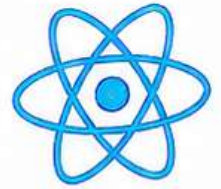


Chapter 9: React Router

9.1 Introduction to React Router

- * React Router is the standard **routing library** for React applications.
- * It enables **navigation** between views of various components in a React Application.
- * It keeps the **UI** in sync with the **URL**.
- * With React Router, we can build **Single Page Applications (SPAs)** with multiple pages without reloading the browser.
- * It allows us to define **routes**, **navigate** between pages and pass data using **URL parameters**.

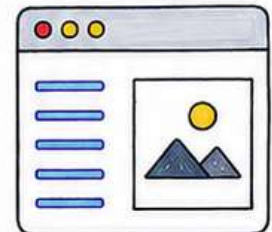


Key Points :

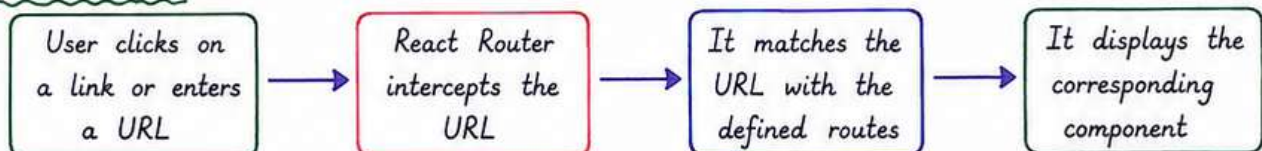
- ✓ React Router helps in building dynamic and interactive web apps.
- ✓ It provides a declarative way to handle routing in React.
- ✓ It enables navigation without full page reload (client-side routing).
- ✓ It supports nested routes, route parameters and dynamic routing.

9.1.1 What is Routing?

- * Routing is the process of **navigating** from one view or page to another based on the **URL**.
- * In traditional web applications, routing is handled by the **server**.
- * In React (SPA), React Router handles routing on the **client-side** without refreshing the page.



How it Works?



Example :

URL	Component Rendered
/	Home Component
/about	About Component
/contact	Contact Component
/user/1	User Component (with ID 1)

React Router makes our application behave like a **multi-page** website without reloading pages.



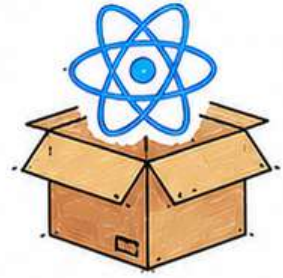
Key Takeaway :



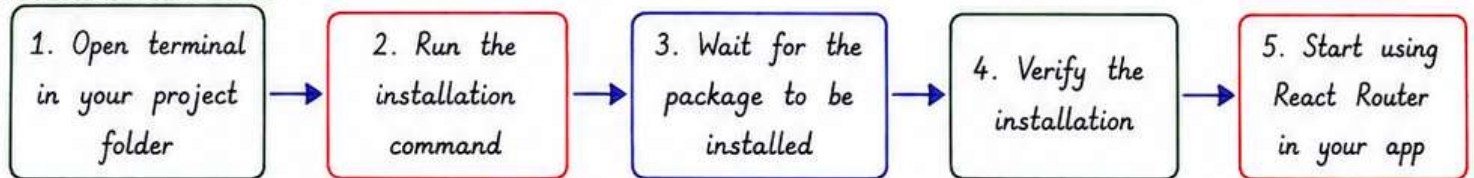
React Router allows us to build **fast**, **dynamic** and **user-friendly** applications by managing navigation between components using **URLs**.

9.2 Installation

- * To use React Router in our React application, we need to install the **react-router-dom** package.
- * React Router v6 and above are built on top of the **react-router-dom** package.
- * We will be using the latest version.



