

React Fundamentals

Module 03

ADC502 : Web Development

What is React

- React is a popular JavaScript library for building user interfaces (UIs) and web applications.
- It was developed and is maintained by Facebook and a community of individual developers and companies.
- React is the view layer of an MVC application (Model View Controller)
- React is often used for building single-page applications (SPAs) and is known for its efficiency and performance.

Prerequisites

- HTML/CSS:
 - basic understanding of HTML and CSS for building web interfaces.
- JavaScript:
 - Familiarity with JavaScript, including ES6 (ECMAScript 2015) features.

Node.js and npm

- Install Node.js and npm (Node Package Manager).
- It can be downloaded them from the official website:
<https://nodejs.org/>

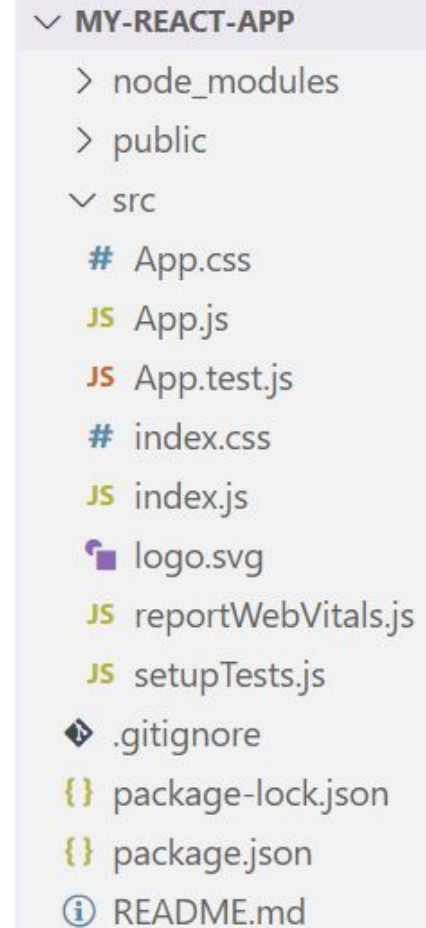
Create a React Application

- **Create React App** is a popular tool to set up a React development environment quickly.
- To start a new React project using Create React App, open terminal and run the following command:

```
npx create-react-app my-react-app
```
- This command will create a new directory called ***my-react-app*** with a basic React project structure.

Understanding the Project Structure

- Key folders and files include:
 - ***src/***: This is where your application's source code resides.
 - ***src/index.js***: The entry point of your application.
 - ***src/App.js***: The main component of your application.
 - ***public/index.html***: The HTML file where your React app is mounted.
 - ***package.json***: Contains project dependencies and scripts.



```
▼ MY-REACT-APP
  > node_modules
  > public
  ▼ src
    # App.css
    JS App.js
    JS App.test.js
    # index.css
    JS index.js
    logo.svg
    JS reportWebVitals.js
    JS setupTests.js
  .gitignore
  {} package-lock.json
  {} package.json
  ⓘ README.md
```

A screenshot of a file explorer showing the project structure of 'MY-REACT-APP'. The root directory contains 'node_modules', 'public', and 'src' folders, along with '.gitignore', 'package-lock.json', 'package.json', and 'README.md' files. The 'src' folder is expanded, showing 'App.css', 'App.js', 'App.test.js', 'index.css', 'index.js', 'logo.svg', 'reportWebVitals.js', and 'setupTests.js' files.

