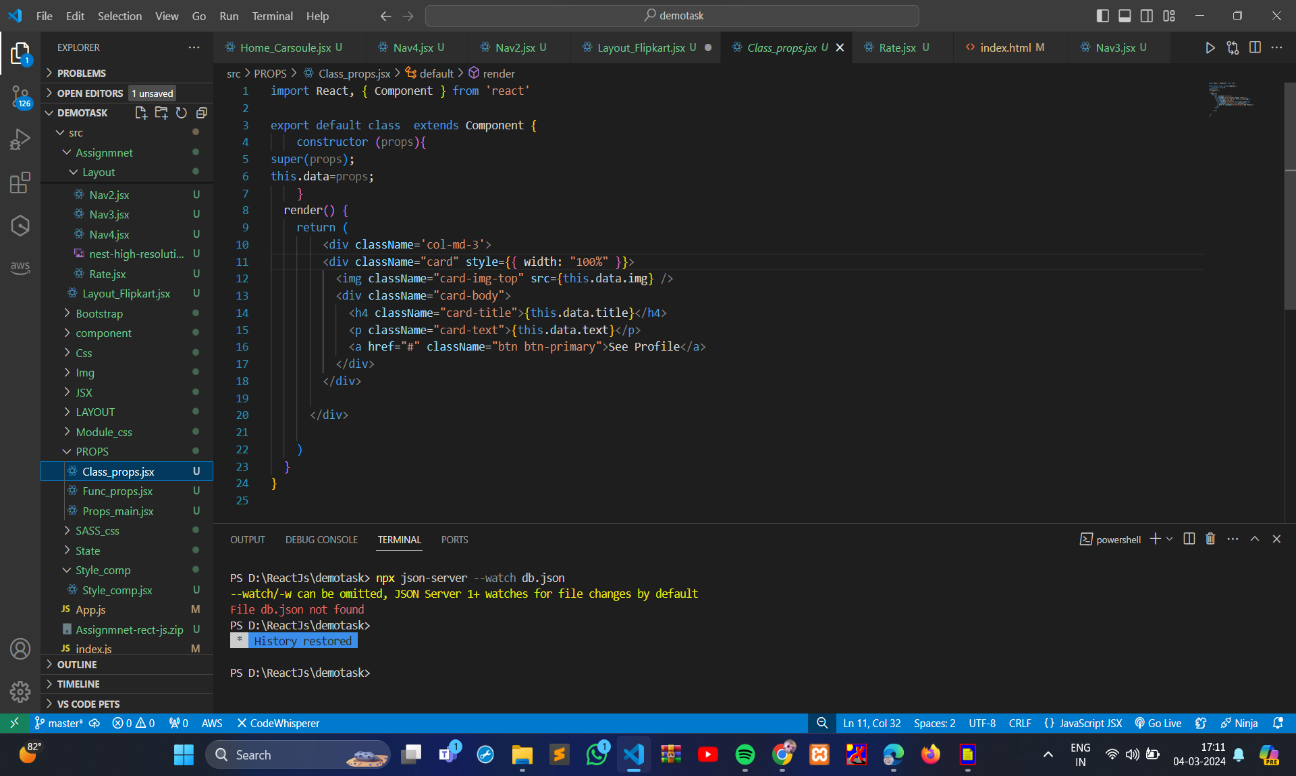
**React Component Lifecycle: Class Components vs Functional Components with Hooks**