Steps to Install:



Installation Command:

Open your terminal and run the following command:

```
npm install react-router-dom
```



Verify Installation:

You can verify that the package is installed by checking the **package.json** file.

```
"dependencies": {  
  "react": "^18.x.x",  
  "react-dom": "^18.x.x",  
  "react-router-dom": "^6.x.x"  
}
```

Version number may vary. It will be the latest available version.



Importing React Router:

After installation, we can import the required components from **react-router-dom**.

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



Key Points:

- ✓ React Router is installed using the **react-router-dom** package.
- ✓ Always use the latest stable version for better performance and bug fixes.
- ✓ Ensure your project is created using Create React App or any React setup.
- ✓ After installation, import and start using the components for routing.



Key Takeaway:

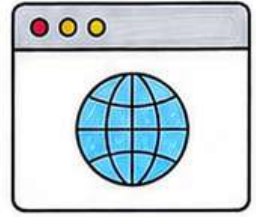


Installation of React Router is simple and fast. With a single command, we can add **powerful routing capabilities** to our React applications.



9.3 BrowserRouter

- * BrowserRouter is a router implementation that uses the **HTML5 History API** (pushState, replaceState and popstate) to keep the UI in sync with the URL.
- * It is the most commonly used router in React applications.
- * It enables **clean URLs** without the **#** symbol.
- * BrowserRouter should **wrap** the entire application so that all components can access routing features.



What is BrowserRouter?

BrowserRouter is a `<Router>` implementation that keeps your UI in sync with the URL using the **HTML5 History API**. It uses clean URLs and does not add any hash (**#**) in the URL.

Syntax :

```
<BrowserRouter>
  { /* All your routes and components */ }
</BrowserRouter>
```

Wrap your entire application inside **<BrowserRouter>** to enable routing.

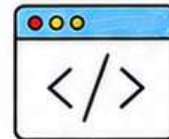


Example :

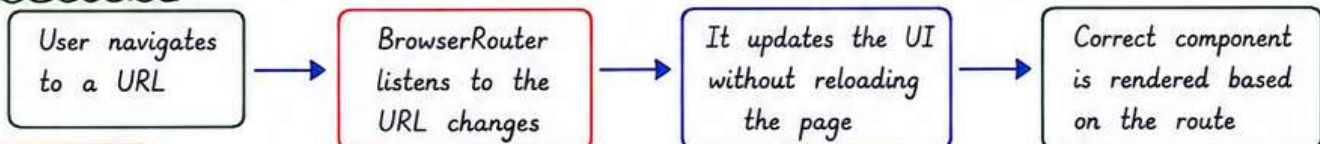
```
import React from 'react';
import { BrowserRouter } from 'react-router-dom';
import App from './App';

function Main() {
  return (
    <BrowserRouter>
      <App />
    </BrowserRouter>
  );
}
export default Main;
```

Here, **<BrowserRouter>** provides routing functionality to the entire application.



How it Works?



Key Points :

- ✓ BrowserRouter uses **HTML5 History API**.
- ✓ It provides clean and user-friendly URLs.
- ✓ It should wrap the whole application.
- ✓ It is ideal for production applications.



Key Takeaway :



BrowserRouter is the foundation of routing in React. It keeps the UI and URL in sync using the **HTML5 History API** with clean URLs.



9.4 Routes

- * In React Router v6 and above, `<Routes>` is used to define a set of routes.
- * It is the replacement for the old `<Switch>` component.
- * `<Routes>` looks through its child `<Route>` elements and renders the **best match**.
- * Only the first matching route is rendered.

What are Routes?

`<Routes>` is a component that holds one or more `<Route>` elements. Each `<Route>` element maps a path to a component.