Run the Development Server

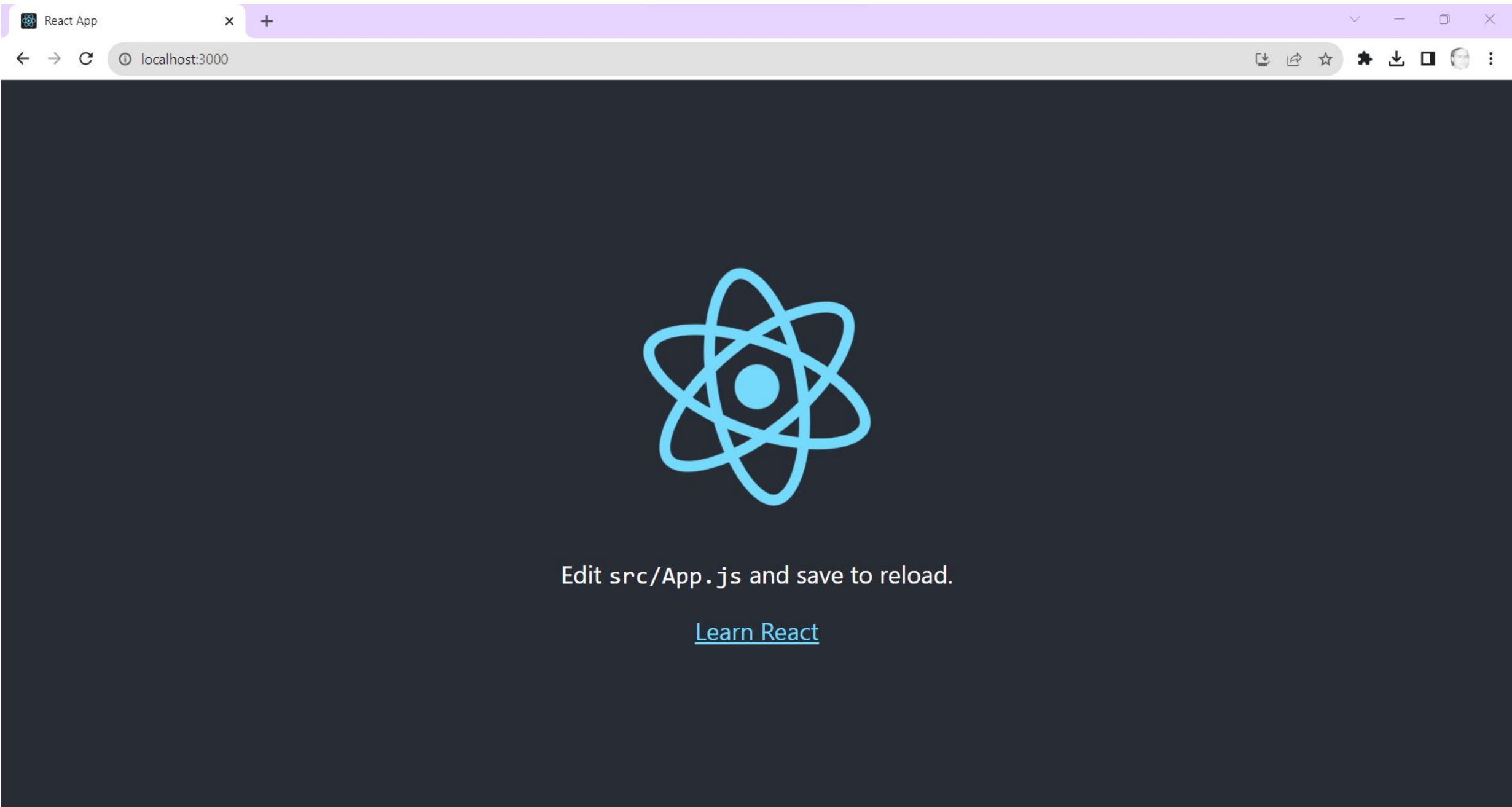
- Change into your project directory:

cd my-react-app

- Start the development server to see the React app in action

npm start

- This will open a new browser window with the React app running at <http://localhost:3000>.



Make 'Hello World' app

- Open the src/App.js file in a code editor.
- You'll see the default React component created by Create React App.
- Replace its contents with the following code

```
import React from 'react';
function App() {
  return (
    <div className="App">
      <h1>Hello, React!</h1>
    </div>
  );
}
export default App;
```

