National University of Computer and Emerging Sciences

Laboratory Manual

for

Web Engineering
(SL3003)

Lab Instructors	Sana Ejaz, Hassan Raza
Section	6C 1,2
Semester	Spring 2025

Department of Software Engineering

FAST-NU, Lahore, Pakistan

Interactive Bookstore App with React Router

Scenario:

You're tasked with creating a functional mockup for a documentation website covering key React concepts. The primary goal is to demonstrate your understanding of **React Router (v6)** by implementing a multi-page application structure with shared layouts, dynamic content based on URL parameters, nested views, and various navigation methods. State management beyond local component state is **not** required for this lab.

Assumed Data:

Use a simple, hardcoded array of objects to represent the documentation concepts. Each object should have at least an id (a URL-friendly string like 'state' or 'props'), a name (display-friendly title), a description, and an array of simple examples (strings).

```
// Example Sample Data (Place this in a suitable location)
const conceptsData = [
  {
    id: 'state',
    name: 'State',
    description: 'State allows React components to change their output over time in response to user actions, network responses, and anything else.',
    examples: [
      'Using useState hook: const [count, setCount] = useState(0);',
      'Updating state based on previous state: setCount(prevCount => prevCount + 1);'
    ]
  },
  {
    id: 'props',
    name: 'Props',
```

```
    description: 'Props (short for properties) are read-only inputs to
components. They allow passing data from parent to child components.',
    examples: [
      'Passing a prop: <Welcome name="Sara" />',
      'Accessing a prop: function Welcome(props) { return <h1>Hello,
{props.name}</h1>; }',
      'Using prop-types for validation (optional concept).'
    ]
  },
  {
    id: 'hooks',
    name: 'Hooks',
    description: 'Hooks are functions that let you “hook into” React
state and lifecycle features from function components.',
    examples: [
      'useState for state management.',
      'useEffect for side effects.',
      'useContext for accessing context.',
      'Custom Hooks for reusable logic.'
    ]
  },
  // Add 1-2 more concepts if you want, this should suffice for the
lab though
];
```

Tasks:

Part 1: Basic Setup, Layout, and Navigation (React Router Basics)

1. **Dependencies:** Ensure react-router-dom (v6+) is installed.
2. **Router Setup:** Configure BrowserRouter in your application's entry point (e.g., index.js).
3. **Layout Component:** Create a reusable Layout component. This component should

render:

   - A header section containing navigation links. Use NavLink for these links (pointing to "Home", "Concepts", "About") so you can visually indicate the active page. Apply some basic active styling (e.g., change color or add an underline).
   - A main content area where child routes will be rendered using the <Outlet /> component.

4. **Route Configuration:** Inside your main App component, set up your <Routes>. Define a parent Route that uses the Layout component for its element. Define the following child routes *within* this layout route:
   - /: An index route rendering a HomePage component (create a simple placeholder).
   - /concepts: Renders a ConceptsOverviewPage component (create a placeholder).
   - /about: Renders an AboutPage component (create a simple placeholder).
5. **Placeholder Components:** Create the basic functional components (HomePage, ConceptsOverviewPage, AboutPage) displaying at least their respective titles.
   - **Hint:** Pay attention to how NavLink's className or style prop can accept a function to determine active state styling.

Part 2: Dynamic Routes for Concept Details (Dynamic Segments & useParams)

1. **Concepts Listing:** Modify the ConceptsOverviewPage. Import your conceptsData. Map over the data and display a list of concept names (e.g., "State", "Props", "Hooks").
2. **Dynamic Links:** Make each concept name in the list a Link (or NavLink) that navigates to a unique URL path for that concept's detail page, incorporating its id (e.g., /concepts/state, /concepts/props).
3. **Dynamic Route Definition:** Add a new route definition *within* the layout route structure to handle these dynamic paths: /concepts/:conceptId. This route should render a ConceptDetailPage component (create this component).
4. **Fetching & Displaying Details:** Implement the ConceptDetailPage.
   - Use the useParams hook to extract the conceptId from the URL.
   - Find the specific concept object from your conceptsData array that matches the extracted conceptId.
   - Display the concept's name and description. (Handle the case where no matching concept is found, perhaps by displaying a message, though a 404 page will be

added later).

- **Hint:** Remember that URL parameters obtained via useParams are initially strings.

