### **Connecting the React Frontend to the Golang Backend**

The React app can use the fetch API or libraries like Axios to request HTTP to the Golang backend. State management in React allows the app to store and update the UI based on the data received from the server.

*1*fetch('/api/data')

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

*3* .then(data => {

*4* *// Update the state with the data received from the backend*

*5* });

*6*

### **Routing in React with React Router**

For single-page applications built with React, react-router-dom is the de facto library for handling navigation and routing without full page reloads.

### **Setting Up 'react-router-dom' for Navigation in the React App**

To use React Router, you install the package and define routes corresponding to your application's different components.

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

*2*

*3*function App() {

*4* return (

*5* <Router>

*6* <Switch>

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

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

*9* // Other routes

*10* </Switch>

*11* </Router>

*12* );

*13*}

*14*

### **Setting Up Go Server for Hot Reloading During Development**

For the Golang server, hot reloading isn't built-in, but tools like fresh or realize can be used to watch for file changes and restart the server automatically.

*1# Install a hot reload tool for Golang*

*2*go get github.com/pilu/fresh

*3*

Then, you can start your server with the tool to enable hot reloading:

*1*fresh

*2*

### **Writing Unit Tests for React Components**

React testing libraries like Jest and React Testing Library allow developers to write comprehensive tests for their components.

*1*import { render, screen } from '@testing-library/react';

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

*3*

*4*test('renders learn react link', () => {

*5* render(<App />);

*6* const linkElement = screen.getByText(/learn react/i);

*7* expect(linkElement).toBeInTheDocument();

*8*});

*9*