Add Custom Style

- This step is optional.
- update src/App.css with following content

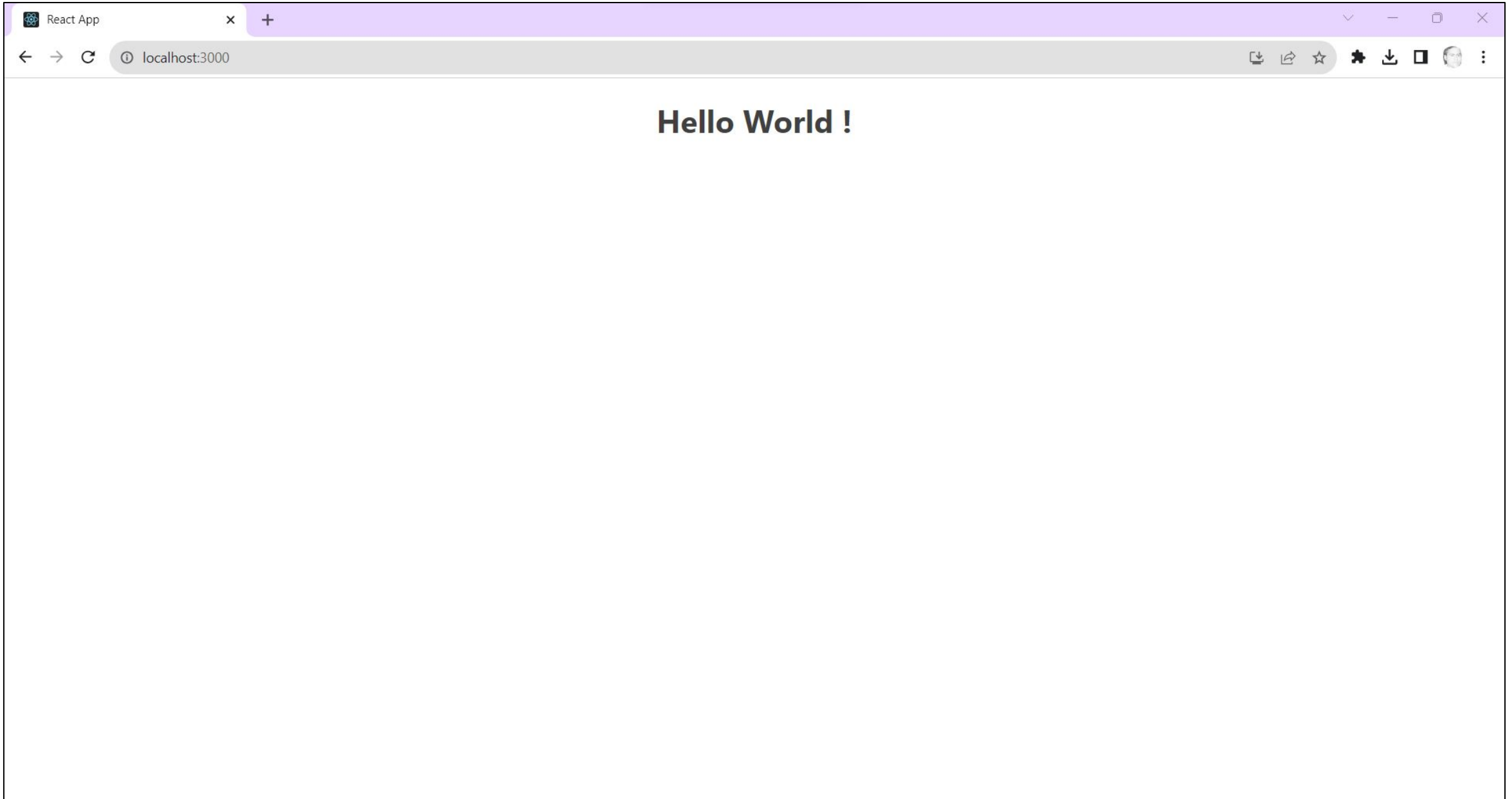
```
.App {  
  text-align: center;  
}
```

```
h1 {  
  color: #414141;  
}
```

Import CSS

- Import the CSS file into your App.js component:

```
import './App.css';
```



React Components

- In React, a component is a reusable building block that encapsulates a part of a user interface (UI) and its behavior.
- Components are the fundamental units of a React application and can be thought of as self-contained, isolated pieces of code that can be composed together to build complex UIs.
- There are two main types of components in React:
 - Function Components
 - Class Components

Function Components

- These are also known as stateless functional components.
- Function components are defined as JavaScript functions that take a set of input properties (called props) and return React elements that describe what should be rendered on the screen.
- Function components are typically used for simple, stateless parts of a UI.

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

Class Components

- Class components are defined as ES6 classes that extend the `React.Component` class.
- They have a `render()` method that returns the UI elements.
- Class components are used when you need to manage state or use lifecycle methods.

```
class Welcome extends React.Component {  
  render() {  
    return <h1>Hello, {this.props.name}</h1>;  
  }  
}
```

Components with other components

- Example of how to use a component within another component:

```
function App() {  
  return (  
    <div>  
      <Welcome name="Alice" />  
      <Welcome name="Bob" />  
    </div>  
  );  
}
```

- In this example, the Welcome component is used twice within the App component. Each instance of Welcome receives a different name prop.