Syntax :

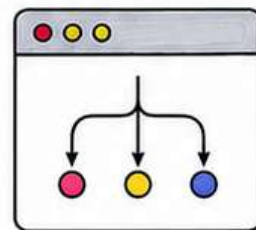
```
<Routes>
  <Route path="/path1" element={<Component1 />} />
  <Route path="/path2" element={<Component2 />} />
  <Route path="*" element={<NotFound />} />    {/* For 404 pages */}
</Routes>
```

Example :

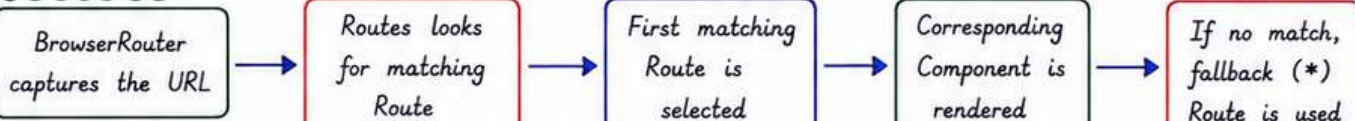
```
import { BrowserRouter } from 'react-router-dom';
import { Routes, Route } from 'react-router-dom';
import Home from './Home';
import About from './About';
import Contact from './Contact';
import NotFound from './NotFound';

function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="/contact" element={<Contact />} />
        <Route path="*" element={<NotFound />} />
      </Routes>
    </BrowserRouter>
  );
}
export default App;
```

`<Routes>` checks all the `<Route>` elements and renders the best matching component.



How it Works?



Key Points :

- ✓ `<Routes>` is used to define and group multiple `<Route>` elements.
- ✓ It renders only the first matching route.
- ✓ The `*` route is used to handle 404 - Page Not Found.
- ✓ The `element` prop is used to specify which component to render.

Key Takeaway :



`<Routes>` helps in managing multiple routes in a React application. It ensures only the best matching route is rendered, making routing efficient and clean.



9.5 Route

- * `<Route>` is used to define a **single route** in React Router.
- * It maps a **URL path** to a component.
- * When the URL **matches** the path, the component is rendered.
- * It is always used inside the `<Routes>` component.
- * It is the **core building block** of routing in React.



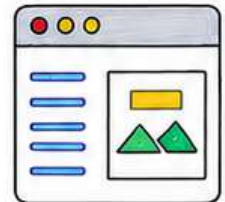
What is Route ?

`<Route>` is a component that defines a mapping between a URL path and a UI component.
When the current URL **matches** the path, the specified component is rendered.

Syntax :

```
<Route path="path" element={<Component />} />
```

path : The URL path to match
element : The component to render



Example :

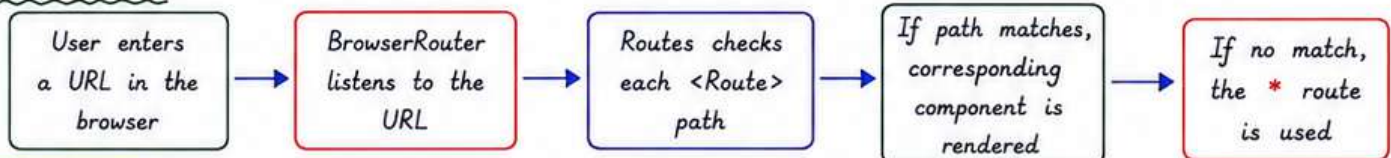
```
import { BrowserRouter } from 'react-router-dom';
import { Routes, Route } from 'react-router-dom';
import Home from './Home';
import About from './About';
import Contact from './Contact';
import NotFound from './NotFound';

function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="/contact" element={<Contact />} />
        <Route path="*" element={<NotFound />} />
      </Routes>
    </BrowserRouter>
  );
}
export default App;
```

Each `<Route>` defines one route. When the URL matches the **path**, the component in **element** is rendered.



How it Works?



Key Points :

- ✓ `<Route>` is used to map a path to a component.
- ✓ It is used inside the `<Routes>` component.
- ✓ Only the first matching `<Route>` is rendered.
- ✓ Use `*` path to handle 404 - Page Not Found.



Key Takeaway :



`<Route>` is the most important component in React Router.
It connects **URL paths** to components and makes navigation possible.



