComponent;
```

3.3 Prop Types and Props Default

- **What are Prop Types?**
 - Prop types are a way to specify the expected data types of props passed to a component.
 - They provide type checking and help catch bugs and issues related to incorrect prop data.
- **Using Prop Types:**
 - To use prop types, you import the `prop-types` library and define the prop types for a component.

```
import PropTypes from "prop-types";

function MyComponent({ prop1, prop2 }) {
  // ...
}

MyComponent.propTypes = {
```

```
prop1: PropTypes.string.isRequired, // String prop that is required
prop2: PropTypes.number, // Optional number prop
};
```

- In the example above, we specify that **prop1** must be a required string prop, and **prop2** is an optional number prop.

Default Props:

- **What are Default Props?**

- Default props allow you to set default values for props in case they are not provided by the parent component.

```
function MyComponent(props) {
  // Use props.prop1, which defaults to "Default Value" if not
  // provided
  // ...
}

MyComponent.defaultProps = {
  prop1: "Default Value",
};
```

Sample component using default props and prop types

```
import React from "react";
import PropTypes from "prop-types";

const MyComponent = ({ name, age, isStudent }) => {
  return (
    <div>
      <p>Name: {name}</p>
      <p>Age: {age}</p>
      <p>Is Student: {isStudent ? "Yes" : "No"}</p>
    </div>
  );
};

// Default props
MyComponent.defaultProps = {
  name: "John Doe",
  age: 25,
  isStudent: false,
};

// Prop types
```

```
MyComponent.propTypes = {
  name: PropTypes.string.isRequired,
  age: PropTypes.number.isRequired,
  isStudent: PropTypes.bool.isRequired,
};

export default MyComponent;
```

3.4 Component Styles - Inline

We have seen how to use the global `index.css` to style the component. Another way to apply css is using a javascript, this is called an `inline style`:

```
import React from "react";

const MyComponent = () => {
  const buttonStyle = {
    backgroundColor: "#4CAF50",
    color: "white",
    padding: "10px 15px",
    border: "none",
    borderRadius: "5px",
  };

  return (
    <div>
      <button style={buttonStyle}>Click me</button>
    </div>
  );
};

export default MyComponent;
```

3.5 State and useState hook

In React, state represents the dynamic data that a component can maintain and modify over time. State is essential for building interactive and responsive user interfaces.

There are two types of state data:

- component level state: This is a data that is specific to a given component and cannot be shared by multiple components.
- App level state: This is an app-wide application data. This is usually stored in the `App.jsx` root component or coming from a more dedicated state management component like the context API.

useState Hook:

The `useState` hook is a React hook that enables functional components to declare and manage state. It takes an initial state value and returns an array with two elements: the current state and a function to update that state.

Usage Example:

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

const Counter = () => {
  // Declare a state variable named 'count' with an initial value of 0
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
};
```

In this example, the `count` state variable is initialized to `0`, and the `setCount` function is used to update its value. When the "Increment" button is clicked, the `setCount` function is called with the new value, triggering a re-render with the updated state.

Key Concepts:

1. Immutable Updates:

- React state should be treated as immutable. Instead of modifying the state directly, the `setCount` function (or an event handler that wraps the `setCounter` function for your state variable) is used to provide a new state based on the current state.

2. Asynchronous Updates:

- State updates with `useState` are asynchronous. React batches state updates for performance reasons. Therefore, you should not rely on the immediate state value after calling the setter function.

3. Multiple State Variables:

- You can use the `useState` hook multiple times in a component to manage different pieces of state independently.

```
const [name, setName] = useState("John");
const [age, setAge] = useState(25);
```

3.6 The Student Attendance Rating App

Enter a Student and Rate

1

2

3

4

5

6

7

8

9

10

Send

3 Reviews

6.666666666666667 Average Rating

2

Femi Pedro

10

Folorunso Olumide

8

Now that the basics of react: jsx, component, props and state are covered, learning react further is better with a sample project to demonstrate the concepts.

We are going to build a **student attending rating** app. The app displays a list of student with their attendance rating. Each student card also has two buttons to edit and delete the student entry.

This is a fairly simple app, but it demonstrate a lot of the core concepts you need to master to be proficient in react.

Create the Student App

Navigate into your workspace folder and create the app:

- `cd ~/workspace`
- `create-react-app studentapp --use-npm`

This will take a couple of minutes. When it is done, strip out the generated files you don't need to only `App.js`, `index.js` and `index.css`. Change the `App.js` to `App.jsx` to have a better developer experience:

```
function App() {  
  return <h1>My Student App</h1>;  
}
```

```
}  
  
export default App;
```

← → ↻ 🏠 ⓘ http://localhost:3001

My Student App

Add a Header component

Let's add a header component to the app. This will be inside a **components** folder.