Component Tree

- Components are the building blocks of a React application, and a typical React application is structured as a tree of components, with a single root component at the top.
- This component tree represents the entire UI of the application and is what makes React applications highly modular, maintainable, and easy to reason about.

State and Props

- Components can have properties, called props, which allow passing data from a parent component to a child component.
- These props are read-only and should not be modified within the child component.
- Components can also have their own internal state, which allows them to manage and update data that is specific to that component.
- Class components can define and manage state using the `setState` method, while functional components can use React Hooks like `useState` to manage state.

State and Props : Example

- In the following example, we'll create a simple "Counter" component that takes a starting value as a prop and allows you to increment and decrement the counter using buttons.
- The counter value will be stored in the component's state.

State and Props : Example

```
import React, { Component } from 'react';

class Counter extends Component {
  constructor(props) {
    super(props);

    // Initialize the component's state with the starting value received as a
    prop.
    this.state = {
      count: props.initialValue,
    };
  }

  // Function to increment the counter.
  increment = () => {
    this.setState({ count: this.state.count + 1 });
  };

  // Function to decrement the counter.
  decrement = () => {
    this.setState({ count: this.state.count - 1 });
  };

  render() {
    return (
      <div>
        <h2>Counter Value: {this.state.count}</h2>
        <button onClick={this.increment}>Increment</button>
        <button onClick={this.decrement}>Decrement</button>
      </div>
    );
  }
}
```

State and Props : Example

- The Counter component receives an initialValue prop, which is used to set the initial state of the counter.
- Inside the constructor, we initialize the component's state with the initialValue prop.
- We define two methods, increment and decrement, to update the state when the respective buttons are clicked. These methods use this.setState to modify the count property in the component's state.

State and Props : Example

- In the render method, we display the current counter value from the component's state along with "Increment" and "Decrement" buttons. When these buttons are clicked, the increment or decrement methods are called to update the state, causing the component to re-render with the new count value.
- You can use this Counter component in another component or in your App component, passing different initial values as props to create multiple counters with different starting values.

State and Props : Example

```
import React from 'react';
import Counter from './Counter';
function App() {
  return (
    <div>
      <Counter initialValue={0} />
      <Counter initialValue={5} />
      <Counter initialValue={10} />
    </div>
  );
}
export default App;
```

In this App component, we use the Counter component three times with different initial values, demonstrating the use of props and component state in React.

FAQ about Components, state and props

1. What is the difference between props and state in React?

- Props (Properties): Props are used to pass data from a parent component to a child component. They are immutable and read-only within the child component. Props are used for data that doesn't change within the child component.
- State: State is used to manage and store data that can change within a component. Components can have their own state, and changes to state trigger re-renders. State should be used for data that will change over time within the component.

FAQ about Components, state and props

2. Can props be changed within a component?

- No, props cannot be changed within a component. They are immutable and read-only. If you need to modify data within a component, you should use component state instead.

3. How do you pass props to a component?

- Props are passed to a component when you use that component in JSX. You specify the prop values as attributes. For example:

`<MyComponent name="John" age={30} />`

- In this example, name and age are props being passed to MyComponent.

FAQ about Components, state and props

4. How can a child component communicate with its parent component?

- Child components can communicate with their parent components by invoking callback functions passed to them as props.
- The child component can call these functions, passing data to the parent component.

5. When should you use component state versus props?

- Use props when you have data that is passed from a parent component and should not be modified within the child component.
- Use state when you need to manage and update data within a component, especially when that data can change over time due to user interactions or other factors.

FAQ about Components, state and props

6. Can a component have both state and props?

- Yes, a component can have both state and props.
- Props are used for data that comes from outside the component, while state is used for managing data within the component.

7. What is the purpose of the setState method?

- The setState method is used to update a component's state.
- When you call setState, React re-renders the component with the new state, reflecting any changes in the UI.

FAQ about Components, state and props

8. Can you directly modify state using assignment in React?

- No, you should never directly modify the state using assignment. Always use the `setState` method to update state.
- Direct assignment will not trigger a re-render, and React's state management won't work as expected.

9. What are the best practices for passing complex data as props?

- It's a good practice to pass down complex data as props in a structured format, such as objects or arrays.
- This makes the data easier to work with in the child component. If the data is deeply nested, consider using libraries like `Lodash` or functions like `map` to extract the required data.