Part 3: Nested Routes for Examples (Nesting & Outlet)

1. **Nested Route Definition:** Define a *nested* route structure. Modify the dynamic route /concepts/:conceptId to make it a parent route capable of having its own children. Add a child Route to it with the path examples. This child route should render an ExamplesPage component (create this component). The full path to this nested route will be like /concepts/state/examples.
2. **Linking to Nested Route:** On the ConceptDetailPage, add a Link (e.g., "View Examples") that specifically navigates to the examples nested route for the *current* concept being viewed.
3. **Rendering Nested Content:** Crucially, modify the ConceptDetailPage component's JSX. In addition to displaying the concept name and description, you must include *another* <Outlet /> component within it. This second Outlet is where the component for the nested route (ExamplesPage) will be rendered.
    - **Hint:** Think about the hierarchy. The main Layout has an Outlet for pages like ConceptDetailPage. The ConceptDetailPage *itself* now needs an Outlet to render *its* child routes like ExamplesPage.

Part 4: Nested Content Display & Programmatic Navigation (useParams & useNavigate)

1. **Displaying Examples:** Implement the ExamplesPage component.
    - Even though it's nested, it still needs to know which concept's examples to show. Use the useParams hook *again* here to get the conceptId from the URL.
    - Find the corresponding concept object in conceptsData using the conceptId.
    - Display the examples array associated with that concept (e.g., as a list).
2. **Programmatic Navigation:** Add a "Back to Concept Details" button on the ExamplesPage.
    - Import and use the useNavigate hook.
    - Implement an onClick handler for the button that uses the Maps function to go back to the parent concept's detail page (e.g., navigate from /concepts/state/examples back to /concepts/state). You can achieve this using

relative path navigation.

- **Hint:** useNavigate can accept numbers (like -1 for back) or path strings (like '..' for one level up, or absolute paths like '/concepts/${conceptId}'). Relative paths are often useful here.

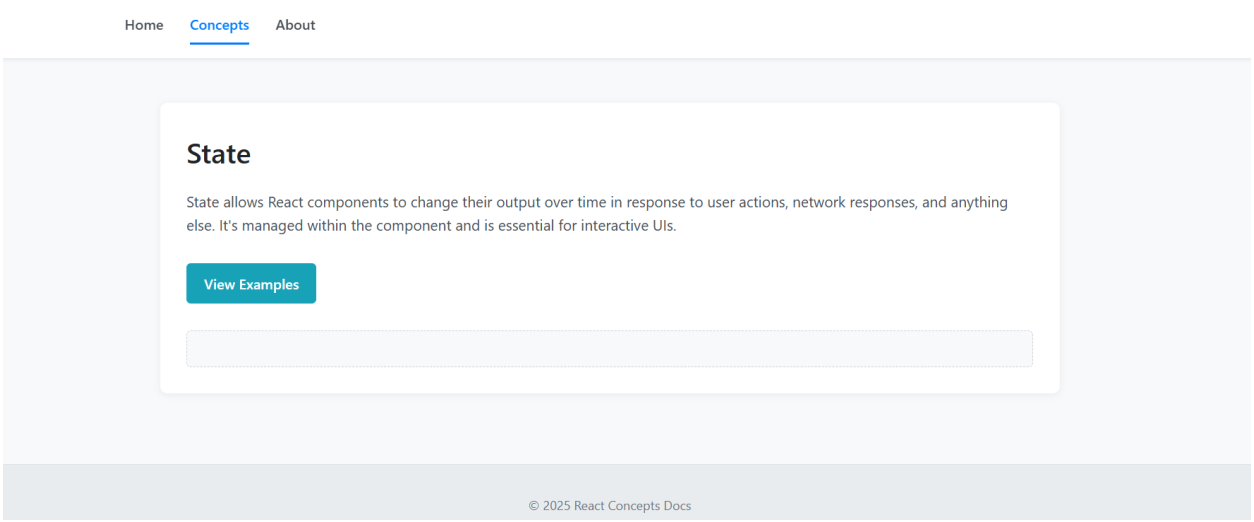
Part 5: Handling Not Found Pages (Catch-all Route)

1. **Not Found Component:** Create a simple NotFoundPage component that displays a clear "404 - Page Not Found" message.
2. **Catch-all Route:** Add a final Route within your main <Routes> configuration with path="*" . This route should render your NotFoundPage component. Ensure this route is placed *after* all other routes and is likely *outside* the main Layout route structure if you don't want the header/footer on the 404 page.

Evaluation Criteria:

- Correct setup and usage of BrowserRouter, Routes, Route, Outlet.
- Effective implementation of a shared Layout component using Outlet.
- Proper use of NavLink for navigation with active state indication.
- Successful definition and handling of dynamic routes using URL parameters (:conceptId).
- Correct usage of useParams to read URL parameters in both parent and nested routes.
- Correct implementation of nested routes and the use of nested Outlet components.
- Demonstrated ability to navigate programmatically using useNavigate.
- Proper implementation of a catch-all "Not Found" route.
- Clear, well-structured code demonstrating understanding of React Router v6 concepts.

Good luck building your documentation site structure!



Here are a few reference screenshots (strict following of this this style, colors, fonts aren't necessary). This is just to give you an idea. Be creative!