**JSX**

JSX (JavaScript XML) is a syntax extension for JavaScript that allows users to write HTML-like markup within a JavaScript file

JSX expressions can be used in if statements and for loops, assigned to variables, accepted as arguments, and returned from functions

**const numbers = [1, 2, 3, 4, 5];**

**const listItems = numbers.map((number) =>**

**<li key={number.toString()}>{number}</li>**

**);**

**const element = <ul>{listItems}</ul>;**

**Components**

In React, a component is a reusable piece of UI that can be independently defined, manipulated, and managed. Components are the building blocks of a React application, and they allow developers to break down complex user interfaces into smaller, more manageable parts. There are two main types of components in React: functional components and class components. Components take props as input and return a JSX tree as output

**Virtual DOM**

In React, the virtual DOM (VDOM) is a lightweight copy of the DOM (Document Object Model) that's kept in memory and synchronized with the real DOM The Virtual DOM is like a blueprint or a copy of the real DOM that is stored in the computer's memory. It's a concept used by React to make updating and changing things on a webpage more efficient. When we make changes to a webpage, like updating a list, traditional methods often involve updating the entire webpage, even if only a small part has changed. This can be slow and inefficient.

**Data Binding**

Data binding in React is the process of connecting the UI (user interface) elements to the data that drives the application In React, data flows in one direction: from parent to child components.

**Element**

In React, an element is the smallest building block of a React application. Unlike a component, which can be a complex piece of the UI, an element is a simple, immutable object that describes what you want to see on the screen. React elements are the objects that React uses to construct the user interface.

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

OR

const element = React.createElement('h1', null, 'Hello, world!');

**Function Components**

Function components are the simplest way to create a component in React. They are pure JavaScript functions that take a props object as the first parameter and return React elements to display the output.

**function Welcome({ personName }) {**

**return <h1>{`Welcome, ${personName}!`}</h1>;**

**}**

**Class Components**

Alternatively, you can use ES6 classes to define a component. The equivalent class component for the above function component would look like this

**class Greeting extends React.Component {**

**render() {**

**return <h1>{`Greetings, ${this.props.userName}!`}**

**</h1>;**

**}**

**}**

**What are keys in React and why do we need them?**

The "key" is a special attribute used when working with arrays of elements in React. It helps React keep track of changes, additions, and removals in the array. When you're rendering a list of items, React needs a way to identify each item uniquely. The "key" prop serves this purpose, allowing React to efficiently update the user interface.

You can use the "key" prop when mapping over this array to render each book:

**const bookItems = books.map((book) => <li**

**key={book.id}>{book.title}</li>);**

It's crucial to use unique keys among siblings to avoid issues.

The "key" attribute accepts either a string or a number, and it's converted internally to a string type.o y A warning message will appear in the console if the "key" prop is not present on list items.

**Create React App**

npx create-react-app my-react-app

npm start

**How are comments written in React?**

**<div>**

**{/\* Single-line comments(In vanilla JavaScript, the**

**single-line comments are represented by double**

**slash(//)) \*/}**

**{`Welcome, ${userName}! Let's dive into React`}**

**</div>**

**Explain how lists are created in React?`**

**import React from 'react';**

**import ReactDOM from 'react-dom';**

**// Example list of items**

**const fruits = ['Apple', 'Banana', 'Orange', 'Grapes',**

**'Watermelon'];**

**// Using map to create a list of JSX elements**

**const fruitList = fruits.map((fruit, index) => {**

**return <li key={index}>{fruit}</li>;**

**});**

**// Rendering the list inside an unordered list**

**ReactDOM.render(**

**<ul>**

**{fruitList}**

**Explain the difference between functional components and class components.**

**Functional components** are simple functions that accept props and return JSX. They are stateless and don't use a constructor or React lifecycle methods.

**Class components** are ES6 classes that extend React.Component. They have a render method, can use state, a constructor, and React lifecycle methods. They are suitable for managing state and implementing more complex logic.