FAQ about Components, state and props

10. What is the role of the constructor in a React component?

- The `constructor` is a special method used in class components to initialize the component's state and bind event handlers.
- It is typically used to set the initial state and perform any setup needed before rendering.

11. What is the purpose of components in React?

- Components in React promote code reusability, maintainability, and a modular structure.
- They allow you to break down complex UIs into smaller, manageable pieces.

FAQ about Components, state and props

12. What is state in React components?

- State is a way to manage and store data that can change within a component.
- It allows components to be dynamic and respond to user interactions or other triggers.
- Class components have a state object, while functional components can use React hooks to manage state.

13. How do you render a component in React?

- Components are rendered by invoking them within JSX. For example, `<MyComponent />` is used to render the MyComponent component.

FAQ about Components, state and props

14. How can components communicate with each other in React?

- Components can communicate by passing data through props. Parent components can pass data and functions as props to child components.
- For more complex communication, you can use state management libraries like Redux or context API.

15. Can components have their own styles?

- Yes, components can have their own styles.
- You can apply styles using CSS, CSS modules, or CSS-in-JS libraries like styled-components.

FAQ about Components, state and props

16. How can components be tested in React?

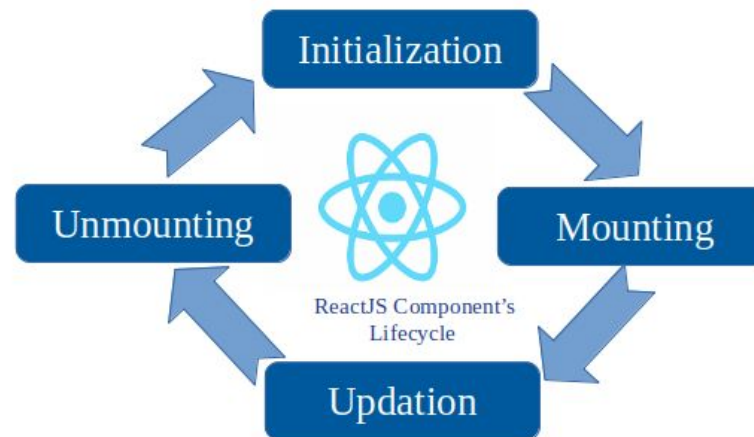
- Components can be tested using testing libraries like Jest and React Testing Library. You can write unit tests to ensure that your components behave correctly.

17. What is the lifecycle of a class component in React?

- Class components have several lifecycle methods, including `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount`, which allow you to perform actions at different stages of a component's existence.

Component Lifecycle

- The React component lifecycle refers to the various stages a React component goes through from its creation to its removal from the DOM.
- It's important to understand these stages to manage component behavior and side effects effectively.
- A component's lifecycle is broadly classified into four parts:
 - initialization
 - mounting
 - updating
 - unmounting



Initialization

- In the initialization phase, the **constructor** of the component is called.
- State can be initialized in the constructor.
- The component is not yet mounted to the DOM.

```
class ExampleComponent extends
React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }
  // ...
}
```

Mounting

- After initialization, the component is mounted into the DOM.
- The **componentDidMount** lifecycle method is called once after mounting, making it an ideal place for data fetching and initialization that requires access to the DOM.

```
componentDidMount() {  
  // Example: Fetch data from an API  
  // after the component is mounted.  
  
  fetch('https://api.example.com/data'  
  )  
    .then(response => response.json())  
    .then(data => {  
      this.setState({ data });  
    });  
}
```

Updating

- When a component's state or props change, it triggers the update phase.
- The `componentDidUpdate` lifecycle method is called after each update, allowing for side effects.

```
componentDidUpdate(prevProps, prevState) {  
  // Example: Check if props or state changed and  
  // perform actions accordingly.  
  if (this.props.someProp !==  
      prevProps.someProp) {  
    // Handle prop change  
  }  
  if (this.state.someState !==  
      prevState.someState) {  
    // Handle state change  
  }  
}
```

Unmounting

- If a component is removed from the DOM, the **componentWillUnmount** method is called.
- This is the place to clean up resources, such as canceling network requests or removing event listeners.

```
componentWillUnmount() {  
    // Example: Cleanup operations  
    // before unmounting the component.  
    // Cancel network requests,  
    // remove event listeners, etc.  
}
```

More to follow!