Routing in React (React Router)

Question 1: What is React Router? How does it handle routing in single-page applications?

Answer:

React Router is a popular JavaScript library that enables client-side routing in single-page applications (SPAs) built with React. It allows you to manage multiple routes, navigate between them, and maintain a clean, scalable codebase.

How React Router Handles Routing

Here's a high-level overview of how React Router handles routing:

1. Route Configuration: You define routes using the Route component, specifying the path, component, and any additional props.
2. Route Matching: When the user navigates to a URL, React Router matches the URL path against the configured routes.
3. Component Rendering: If a match is found, React Router renders the associated component.
4. Navigation: When the user navigates to a new route, React Router updates the URL and re-renders the new component.
5. History Management: React Router uses the HTML5 History API to manage the browser's history, allowing users to navigate back and forth between routes.

Key Features of React Router

1. Declarative Routing: Define routes using a declarative syntax.
2. Client-Side Routing: Handle routing on the client-side, without requiring server-side requests.

3. Dynamic Routing: Use parameters and query strings to create dynamic routes.
4. Nested Routing: Create nested routes to manage complex navigation hierarchies.

By using React Router, you can build scalable, maintainable, and user-friendly single-page applications with robust routing capabilities.

Question 2: Explain the difference between BrowserRouter, Route, Link, and Switch components in React Router.

Answer:

1. BrowserRouter

- Wraps the entire application to enable client-side routing.
- Uses the HTML5 History API to manage the browser's history.

2. Route

- Defines a mapping between a URL path and a component.
- Can have multiple props like path, component, render, and children.

3. Link

- Creates a link to navigate between routes.
- Replaces the traditional anchor tag to prevent full-page reloads.

4. Switch

- Renders the first matching Route or Redirect.
- Useful for handling multiple routes with different paths.

Example

jsx

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

function App() {
  return (
    <BrowserRouter>
      <div>
        <h1>Navigation</h1>
        <ul>
          <li>
            <Link to="/">Home</Link>
          </li>
          <li>
            <Link to="/about">About</Link>
          </li>
        </ul>

        <Switch>
          <Route exact path="/" component={Home} />
          <Route path="/about" component={About} />
        </Switch>
      </div>
    </BrowserRouter>
  );
}
```

Context API

Question 1: What is the Context API in React? How is it used to manage global state across multiple components?

Answer:

Context API in React

The Context API is a built-in React API that allows you to share data between components without passing props down manually. It provides a way to manage global state across multiple components.

How Context API Works

1. Create a Context: Create a context object using the React.createContext() method.
2. Provider Component: Wrap your app with a Provider component, passing the context object as a prop.
3. Consumer Component: Use the Consumer component to access the context data in any component.

Example

jsx

```
// Create a context
const ThemeContext = React.createContext();

// Provider component
const ThemeProvider = ({ children }) => {
  const theme = { background: 'light', text: 'dark' };
  return (
    <ThemeContext.Provider value={theme}>
      {children}
```

```
    </ThemeContext.Provider>
  );
};

// Consumer component
const Button = () => {
  return (
    <ThemeContext.Consumer>
      {(theme) => (
        <button style={{ background: theme.background, color:
theme.text }}>
          Click me
        </button>
      )}
    </ThemeContext.Consumer>
  );
};
```

Benefits of Context API

1. Easy state management: Manage global state without passing props down manually.
2. Decoupling: Components are decoupled from each other, making it easier to maintain and update.
3. Reusability: Components can be reused across the app without worrying about state management.

Question 2: Explain how createContext() and useContext() are used in React for sharing state.

Answer:

createContext() and useContext()

- createContext(): Creates a context object to share state between components.
- useContext(): Hooks into the context object to access the shared state.

Example

jsx

```
const ThemeContext = React.createContext();

const ThemeProvider = ({ children }) => {
  const theme = { background: 'light', text: 'dark' };
  return (
    <ThemeContext.Provider value={theme}>
      {children}
    </ThemeContext.Provider>
  );
};

const Button = () => {
  const theme = React.useContext(ThemeContext);
  return (
    <button style={{ background: theme.background, color: theme.text
}}>
      Click me
    </button>
  );
};
```