**What are React Hooks?**

React Hooks are built-in functions introduced in React version 16.8 that allow developers to utilize state and lifecycle methods within functional components. They enhance code reusability and provide flexibility in navigating the component tree.

Before Hooks, class components were primarily used for managing state and lifecycle methods. With Hooks, developers can now access these features directly in functional components, eliminating the need for class components

**What is Strict Mode in React?**

Strict Mode is a feature in React designed to help developers write better and more resilient code. It activates additional checks and warnings for its descendants, aiming to highlight potential problems in an application. It's a development tool and does not affect the production build, meaning the checks and warnings are only active during development.

React Strict Mode is a powerful tool for improving the quality and reliability of your React application. By enabling additional checks and warnings, it helps developers identify potential issues early in the development process, ensuring that the codebase is robust, future-proof, and aligned with React’s best practices.

**import React from 'react';**

**import ReactDOM from 'react-dom';**

**import App from './App';**

**ReactDOM.render(**

**<React.StrictMode>**

**<App />**

**</React.StrictMode>,**

**document.getElementById('root')**

**)**

**What are the lifecycle methods of React?**

In React, lifecycle methods are special methods each component can use to run code at specific times in the component's life. They are primarily used in class components. Here’s an easy way to understand the most common lifecycle methods:

**Mounting**: These methods are called when a component is being inserted into the DOM

1. **constructor(props):** Initialize state and bind methods.
2. **componentDidMount()** Perform actions after the component has been inserted into the DOM, like fetching data or setting up subscriptions.

**Updating**: These methods are called when a component is being re-rendered due to changes in props or state.

1. **shouldComponentUpdate(nextProps, nextState)**: Decide whether to re-render the component based on changes in props or state.
2. **componentDidUpdate(prevProps, prevState) :** Perform actions after the component has updated in the DOM, like fetching new data if props have changed.

**Unmounting**: These methods are called when a component is being removed from the DOM.

1. **componentWillUnmount**() : Clean up things before the component is removed, like canceling network requests or clearing timers.

**Life Cycle Methods**

**constructor()**

* Called during component initiationg
* Sets up initial state and values.

**getDerivedStateFromProps()**

* Called just before rendering elements in the DOM.'
* Sets up the state based on initial props
* First method called on component update

**render()**

* Outputs or re-renders HTML to the DOM with new changes
* Essential method called on every render

**componentDidMount()**

* Called after component rendering.
* Executes statements requiring the component to be in the DOM

**shouldComponentUpdate()**

* Returns a Boolean specifying whether React should proceed with rendering.'
* Default value is true

**componentDidUpdate()**

* Called after the component is updated in the DOM

**componentWillUnmount()**

* Called when the component is about to be removed from the DOM

**Prop drilling in React**

Prop drilling in React is the process of passing data, or props, from a parent component to a child component through multiple levels of a component tree. It's a common practice, but can lead to issues when an app grows larger.

**What is React Router?**

React Router is like a navigation manager for React applications. It helps build single-page web apps where you can navigate to different sections without refreshing the entire page. This keeps the user experience smooth and also updates the browser URL as you move around.

In React, components are a big deal, and React Router uses this concept. You don't have to use React Router, but it's a popular choice for managing navigation

**BrowserRouter** : This is like the boss. It uses the HTML5 history API to keep your app in sync with the URL. It's like the container that holds all the other components

**Routes :** This is a newer addition to React (as of version 6). Think of it as an upgraded version of the component that helps with routing.