9.6 Link

- * `<Link>` is used to navigate between routes in a React application.
- * It is an alternative to the `<a>` tag.
- * It updates the URL **without reloading** the page.
- * It provides a **smooth** and **fast** navigation experience.
- * It is imported from `'react-router-dom'`.



What is Link ?

`<Link>` is a component that is used to create navigation links in React applications.
It is similar to `<a>` tag but prevents the page from reloading.

HOME ABOUT CONTACT

Syntax :

`<Link to="path">Link Text</Link>`

to : The path to navigate to
Link Text : The text to display



Example :

```
import { BrowserRouter } from 'react-router-dom';
import { Routes, Route, Link } from 'react-router-dom';
import Home from './Home';
import About from './About';
import Contact from './Contact';

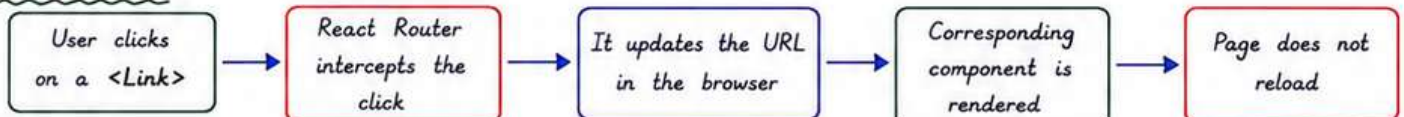
function App() {
  return (
    <BrowserRouter>
      <div style={{ padding: '20px' }}>
        <nav>
          <Link to="/">Home</Link> |
          <Link to="/about">About</Link> |
          <Link to="/contact">Contact</Link>
        </nav>

        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/about" element={<About />} />
          <Route path="/contact" element={<Contact />} />
        </Routes>
      </div>
    </BrowserRouter>
  );
}
export default App;
```

`<Link>` is used in the navigation menu to move between different routes without reloading the page.



How it Works?



Key Points :

- ✓ `<Link>` provides navigation without full page reload.
- ✓ It is imported from `'react-router-dom'`.
- ✓ Use the **to** prop to specify the path.
- ✓ It improves performance and user experience.



Key Takeaway :



`<Link>` is the recommended way to navigate between routes in React.
It keeps the application **fast**, **smooth** and **SPA-friendly**.



9.7 NavLink

- * `<NavLink>` is an enhanced version of `<Link>` provided by React Router.
- * It automatically applies an **active style** to the link of the current route.
- * It helps to **highlight** the **active link** in the navigation menu.
- * It accepts a **className** or **style** based on whether the link is **active**.
- * It is imported from `'react-router-dom'`.

HOME ABOUT CONTACT

What is NavLink ?

`<NavLink>` is a special component used to create navigation links. It knows when the link is **active** and applies styles or classes automatically. It is similar to `<Link>` but with extra **active** link features.

Syntax :

```
<NavLink to="path" className={({ isActive }) => isActive ? 'active' : ''}>  
  Link Text  
</NavLink>
```

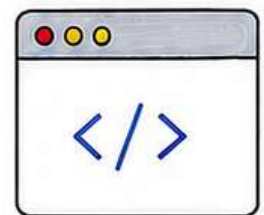


*isActive is true
when the link
matches the current
route.*

Example :

```
import { BrowserRouter } from 'react-router-dom';  
import { Routes, Route, NavLink } from 'react-router-dom';  
import Home from './Home';  
import About from './About';  
import Contact from './Contact';  
  
function App() {  
  return (  
    <BrowserRouter>  
      <div style={{ padding: '20px' }}>  
        <nav style={{ display: 'flex', gap: '20px', marginBottom: '20px' }}>  
          <NavLink to="/" className={({ isActive }) => isActive ? 'active' : ''}>Home</NavLink>  
          <NavLink to="/about" className={({ isActive }) => isActive ? 'active' : ''}>About</NavLink>  
          <NavLink to="/contact" className={({ isActive }) => isActive ? 'active' : ''}>Contact</NavLink>  
        </nav>  
  
        <Routes>  
          <Route path="/" element={<Home />} />  
          <Route path="/about" element={<About />} />  
          <Route path="/contact" element={<Contact />} />  
        </Routes>  
      </div>  
    </BrowserRouter>  
  );  
}  
export default App;
```

*`<NavLink>` adds
an **'active'** class
when the link is
active. Use this
class to apply
styles.*



How it Works ?