~/components/Header.jsx

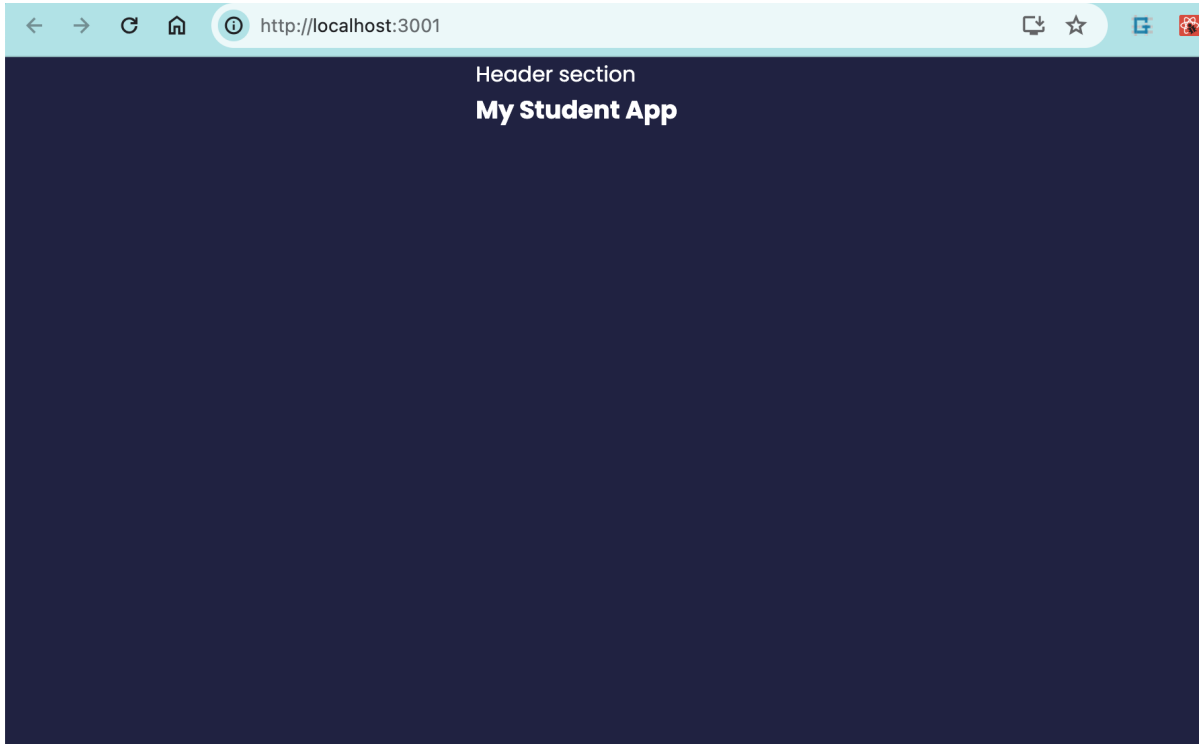
```
function Header() {  
  return <div className="container">Header section</div>;  
}  
  
export default Header;
```

~/App.jsx

```
import Header from "../components/Header";  
function App() {  
  return (  
    <>  
      <Header />  
      <div className="container">  
        <h3>My Student App</h3>  
      </div>  
    </>  
  );  
}
```

```
        </div>
      </>
    );
  }

  export default App;
```



StudentItem Component

The first component that will be created is the component that represents a single student entry (The name and the attendance rating).

STEPS

- Create `StudentItem.jsx` in the components folder
- Import it into the `App.jsx` main component

`components/StudentItem.jsx`

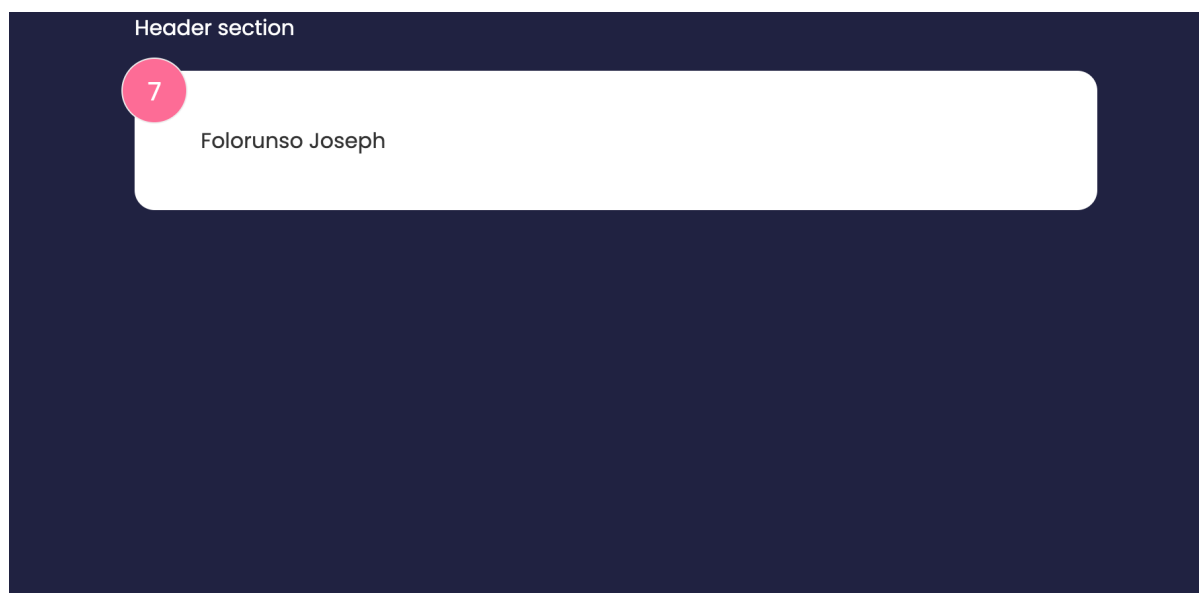
```
function StudentItem() {
  return (
    <div className="card">
      <div className="num-display">7</div>
      <div className="text-display">Folorunso Joseph</div>
    </div>
  );
}

export default StudentItem;
```