**Route**: This is where the action happens. Whenever the URL matches the path you set, this component decides what UI to show. It's like a conditionally displayed part of your app

**Link:** Similar to an anchor tag in HTML, this helps create links to different routes, making navigation smooth across your application

**npm install react-router-dom**

**// index.js**

**import React from 'react';**

**import ReactDOM from 'react-dom';**

**import { BrowserRouter as Router, Route, Switch, Link } from 'react-router-dom';**

**import Home from './Home';**

**import About from './About';**

**function App() {**

**return (**

**<Router>**

**<nav>**

**<ul>**

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

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

**</ul>**

**</nav>**

**<Switch>**

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

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

**</Switch>**

**</Router>**

**);**

**}**

**ReactDOM.render(<App />, document.getElementById('root'));**

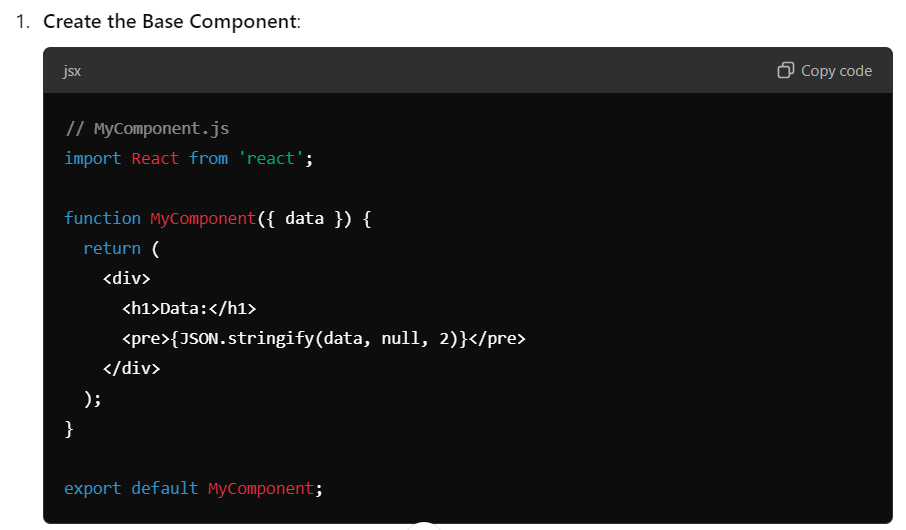
**What are higher order components in React?**

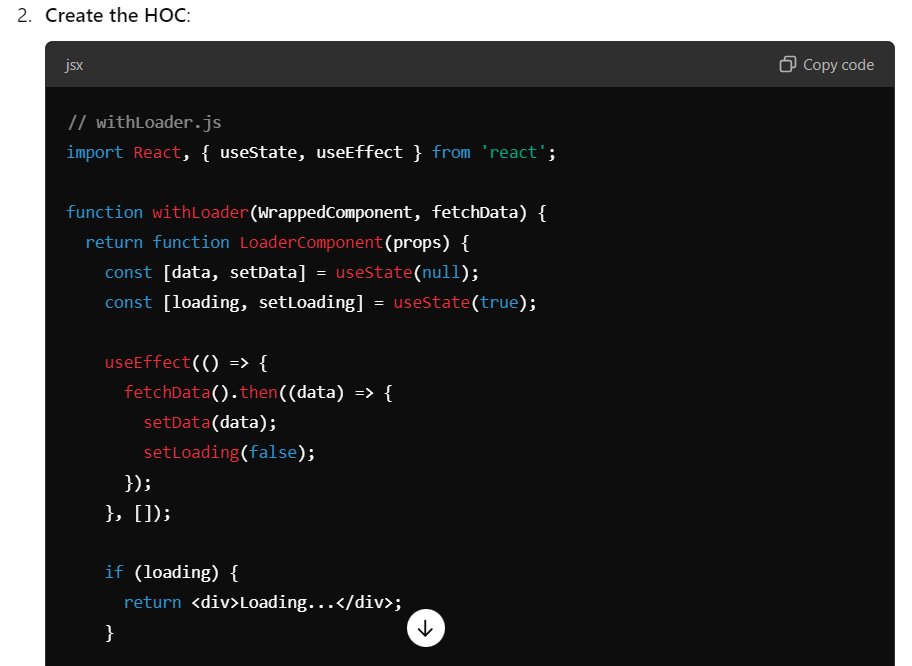
A Higher-Order Component (HOC) in React is a pattern that allows you to reuse component logic. It is a function that takes a component and returns a new component with additional props or behaviors.