User clicks on a
`<NavLink>`

React Router
checks the
current route

If the route
matches the link,
isActive is true

Active class/style
is applied to the
link

The active link
is highlighted in
the navigation

Key Points :

- ✓ `<NavLink>` is used for navigation just like `<Link>`.
- ✓ It automatically knows when it is active.
- ✓ It helps to apply **active** styles or classes to the current link.
- ✓ Use the **isActive** prop to apply dynamic styles or class names.



Key Takeaway :

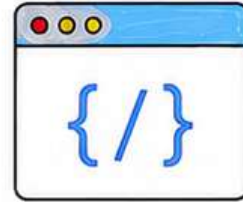


`<NavLink>` is the best way to create navigation menus in React applications. It improves user experience by highlighting the **active route** automatically.



9.8 Dynamic Routes

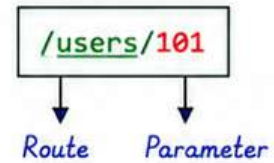
- * Dynamic Routes allow us to create routes with **parameters**.
- * They are useful when the URL contains **dynamic values**.
- * In React Router, dynamic segments are defined using a **colon (:)** followed by the parameter name.
- * These parameters can be accessed in the component using **useParams()**.



What are Dynamic Routes?

Dynamic Routes are routes that contain parameters in the URL. These parameters make the route flexible and reusable.

Example: `/users/:id` , `/products/:productId`



Syntax :

```
<Route path="/path/:paramName" element={<Component />} />
```

`:paramName` is a dynamic parameter.

Example :

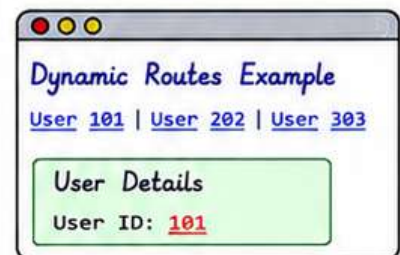
```
import { BrowserRouter } from 'react-router-dom';
import { Routes, Route, Link, useParams } from 'react-router-dom';

function User() {
  const { id } = useParams(); // get the dynamic parameter
  return (
    <div style={{ padding: '20px' }}>
      <h2>User Details</h2>
      <p>User ID: {id}</p>
    </div>
  );
}

function App() {
  return (
    <BrowserRouter>
      <div style={{ padding: '20px' }}>
        <h2>Dynamic Routes Example</h2>
        <nav>
          <Link to="/users/101">User 101</Link> |
          <Link to="/users/202">User 202</Link> |
          <Link to="/users/303">User 303</Link>
        </nav>
        <Routes>
          <Route path="/users/:id" element={<User />} />
        </Routes>
      </div>
    </BrowserRouter>
  );
}

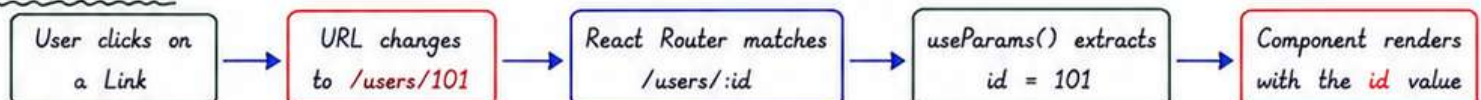
export default App;
```

`useParams()` is a hook that returns an object of all URL parameters. Here, we get the **id** from the URL.