App.jsx

```
import Header from "../components/Header";
import StudentItem from "../components/StudentItem";
function App() {
  return (
    <>
      <Header />
      <div className="container">
        <StudentItem />
      </div>
    </>
  );
}

export default App;
```

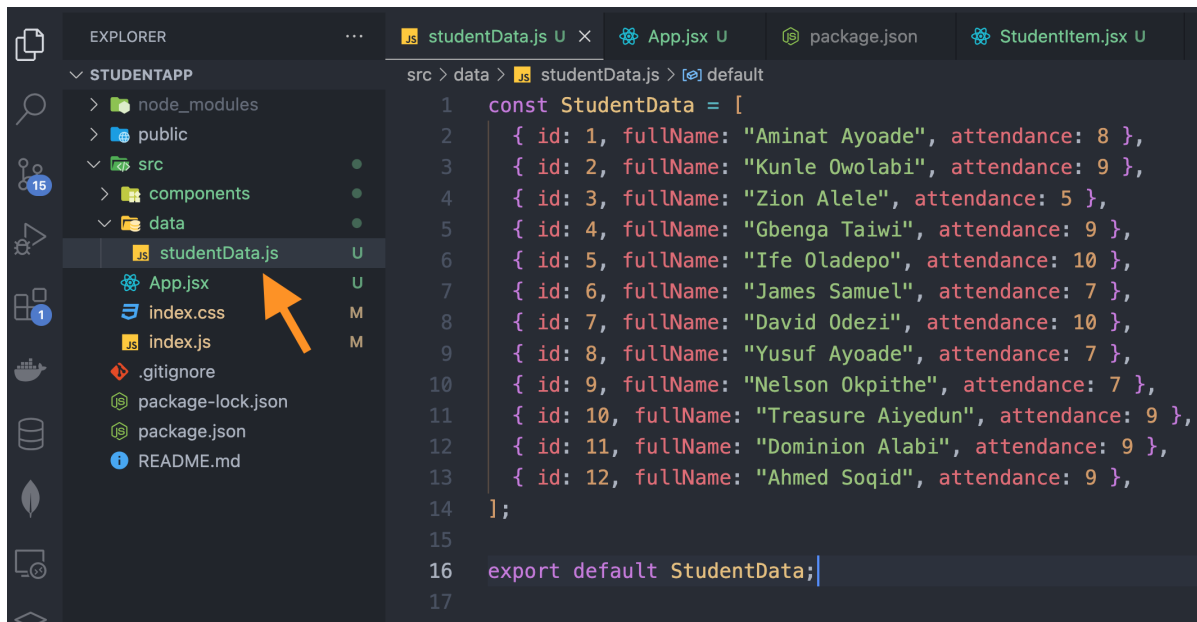


3.7 StudentItem, StudentList components

The StudentItem component above had no state data. Now let's create an App level state and load each StudentItem component with it.

StudentItem Component

Now let some state data to the App.jsx component. The data is an array student objects exported from javascript file:



data/StudentData.js

```
const StudentData = [  
  { id: 1, fullName: "Aminat Ayoade", attendance: 8 },  
  { id: 2, fullName: "Kunle Owolabi", attendance: 9 },  
  { id: 3, fullName: "Zion Alele", attendance: 5 },  
  { id: 4, fullName: "Gbenga Taiwo", attendance: 9 },  
  { id: 5, fullName: "Ife Oladepo", attendance: 10 },  
  { id: 6, fullName: "James Samuel", attendance: 7 },  
  { id: 7, fullName: "David Odezi", attendance: 10 },  
  { id: 8, fullName: "Yusuf Ayoade", attendance: 7 },  
  { id: 9, fullName: "Nelson Okpithe", attendance: 7 },  
  { id: 10, fullName: "Treasure Aiyedun", attendance: 9 },  
  { id: 11, fullName: "Dominion Alabi", attendance: 9 },  
  { id: 12, fullName: "Ahmed Soqid", attendance: 9 },  
];  
  
export default StudentData;
```

App.jsx

```
import { useState } from "react";  
import Header from "../components/Header";  
import StudentItem from "../components/StudentItem";  
import StudentData from "../data/studentData";  
  
function App() {  
  const [students, setStudents] = useState(StudentData);  
  return (  
    <>  
      <Header />  
    </>  
  );  
}
```