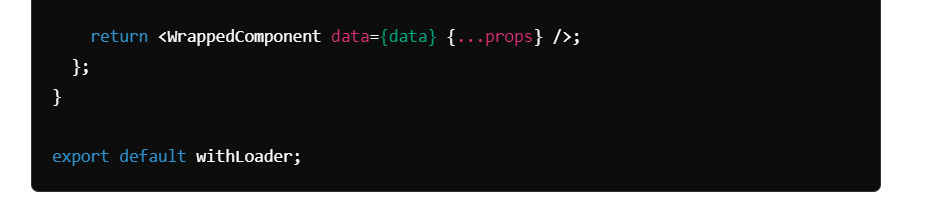
To share common logic between components without duplicating code.

HOCs can be used for tasks like authentication, logging, caching, etc.

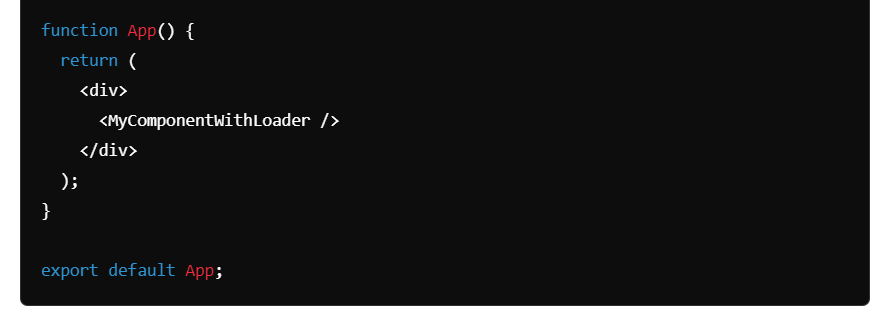
A Higher-Order Component (HOC) in React is a powerful pattern for reusing component logic. It allows you to enhance components with additional functionality by wrapping them in a function that returns a new component. This pattern promotes code reusability, separation of concerns, and makes it easy to add new behaviors to existing components.

****

****

****

****

****

**What is the difference between state and props?**

**State**: A local state is owned and managed by the component itself. It can be changed inside the component using setState().

**Props**: Props are read-only attributes passed from parent to child components. They allow you to pass data and event handlers down the component tree.

**What is the Context API?**

The Context API provides a way to pass data through the component tree without having to pass props down manually at every level. It is useful for global state management

**What are React Hooks?**

Hooks are functions that let you use state and other React features in functional components. Common hooks include **useState, useEffect, useContext,** and **useReducer.**

**What is useEffect used for?**

**useEffect** is a hook that lets you perform side effects in function components. It is similar to lifecycle methods like **componentDidMount**, **componentDidUpdate**, and **componentWillUnmount** combined.

**What are controlled and uncontrolled components?**

* **Controlled Components**: Form elements whose value is controlled by React state. They provide a single source of truth for form data.
* **Uncontrolled Components**: Form elements that maintain their own state and are accessed using refs.

**What are React Fragments?**

React Fragments allow you to group a list of children without adding extra nodes to the DOM. They can be used with the <React.Fragment> tag or the shorthand <> </>.

**Explain the concept of lifting state up in React.**

Lifting state up refers to the process of moving state to a common ancestor of components that need to share this state. It helps manage shared state and enables better communication between components

**Explain styled components in React?**

Styled-components is a popular library for styling React applications using a CSS-in-JS approach. It allows you to write plain CSS in your JavaScript files and bind styles directly to components, making it easier to manage and scope styles.

Key Features:

* **CSS-in-JS**: Write CSS directly in your JavaScript files.
* **Scoped Styles**: Styles are scoped to the component, preventing conflicts.
* **Dynamic Styling**: Style components based on props or application state.
* **Theming**: Easily create and manage themes.

