**React.js Full Course Notes**

**1. Introduction to React**

* **React.js**: A JavaScript library for building user interfaces, developed and maintained by Facebook.
* **Component-based**: Build encapsulated components that manage their own state.
* **Virtual DOM**: React uses a virtual DOM to improve rendering efficiency.
* **Single Page Application (SPA)**: React helps create SPAs, meaning the app loads a single HTML file and dynamically updates the content.

**2. React Setup**

1. **Using Create-React-App (CRA)**:

bash

Copy code

npx create-react-app my-app

cd my-app

npm start

1. **Folder Structure**:
   * src: Contains the main React code.
   * public: Stores static assets and the index.html.

**3. JSX (JavaScript XML)**

* **JSX**: Syntax extension for JavaScript that looks similar to HTML.
* Example:

jsx

Copy code

const element = <h1>Hello, React!</h1>;

* **Why JSX?**: Easier to read and write HTML-like code in JavaScript.

**4. Components**

* **Functional Component**:

jsx

Copy code

function Welcome() {

return <h1>Hello, World!</h1>;

}

* **Class Component** (older syntax):

jsx

Copy code

class Welcome extends React.Component {

render() {

return <h1>Hello, World!</h1>;

}

}

* **Component Props**:
  + Used to pass data between components.

jsx

Copy code

function Greet(props) {

return <h1>Hello, {props.name}</h1>;

}

**5. State and Lifecycle**

* **State**: A component's internal data that may change over time.

jsx

Copy code

function Counter() {

const [count, setCount] = React.useState(0);

return (

<div>

<p>You clicked {count} times</p>

<button onClick={() => setCount(count + 1)}>Click Me</button>

</div>

);

}

* **Lifecycle Methods** (for Class Components):
  + componentDidMount(): Runs after the component is rendered.
  + componentDidUpdate(): Runs after the component updates.
  + componentWillUnmount(): Runs before the component is removed.
  + **Props in React**
  + **Passing Props**:
  + You pass props from the parent component to the child component by adding them as attributes in JSX.
  + // Parent Component
  + function Parent() {
  + return <Child name="Alice" age={30} />;
  + }
  + // Child Component
  + function Child(props) {
  + return (
  + <div>
  + <h1>{props.name}</h1>
  + <p>Age: {props.age}</p>
  + </div>
  + );
  + }
  + **Accessing Props**:
  + Props are accessed using props object in the child component.
  + This object is passed automatically to function components or class components.
  + jsx
  + Copy code
  + function Child(props) {
  + return <h1>{props.name}</h1>;
  + }
  + For **destructuring props** directly:
  + jsx
  + Copy code
  + function Child({ name, age }) {
  + return (
  + <div>
  + <h1>{name}</h1>
  + <p>{age}</p>
  + </div>
  + );
  + }
  + **Prop Types**
  + **Prop Types** is a utility that allows you to define and validate the types of props passed to a component. It helps in development by ensuring the right type of data is passed to the component and provides warnings during development if the prop types do not match.
  + **Using PropTypes**:
  + You need to import PropTypes from the prop-types library to use it for prop type validation.
  + bash
  + Copy code
  + npm install prop-types
  + **Basic Usage**:
  + In your component, you can specify the expected type of props by using PropTypes.
  + jsx
  + Copy code
  + import PropTypes from 'prop-types';
  + function Child({ name, age }) {
  + return (
  + <div>
  + <h1>{name}</h1>
  + <p>{age}</p>
  + </div>
  + );
  + }
  + // PropType validation
  + Child.propTypes = {
  + name: PropTypes.string.isRequired,
  + age: PropTypes.number.isRequired
  + };
  + In the example above:
  + name is expected to be a string.
  + age is expected to be a number.
  + .isRequired means that these props are mandatory, and React will log a warning if they are missing.
  + **Common Prop Types**:
  + **PropTypes.string** – expects a string.
  + **PropTypes.number** – expects a number.
  + **PropTypes.bool** – expects a boolean.
  + **PropTypes.array** – expects an array.
  + **PropTypes.object** – expects an object.
  + **PropTypes.func** – expects a function.
  + **PropTypes.node** – can be anything that can be rendered (string, number, JSX, etc.).
  + **PropTypes.element** – expects a React element.
  + **PropTypes.arrayOf(type)** – expects an array of a certain type.
  + **PropTypes.objectOf(type)** – expects an object of a certain type.
  + **PropTypes.oneOf([value1, value2])** – expects one of a given set of values.
  + **PropTypes.any** – can be of any type.
  + **Example of PropTypes with Array and Object**:
  + jsx
  + Copy code
  + function Profile({ user, hobbies }) {
  + return (
  + <div>
  + <h1>{user.name}</h1>
  + <p>Age: {user.age}</p>
  + <h2>Hobbies:</h2>
  + <ul>
  + {hobbies.map((hobby, index) => (
  + <li key={index}>{hobby}</li>
  + ))}
  + </ul>
  + </div>
  + );
  + }
  + Profile.propTypes = {
  + user: PropTypes.shape({
  + name: PropTypes.string.isRequired,
  + age: PropTypes.number.isRequired
  + }).isRequired,
  + hobbies: PropTypes.arrayOf(PropTypes.string).isRequired
  + };
  + **Default Props**:
  + You can set default values for props in case they are not provided by the parent component using defaultProps.
  + jsx
  + Copy code
  + function Button({ label, color }) {
  + return <button style={{ backgroundColor: color }}>{label}</button>;
  + }
  + Button.defaultProps = {
  + label: 'Click me',
  + color: 'blue'
  + };
  + **Benefits of PropTypes**
  + **Type Safety**: It helps to catch errors early by checking the types of props.
  + **Documentation**: PropTypes can also serve as a form of self-documentation for the expected prop structure.
  + **Development Convenience**: It provides warnings in the browser console during development, helping to avoid common bugs due to incorrect prop types.
  + **Key Differences Between Props and State**
  + **Props**:
  + Passed down from parent to child.
  + Immutable within the component (cannot be modified).
  + Typically used for data or event handlers that a component receives.
  + **State**:
  + Managed within the component.
  + Mutable (can be updated using setState or useState).
  + Used for dynamic data that can change over time.