When you click on User 101, the URL becomes `/users/101` and the **id** is shown.

How it Works?



Key Points :

- ✓ Dynamic routes are defined using **:parameterName**.
- ✓ Parameters are accessed using **useParams()** hook.
- ✓ They make routes flexible and reusable.
- ✓ Useful for details pages, profiles, products, blogs, etc.



Key Takeaway :



Dynamic Routes allow us to build **powerful and flexible** applications by passing values in the URL and using them inside components.



9.9 Route Parameters

- * Route Parameters are **values** passed in the URL to **identify resources**.
- * They are defined in the route path using a **colon (:)** followed by the parameter name.
- * These parameters can be accessed in the component using **useParams()**.
- * Useful for displaying details of a specific item like **user, product, post**, etc.

What are Route Parameters ?

Route Parameters are dynamic parts of the URL. They allow us to pass specific values in the URL and access them inside components.

Example: `/users/101` , `/products/55`

Syntax :

```
<Route path="/path/:paramName" element={<Component />} />
```

`:paramName` is the name of the parameter.

Example :

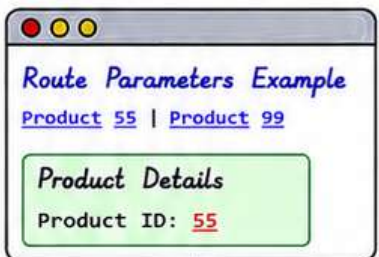
```
import { BrowserRouter } from 'react-router-dom';
import { Routes, Route, Link, useParams } from 'react-router-dom';

// Component to display Product Details
function Product() {
  const { id } = useParams(); // get the parameter
  return (
    <div style={{ padding: '20px' }}>
      <h2>Product Details</h2>
      <p>Product ID: {id}</p>
    </div>
  );
}

function App() {
  return (
    <BrowserRouter>
      <div style={{ padding: '20px' }}>
        <h2>Route Parameters Example</h2>
        <nav style={{ marginBottom: '20px' }}>
          <Link to="/products/55" style={{ marginRight: '10px' }}>Product 55</Link> |
          <Link to="/products/99">Product 99</Link>
        </nav>
        <Routes>
          <Route path="/products/:id" element={<Product />} />
        </Routes>
      </div>
    </BrowserRouter>
  );
}

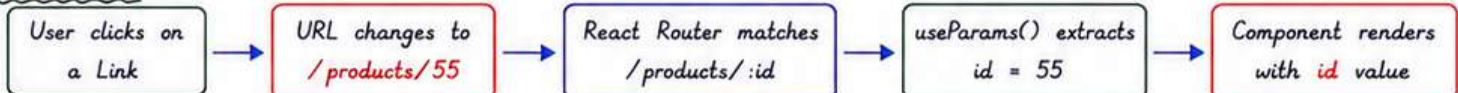
export default App;
```

`useParams()` returns an object containing all the parameters from the URL. Here, we get the **id** from the URL.



When you click on Product 55, the URL becomes `/products/55` and the **id = 55** is displayed.

How it Works ?



Key Points :

- ✓ Route Parameters are defined in the path using a **colon (:)**.
- ✓ Access parameters inside components using **useParams()** hook.
- ✓ Makes routes dynamic and reusable.
- ✓ Commonly used for details pages (users, products, posts, etc.).



Key Takeaway :



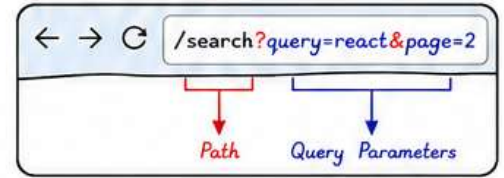
Route Parameters help us build dynamic and flexible applications. They allow us to pass values in the URL and display specific data based on those values.



