

1. How do you create Nested Routes react-router-dom configuration ?

To create nested routes using react-router-dom, we start by importing the necessary functions like `createBrowserRouter` and `RouterProvider`. We then define a root route object, which serves as the parent. Inside this object, we add a `children` array where each child represents a nested route with its own path and element. This object structure can also be further nested if needed. In the component that represents the parent route, we use the `<Outlet />` component to render the child routes at the appropriate place. Finally, we pass the router configuration to `<RouterProvider />` to enable routing in the application. This setup allows us to build hierarchical navigation and layouts efficiently.

```
const AppLayout = () => {
  return (
    <>
      <Header />
      <Outlet />{" "}
      { /* This will be replaced by the children of the AppLayout based on the path */ }
    </>
  );
};

const appRouter = createBrowserRouter([
  {
    path: "/",
    element: <AppLayout />,
    children: [
      { path: "/", element: <Body /> },
      {
        path: "/about",
        element: <About />,
      },
      {
        path: "/contact",
        element: <Contact />,
      },
      {
        path: "/restaurants/:resId", // Dynamic path for the restaurants
        element: <RestaurantMenuPage />,
      },
    ],
  },
  {
    path: "*",
    element: <Error />,
  },
], {
  errorElement: <Error />,
});

const root = createRoot(document.getElementById("root"));
root.render(<RouterProvider router={appRouter} />);
```

2. Read about `createHashRouter`, `createMemoryRouter` from React Router docs.

✓ Use `createHashRouter` when the server cannot handle multiple routes.

More specifically:

- `createHashRouter` is used when your hosting environment or server cannot handle deep links or route requests like `/about` or `/restaurants/123` — meaning, it doesn't know how to serve `index.html` for any route.
- Instead of relying on the server to handle those paths, `createHashRouter` uses the `#` (hash) in the URL (e.g., `/#/about`), and everything after `#` is handled entirely by React on the client side.

What does `createMemoryRouter` do?

`createMemoryRouter` is used to create a router that keeps the navigation history in memory instead of using the browser's address bar or URL. This means the URL does not change in the browser, and the routing is fully controlled programmatically.

It is mainly used for:

- Testing React Router logic without a real browser
- Embedding apps in environments without a real URL bar (like native mobile apps or Electron)
- Server-side rendering (SSR), where you simulate navigation in memory

3. What is the order of life cycle method calls in Class Based Components ?

✓ Order of Lifecycle Method Calls in Class-Based Components

When a class-based component is used in a React application, the following lifecycle methods are called in this order:

1. **`constructor()`** — This is the first method that runs. It is used to initialize state and bind methods.
2. **`render()`** — This method returns the JSX to display on the screen.
3. **`componentDidMount()`** — This method runs after the component is mounted (inserted into the DOM). It's commonly used to perform API calls or set up subscriptions.

If the component's **state or props change**, the following methods are called:

4. **`render()`** — Called again to update the UI with new data.
5. **`componentDidUpdate()`** — Runs after the re-render caused by state or prop updates. You can use it to respond to those changes.

Finally, when the component is removed from the DOM (unmounted):

6. **componentWillUnmount()** — This method is called. It is typically used to clean up tasks like clearing timers or unsubscribing from events.



Quick Recap Order:

constructor() → render() → componentDidMount() → [update? → render() → componentDidUpdate()] → componentWillUnmount()

4. Why do we use componentDidMount?

The componentDidMount() lifecycle method is called once, immediately after the component is mounted (i.e., inserted into the DOM). It runs after the render() method finishes execution.

This method is commonly used for:

- Making API calls to fetch data from a server
- Setting up event listeners
- Starting timers or intervals
- Performing side effects that require access to the rendered DOM

Since it only runs once during the component's lifecycle, it is ideal for initializing data or setting up resources the component needs.

5. Why do we use componentWillUnmount? Show with an example

The componentWillUnmount() method is called just before a component is removed from the DOM.

We use this method to clean up anything that was set up while the component was active, such as:

- Clearing timers or intervals
- Removing event listeners
- Canceling API requests or subscriptions
- Releasing memory or resources

Using this method prevents memory leaks and ensures that your app remains efficient.


Example code-

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

class TimerComponent extends Component {
  componentDidMount() {
    this.timer = setInterval(() => {
      console.log("Timer running...");
    }, 1000);
  }

  componentWillUnmount() {
    clearInterval(this.timer); // 🖱️ Clean up the interval
    console.log("Timer cleared!");
  }

  render() {
    return <h2>Timer is active. Check the console!</h2>;
  }
}
```



6. Why do we use `super(props)` while creating a constructor in React class based components ?

In class-based components, we use `super(props)` inside the constructor because our component is extending `React.Component`, which is the parent class.

Calling `super(props)`:

- Passes the props to the parent class constructor
- Makes `this.props` available inside the constructor and throughout the component
- Ensures that the component is properly initialized according to React's internal behavior

If you don't call `super(props)`, you won't be able to access `this.props` inside the constructor, which may lead to errors.

7. Why can't we have the callback function of `useEffect` async?

The callback function passed to `useEffect` cannot be async directly because:

1. An async function always returns a promise, not a cleanup function.
2. But `useEffect` expects the callback to either:
 - Return nothing, or
 - Return a cleanup function (to run when the component unmounts or dependencies change).

If you make the effect callback async, it returns a promise — and React doesn't know what to do with it. This could lead to unexpected behavior or memory leaks.