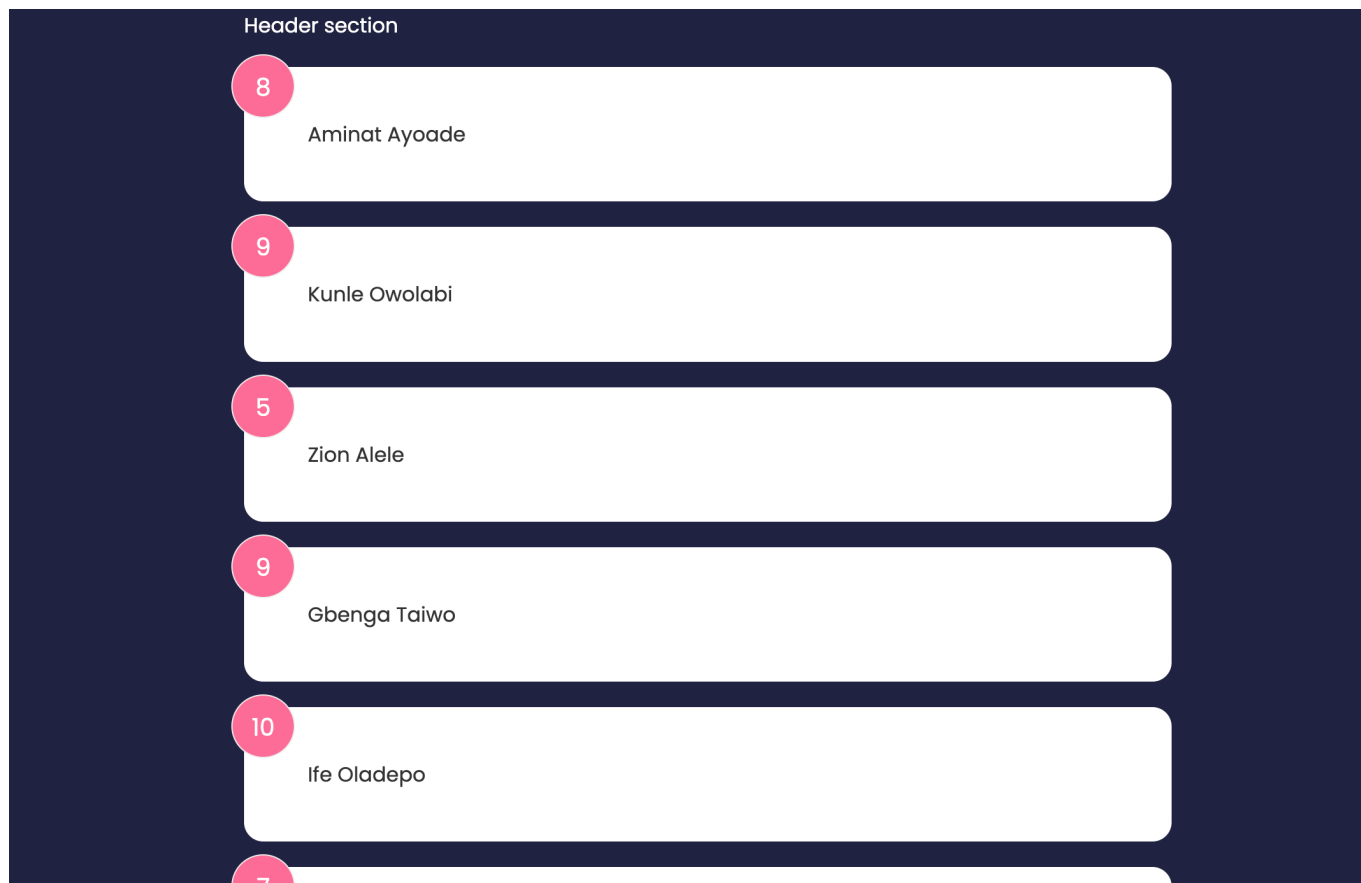
```
        <div className="container">
          <StudentItem students={students} />
        </div>
      </>
    );
  }

  export default App;
```

StudentItem.jsx

```
function StudentItem({ students }) {
  return (
    <>
      {students.map((student) => {
        return (
          <div className="card" key={student.id}>
            <div className="num-display">{student.attendance}</div>
            <div className="text-display">{student.fullName}</div>
          </div>
        );
      })}
    </>
  );
}

export default StudentItem;
```



StudentList Component

While we looped over the `studentData` inside the `StudentItem` component and it worked. A better way to structure the UI is to create to another component called `StudentList`, and then loop over the `StudentItem` component from inside the `StudentList` component.

App.jsx

```
import { useState } from "react";
import Header from "../components/Header";
import StudentList from "../components/StudentList";
import StudentData from "../data/studentData";

function App() {
  const [students, setStudents] = useState(StudentData);
  return (
    <>
      <Header />
      <div className="container">
        <StudentList students={students} />
      </div>
    </>
  );
}

export default App;
```