9.10 Query Parameters

- * Query Parameters are **key-value pairs** added after the **?** in the URL.
- * They are used to pass **optional data** to the next page.
- * Multiple query parameters are separated by **&**.
- * They **do not** define routes, but provide additional information.
- * Access query parameters in React using **useSearchParams()** hook.
- * Example: **/search?query=react&page=2**

URL Example



What are Query Parameters ?

Query Parameters are appended to the URL after the **?**.
They are in the form of **key=value** pairs.
They help to send non-necessary data in the URL.
They are not part of the route path.



Query parameters are used for search, filtering, sorting, pagination, tracking, etc.

Syntax :

```
const [searchParams] = useSearchParams();  
// Returns a URLSearchParams object
```

searchParams is an object that contains all the query parameters.

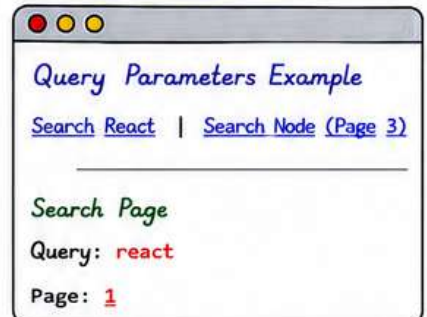
Example URLs

- /search?query=react
- /search?query=react&page=2
- /products?category=laptops&sort=price

Example :

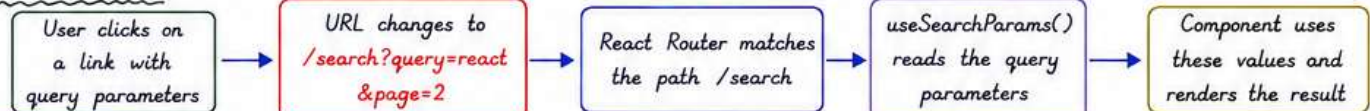
```
import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';  
import { useSearchParams } from 'react-router-dom';  
  
function Search() {  
  const [searchParams] = useSearchParams();  
  const query = searchParams.get('query'); // ?query=...  
  const page = searchParams.get('page'); // ?page=...  
  
  return (  
    <div style={{ padding: '20px' }}>  
      <h2>Search Page</h2>  
      <p><b>Query:</b> {query || 'None'}</p>  
      <p><b>Page:</b> {page || '1'}</p>  
    </div>  
  );  
}  
  
function App() {  
  return (  
    <BrowserRouter>  
      <div style={{ padding: '20px' }}>  
        <h2>Query Parameters Example</h2>  
        <nav>  
          <Link to="/search?query=react">Search React</Link> |  
          <Link to="/search?query=node&page=3">Search Node (Page 3)</Link>  
        </nav>  
        <hr />  
        <Routes>  
          <Route path="/search" element={<Search />} />  
        </Routes>  
      </div>  
    </BrowserRouter>  
  );  
}  
  
export default App;
```

useSearchParams() returns an object of type `URLSearchParams` which provides methods like `get()`, `has()`, `getAll()`, etc.



When you click "Search React", URL becomes **/search?query=react** and the values are read from the query string.

How it Works ?



Key Points :

- ✓ Query Parameters come after **?** in the URL.
- ✓ Multiple parameters are separated by **&**.
- ✓ They are accessed using **useSearchParams()** hook.
- ✓ They are useful for search, filter, pagination, sorting, etc.
- ✓ They do not affect route matching.



Key Takeaway :



Query Parameters allow you to pass extra, optional information in the URL.
They are easy to share, bookmark and very useful for dynamic applications.



9.11 Nested Routes

- * Nested Routes allow you to define routes **inside another route**.
- * They are used for creating **child routes** under a parent route.
- * The child route is rendered inside the parent component using **<Outlet />**.
- * Useful for building layouts like dashboards, settings pages, etc.
- * They keep the URL structure clean and organized.

What are Nested Routes ?

Nested Routes are routes defined within another route.
The parent route renders a layout and **<Outlet />**.
The child routes are rendered inside that outlet.