**6. Handling Events**

* Events are written in camelCase in React.

jsx

Copy code

function HandleClick() {

function handleClick() {

alert('Button clicked!');

}

return <button onClick={handleClick}>Click Me</button>;

}

**7. Conditional Rendering**

* Render elements conditionally using ternary operators or if statements.

jsx

Copy code

function Greeting({ isLoggedIn }) {

return (

<div>

{isLoggedIn ? <h1>Welcome Back!</h1> : <h1>Please Log In</h1>}

</div>

);

}

**8. Lists and Keys**

* Render lists dynamically with .map().

jsx

Copy code

const items = ['Apple', 'Banana', 'Cherry'];

function ItemList() {

return (

<ul>

{items.map((item, index) => (

<li key={index}>{item}</li>

))}

</ul>

);

}

* **Key Prop**: Ensures that elements are uniquely identified.

**9. Forms and Controlled Components**

* **Controlled Components**: Form elements controlled by React state.

jsx

Copy code

function MyForm() {

const [inputValue, setInputValue] = React.useState('');

const handleChange = (e) => setInputValue(e.target.value);

return (

<form>

<input type="text" value={inputValue} onChange={handleChange} />

</form>

);

}

**10. React Router (for Navigation)**

* Install React Router:

bash

Copy code

npm install react-router-dom

* Example of React Router usage:

jsx

Copy code

import { BrowserRouter, Route, Link } from 'react-router-dom';

function App() {

return (

<BrowserRouter>

<nav>

<Link to="/">Home</Link>

<Link to="/about">About</Link>

</nav>

<Route path="/" exact component={Home} />

<Route path="/about" component={About} />

</BrowserRouter>

);

}

**11. React Hooks**

* **useState**: Manage component state.
* **useEffect**: Perform side effects (e.g., fetching data).

jsx

Copy code

React.useEffect(() => {

console.log('Component rendered!');

}, []);

* **useContext**: Share data without props drilling.

jsx

Copy code

const MyContext = React.createContext();

**12. Context API for State Management**

* Use context to avoid passing props through every level of a component tree.

jsx

Copy code

const ThemeContext = React.createContext('light');

function App() {

return (

<ThemeContext.Provider value="dark">

<Toolbar />

</ThemeContext.Provider>

);

}

**13. React with APIs (Fetching Data)**

* Use fetch() or Axios to retrieve data from an API.

jsx

Copy code

function DataFetching() {

const [data, setData] = React.useState([]);

React.useEffect(() => {

fetch('https://jsonplaceholder.typicode.com/posts')

.then((response) => response.json())

.then((data) => setData(data));

}, []);

return (

<ul>

{data.map((item) => (

<li key={item.id}>{item.title}</li>

))}

</ul>

);

}

**14. React Performance Optimization**

* **React.memo**: Prevents unnecessary re-renders.

jsx

Copy code

const MemoizedComponent = React.memo(SomeComponent);

* **useCallback**: Memoizes functions between renders.

jsx

Copy code

const memoizedCallback = React.useCallback(() => doSomething(a, b), [a, b]);

**15. Deploying React App**

1. **Build the App**:

bash

Copy code

npm run build

1. **Deploy to Netlify / Vercel**:
   * Drag the build folder to the deployment platform.

**16. Advanced Concepts**

* **Redux**: State management library for large applications.
* **Server-side rendering (SSR)**: Improve SEO and initial loading time using frameworks like Next.js.
* **Code Splitting**: Load components only when needed using React.lazy

To install and set up a React project with Vite, follow these steps:

**Step 1: Install Vite and Create a React Project**

You can use either **npm** or **yarn** to install the project.

**Using npm:**

1. Open your terminal.
2. Run the following command to create a new Vite project with React template:

bash

Copy code

npm create vite@latest my-react-app

**Using yarn:**

1. Open your terminal.
2. Run the following command to create a new Vite project with React template:

bash

Copy code

yarn create vite@latest my-react-app --template react

Replace my-react-app with your desired project name.

**Step 2: Navigate to Your Project Folder**

Go to the newly created project directory:

bash

Copy code

cd my-react-app

**Step 3: Install Dependencies**

Now, install the required dependencies.

**Using npm:**

bash

Copy code

npm install

**Using yarn:**

bash

Copy code

yarn

**Step 4: Start the Development Server**

To start your React app:

**Using npm:**

bash

Copy code

npm run dev

**Using yarn:**

bash

Copy code

yarn dev