StudentList.jsx

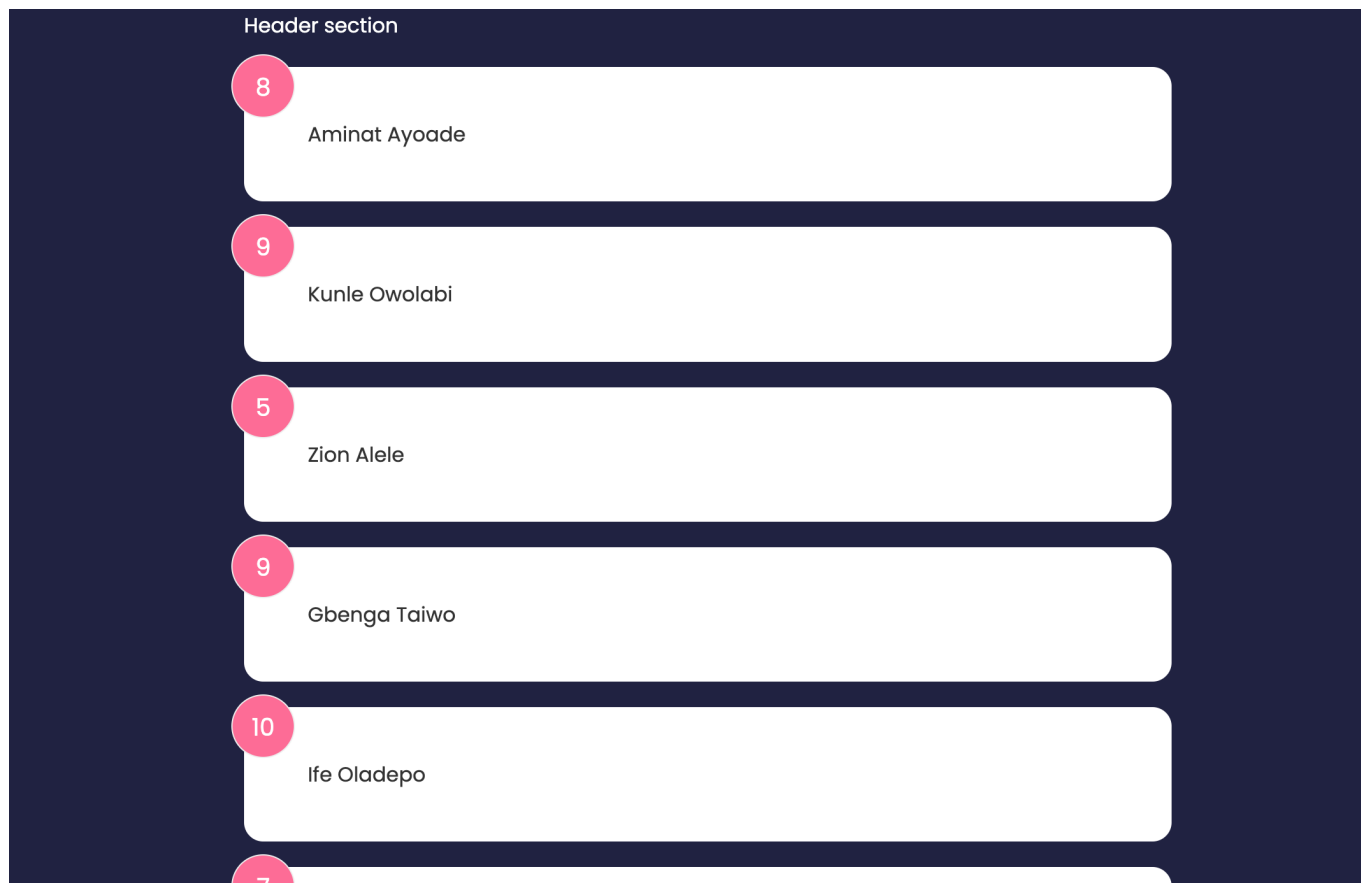
```
import StudentItem from "./StudentItem";
function StudentList({ students }) {
  return (
    <div>
      {students.map((student) => (
        <StudentItem key={student.id} studentData={student} />
      ))}
    </div>
  );
}

export default StudentList;
```

StudentItem.jsx

```
function StudentItem({ studentData }) {
  return (
    <>
      <div className="card" key={studentData.id}>
        <div className="num-display">{studentData.attendance}</div>
        <div className="text-display">{studentData.fullName}</div>
      </div>
    </>
  );
}

export default StudentItem;
```



3.8 Styled component - The Card.jsx

Below is the jsx code for each of the StudentItem component:

```
<div className="card" key="{studentData.id}">
  <div className="num-display">{studentData.attendance}</div>
  <div className="text-display">{studentData.fullName}</div>
</div>
```

The curved edged white background was created by the `.card` csss class. Sometimes you may want to convert this into a **wrapping component** instead of using a css class like so:

```
<Card>
  { some of other components or jsx codes }
</Card>
```

This is what we call Styled Component! The component is only responsible for styling and not data display.

Creating a Styled Component

- Create a folder named 'shared' inside the components folder.
- Add a Card.jsx component inside it.

This is placed inside a shared folder because it will be shared by multiple components.

components/shared/Card.jsx

```
function Card({ children }) {  
  return <div className="card">{children}</div>;  
}  
  
export default Card;
```

Now import and use it inside the `StudentItem.jsx`

```
import Card from "../shared/Card"; <=====  
function StudentItem({ studentData }) {  
  return (  
    <>  
      <Card> <=====  
        <div className="num-display">{studentData.attendance}</div>  
        <div className="text-display">{studentData.fullName}</div>  
      </Card>  
    </>  
  );  
}  
  
export default StudentItem;
```

Card Conditional Styling

We can also change the style of the Card component dynamically by passing a prop into it. In this case, the prop is named `reverse` and it is a boolean variable. When set to true, the class `.card.reverse` in the `index.css` file will be applied to the cards.

index.css

```
.card.reverse {  
  background-color: rgba(0, 0, 0, 0.4);  
  color: #fff;  
}
```

StudentItem.jsx

```
import Card from "../shared/Card";  
function StudentItem({ studentData }) {  
  return (  
    <>
```

```

      <Card reverse={true}> <=====
        <div className="num-display">{studentData.attendance}</div>
        <div className="text-display">{studentData.fullName}</div>
      </Card>
    </>
  );
}

export default StudentItem;