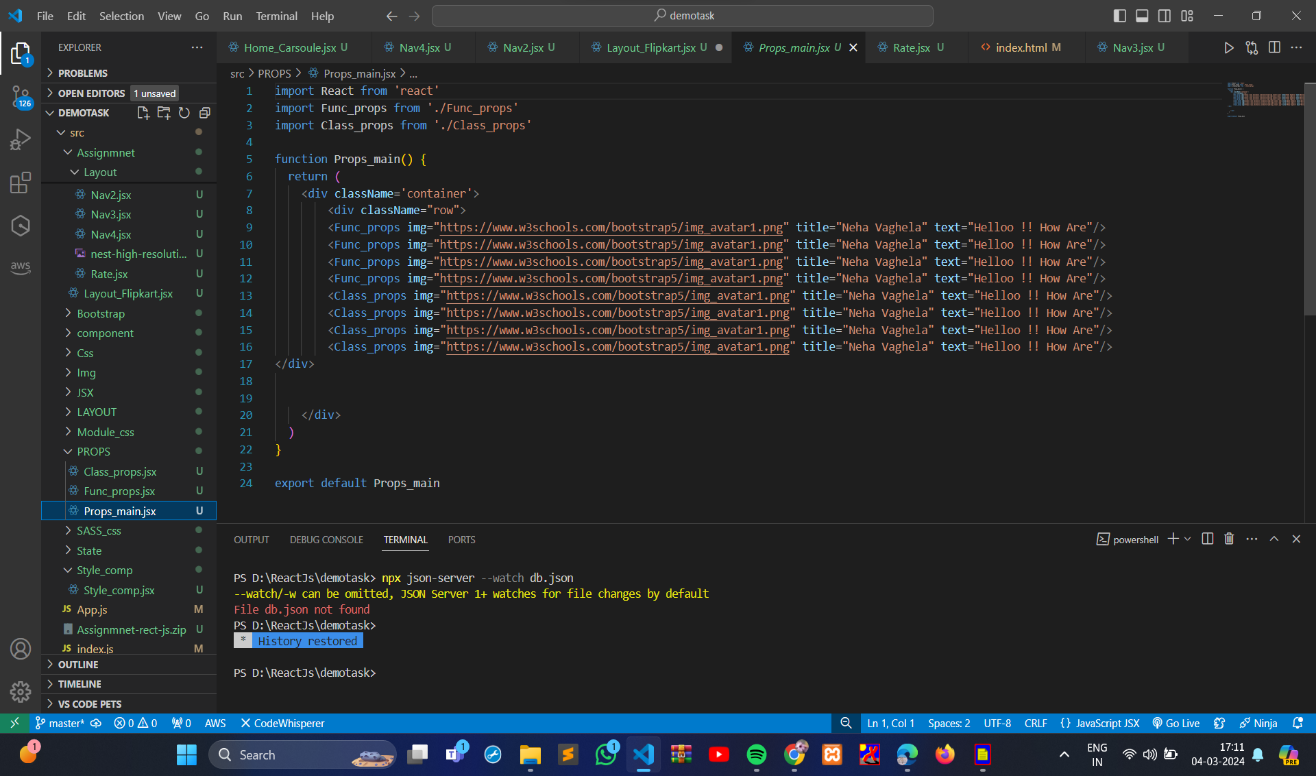
Both class components and functional components with hooks can leverage the React component lifecycle. This lifecycle refers to the different stages a component goes through, from its creation to its eventual removal from the DOM (Document Object Model).

**Class Components:**

Traditionally, class components managed the lifecycle using special methods. These methods allowed developers to perform actions at specific stages. Here's a breakdown of the key lifecycle methods:

* **Mounting:**
  + constructor(props): Used for initializing state and binding event handlers.
  + componentDidMount(): Invoked immediately after a component is mounted (inserted into the DOM). Ideal for fetching data, setting up subscriptions, or DOM manipulations.
* **Updating:**
  + getDerivedStateFromProps(nextProps, prevState) (optional): Used to update state based on changes in props.
  + shouldComponentUpdate(nextProps, nextState) (optional): Controls whether a component should update based on prop or state changes.
  + componentDidUpdate(prevProps, prevState): Invoked immediately after updating occurs. Useful for side effects after an update, like updating DOM based on state changes.
* **Unmounting:**
  + componentWillUnmount(): Invoked immediately before a component is unmounted from the DOM. Used for cleanup tasks like removing event listeners or subscriptions to prevent memory leaks.





**Functional Components with Hooks:**

With the introduction of hooks in React 16.8, functional components gained the ability to manage state and side effects. Hooks provide a more concise and functional approach to the lifecycle compared to class methods. Here are the key hooks for mimicking lifecycle behavior:

* **Mounting:**
  + useState: Used to initialize state within the component.
  + useEffect: A general-purpose hook for performing side effects like data fetching, subscriptions, or DOM manipulations after the component renders. You can configure it to run only on mount by passing an empty dependency array [].
* **Updating:**
  + Similar to mounting, you use useEffect with appropriate dependencies in the array to trigger side effects based on specific prop or state changes.
* **Unmounting:**
  + Inside the useEffect callback, you can return a cleanup function that gets executed when the component unmounts. This is useful for removing event listeners or subscriptions.