```

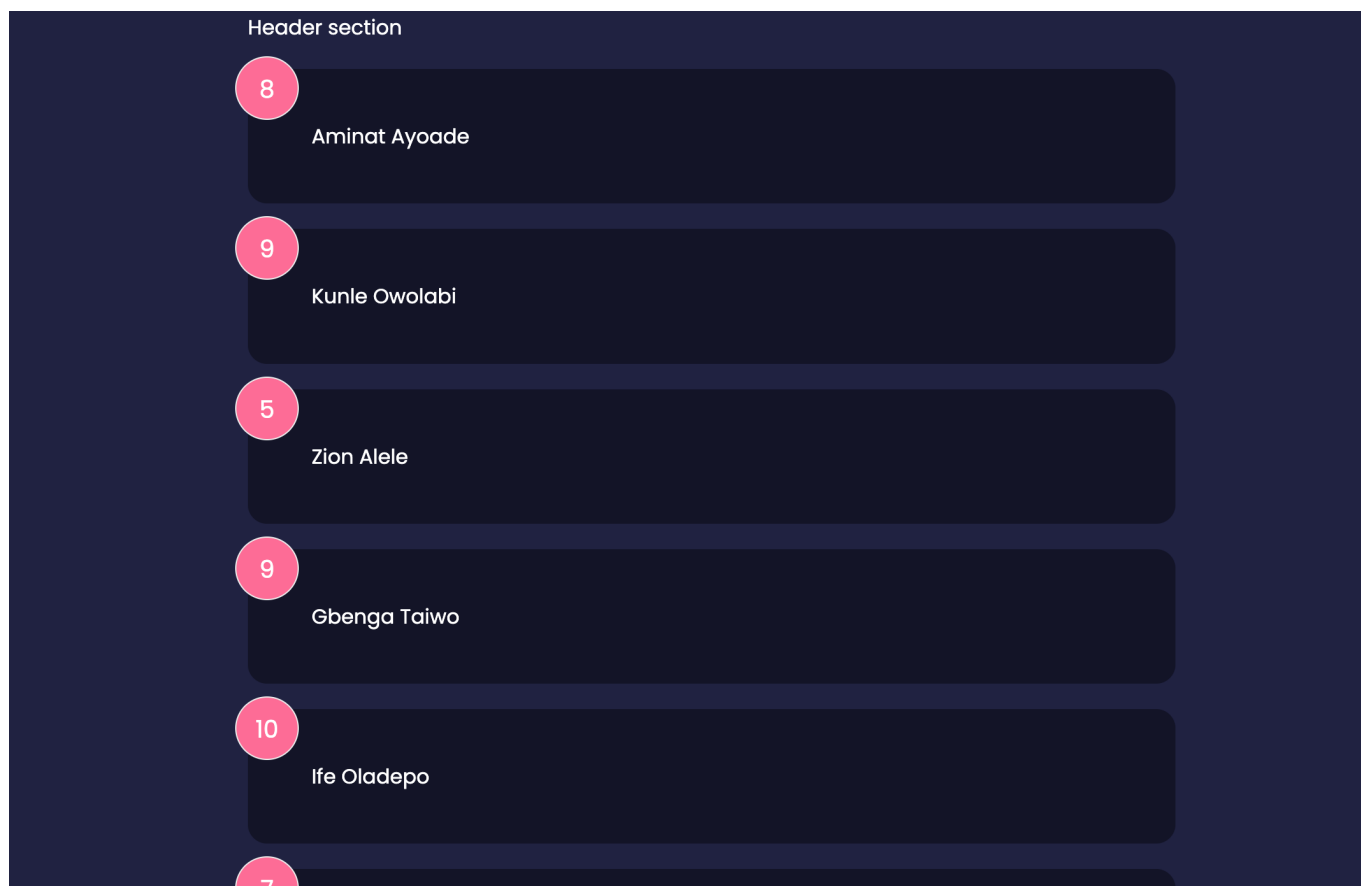
Card.jsx We used the ternary operator to set the css class for the div based on the value of the reverse props

```

function Card({ children, reverse }) {
  return <div className={reverse ? "card reverse" : "card"}>{children}</div>;
}

export default Card;

```



3.9 Prop Drilling and Events

What is Props Drilling?

Props drilling in React refers to the process of passing down props (properties) through multiple layers of nested components. When a component needs to pass data to its child components, it does so by passing props to them. If there are intermediary components that do not use the props but need to relay them to their children, this process is known as props drilling.

Here's an example to illustrate props drilling:

```
// GrandparentComponent.js
import React, { useState } from "react";
import ParentComponent from "../ParentComponent";

const GrandparentComponent = () => {
  const [data, setData] = useState("Hello from Grandparent");

  return (
    <div>
      <ParentComponent GPdata={data} />
    </div>
  );
};

export default GrandparentComponent;
```

```
// ParentComponent.js
import React from "react";
import ChildComponent from "../ChildComponent";

const ParentComponent = ({ GPdata }) => {
  return (
    <div>
      <ChildComponent Pdata={GPdata} />
    </div>
  );
};

export default ParentComponent;
```

```
// ChildComponent.js
import React from "react";

const ChildComponent = ({ Pdata }) => {
  return (
    <div>
      <p>{Pdata}</p>
    </div>
  );
};
```



```
};  
  
export default ChildComponent;
```

In this example, **GrandparentComponent** has some data and passes it to **ParentComponent** as a prop. **ParentComponent** doesn't use the data itself but passes it down to **ChildComponent**, which finally renders it.

The problem with props drilling is that as your component tree grows, passing props through all the intermediary components can become tedious and reduce the code's maintainability. To address this, you might consider using other state management solutions in React, such as Context API or state management libraries like Redux, to avoid excessive props drilling and make the code more maintainable. We'll cover Context API later in the course but it is important that you understand how to pass props.

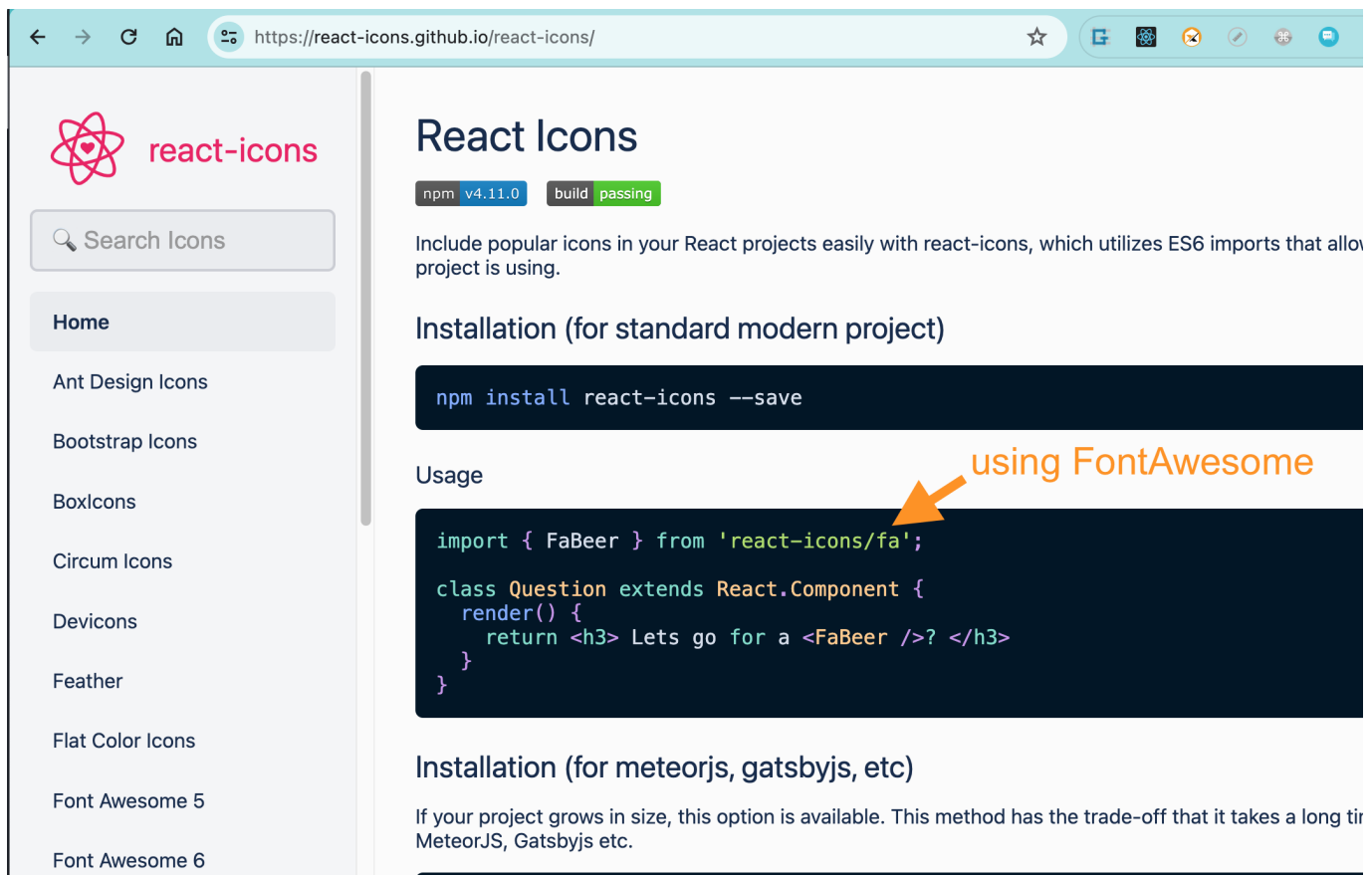
In our current application, let's implement **StudentItem** deletion via props drilling.

Delete button/Icon

Let add an icon to delete each **StudentItem**. There are different ways to add icons, we'll use a library called **react-icons**.

Install React-Icons

```
> npm i react-icons
```



The screenshot shows the React Icons website. The left sidebar contains a search bar and a list of icon sets: Ant Design Icons, Bootstrap Icons, BoxIcons, Circum Icons, Devicons, Feather, Flat Color Icons, Font Awesome 5, and Font Awesome 6. The main content area is titled "React Icons" and shows the npm version (v4.11.0) and build status (passing). It includes a description, installation instructions for standard modern projects, and usage examples. An orange arrow points to the 'fa' import in the usage example, with the text "using FontAwesome" next to it.

React Icons

npm v4.11.0 build passing

Include popular icons in your React projects easily with react-icons, which utilizes ES6 imports that allow your project is using.

Installation (for standard modern project)

```
npm install react-icons --save
```

Usage

```
import { FaBeer } from 'react-icons/fa';  
  
class Question extends React.Component {  
  render() {  
    return <h3> Lets go for a <FaBeer />? </h3>  
  }  
}
```

Installation (for meteorjs, gatsbyjs, etc)

If your project grows in size, this option is available. This method has the trade-off that it takes a long time to build MeteorJS, Gatsbyjs etc.

Lets now import this into the StudentItem component, and add two font awesome icons for delete and edit:

```
import { FaTimesCircle, FaRegEdit } from "react-icons/fa";
import Card from "../shared/Card";

function StudentItem({ studentData }) {
  return (
    <>
      <Card>
        <div className="num-display">{studentData.attendance}</div>
        <div className="text-display">{studentData.fullName}</div>
        <button className="close"> <=====
          <FaTimesCircle color="red" size={"2rem"} />
        </button>
        <button className="edit"> <=====
          <FaRegEdit color="orange" size={"2rem"} />
        </button>
      </Card>
    </>
  );
}

export default StudentItem;
```



Event

Events are various users' interactions that can be captured in our applications. These include click, mouseover, focus etc. The functions we execute in response to events are called **Event Handlers**. In react, there two ways to call event handlers:

1. By reference: This is usually when you are not passing any parameter to this event handler

```
import { FaTimesCircle, FaRegEdit } from "react-icons/fa";
import Card from "../shared/Card";
```

```
function StudentItem({ studentData }) {
  const myClickHandler = () => {
    console.log("Hello event");
  };

  return (
    <>
      <Card>
        <div className="num-display">{studentData.attendance}</div>
        <div className="text-display">{studentData.fullName}</div>
        <button onClick={myClickHandler} className="close"> <=====
          <FaTimesCircle color="red" size={"2rem"} />
        </button>
        <button className="edit">
          <FaRegEdit color="orange" size={"2rem"} />
        </button>
      </Card>
    </>
  );
}

export default StudentItem;
```

As you can see, the event handler's name was just referenced (`onClick={myClickHandler}`) and not executed with the bracket.

2. By passing parameters: If you intend to pass parameters into the event handler, then you cannot call by reference. You have to call the event handler INSIDE AN EMPTY CALLBACK.

```
import { FaTimesCircle, FaRegEdit } from "react-icons/fa";
import Card from "../shared/Card";

function StudentItem({ studentData }) {
  const myClickHandler = (theId) => {
    console.log("Hello event id:", theId);
  };

  return (
    <>
      <Card>
        <div className="num-display">{studentData.attendance}</div>
        <div className="text-display">{studentData.fullName}</div>
        <button
          onClick={() => {
            myClickHandler(studentData.id); <=====
          }}
          className="close"
        >
          <FaTimesCircle color="red" size={"2rem"} />
        </button>
        <button className="edit">
```

```

        <FaRegEdit color="orange" size={"2rem"} />
      </button>
    </Card>
  </>
);
}

export default StudentItem;

```

Deleting StudentItem

To delete a StudentItem, the event handler has to be moved to the component where the state is, which is the App component.

Create a studentItemDeleteHandler function inside the App component and pass it down to the button firing the event inside the StudentItem component via props drilling.

App.jsx

```

import { useState } from "react";
import Header from "../components/Header";
import StudentList from "../components/StudentList";
import StudentData from "../data/studentData";

function App() {
  const [students, setStudents] = useState(StudentData);

  const studentItemDeleteHandler = (theId) => { <=====
    // console.log("App event id:", theId);
    if (window.confirm("Are you sure?")) {
      setStudents(
        students.filter((student) => {
          return student.id !== theId;
        })
      );
    }
  };

  return (
    <>
      <Header />
      <div className="container">
        <StudentList
          students={students}
          deleteHandler={studentItemDeleteHandler} <=====
        />
      </div>
    </>
  );
}

```

```

    );
  }

  export default App;

```

StudentList.jsx

```

import StudentItem from "./StudentItem";
function StudentList({ students, deleteHandler }) { <=====
  if (!students || students.length === 0) {
    return <h2>No student data available</h2>;
  }
  return (
    <div>
      {students.map((student) => (
        <StudentItem
          key={student.id}
          studentData={student}
          deleteHandler={deleteHandler} <=====
        />
      ))}
    </div>
  );
}

export default StudentList;

```

StudentItem.jsx

```

import { FaTimesCircle, FaRegEdit } from "react-icons/fa";
import Card from "./shared/Card";

function StudentItem({ studentData, deleteHandler }) { <=====
  return (
    <>
      <Card>
        <div className="num-display">{studentData.attendance}</div>
        <div className="text-display">{studentData.fullName}</div>
        <button
          onClick={() => {
            deleteHandler(studentData.id); <=====
          }}
          className="close"
        >
          <FaTimesCircle color="red" size={"2rem"} />
        </button>
      </Card>
    </>
  );
}

export default StudentItem;

```

```

        </button>
        <button className="edit">
          <FaRegEdit color="orange" size={"2rem"} />
        </button>
      </Card>
    </>
  );
}

export default StudentItem;

```

3.10 StudentSummary component

Now let add another component called `StudentSummary`. This will display the total number of students and their average attendance rating.

`App.jsx`

```

import { useState } from "react";
import Header from "../components/Header";
import StudentList from "../components/StudentList";
import StudentSummary from "../components/StudentSummary";
import StudentData from "../data/studentData";

function App() {
  const [students, setStudents] = useState(StudentData);

  const studentItemDeleteHandler = (theId) => {
    // console.log("App event id:", theId);
    if (window.confirm("Are you sure?")) {
      setStudents(
        students.filter((student) => {
          return student.id !== theId;
        })
      );
    }
  };

  return (
    <>
      <Header />
      <div className="container">
        <StudentSummary students={students} /> <====
        <StudentList
          students={students}
          deleteHandler={studentItemDeleteHandler}
        />
      </div>
    </>
  );
}

```

```
        </div>
      </>
    );
  }

  export default App;
```

StudentSummary.jsx

```
function StudentSummary({ students }) {
  let totalAttendance = 0;
  for (const astudent of students) {
    totalAttendance += astudent.attendance;
  }
  const avg = totalAttendance / students.length;
  return (
    <div className="student-stat">
      <h4>Student Count:{students.length}</h4>
      <h4>Student Attendance Avg:{totalAttendance}</h4>
    </div>
  );
}

export default StudentSummary;
```