Example: **/dashboard/profile**

Syntax :

```
<Route path="parent" element={<ParentComponent />}>
  <Route path="child" element={<ChildComponent />} />
  <Route path="child2" element={<Child2Component />} />
</Route>
```

Example :

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

// Parent Layout Component
function Dashboard() {
  return (
    <div style={{ padding: '20px' }}>
      <h2>Dashboard</h2>
      <nav style={{ marginBottom: '20px' }}>
        <Link to="profile" style={{ marginRight: '15px' }}>Profile</Link> |
        <Link to="stats" style={{ marginRight: '15px' }}>Stats</Link> |
        <Link to="settings" >Settings</Link>
      </nav>
      <hr />
      { /* Outlet renders the matched child route */ }
      <Outlet />
    </div>
  );
}

// Child Components
const Profile = () => <h3>Profile Page</h3>;
const Stats = () => <h3>Stats Page</h3>;
const Settings = () => <h3>Settings Page</h3>;

function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/dashboard" element={<Dashboard />}>
          <Route path="profile" element={<Profile />} />
          <Route path="stats" element={<Stats />} />
          <Route path="settings" element={<Settings />} />
          { /* Index Route */ }
          <Route index element={<Profile />} />
        </Route>
        <Route path="*" element={<h3>Page Not Found</h3>} />
      </Routes>
    </BrowserRouter>
  );
}

export default App;
```

How it Works ?

User navigates to a URL

React Router matches the parent route (**/dashboard**)

Dashboard component is rendered (with **<Outlet />**)

React Router matches the child route (profile / stats / settings)

The child component is rendered inside **<Outlet />**.

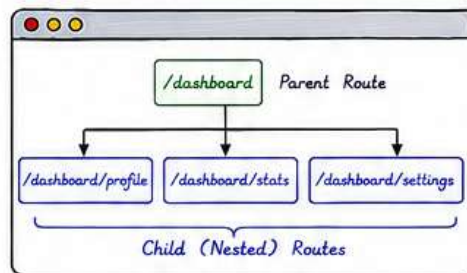
Key Points :

- ✓ Nested Routes are defined inside another **<Route>**.
- ✓ **<Outlet />** is used in the parent component to render child routes.
- ✓ They help in creating layouts with reusable components.
- ✓ Index route is used to render default child content.
- ✓ Improves code organization and user experience.

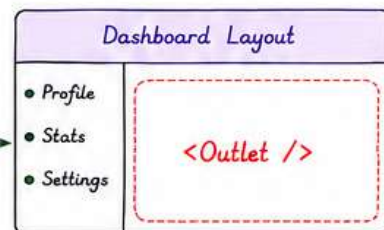
Key Takeaway :



Nested Routes make it easy to build complex UIs with layouts and sub-routes.
They allow child components to be displayed inside parent layouts using **<Outlet />**.



<Outlet /> is a placeholder where the matched child route component is rendered.



URL Structure

/dashboard } Parent
/dashboard/profile } Child Routes
/dashboard/stats }
/dashboard/settings }

How it looks

URL: **/dashboard**

Dashboard
Profile | Stats | Settings
Profile Page

URL: **/dashboard/stats**

Dashboard
Profile | Stats | Settings
Stats Page

URL: **/dashboard/settings**

Dashboard
Profile | Stats | Settings
Settings Page

When no child path is matched (i.e., **/dashboard**), the index route is rendered (Profile Page).



9.12 Protected Routes

- * Protected Routes **restrict** access to certain pages based on authentication.
- * If the user is **not authenticated**, redirect to **login** page.
- * If the user is **authenticated**, **allow** access to the requested page.
- * Implemented using a **wrapper component** (**PrivateRoute**).
- * Commonly used in dashboards, profile pages, settings, etc.

What is Protected Route ?

Protected Routes are routes that can be accessed only by authenticated users.
Unauthenticated users are redirected to the **login** page.
After login, users are redirected back to the page they initially tried to access.

Syntax :

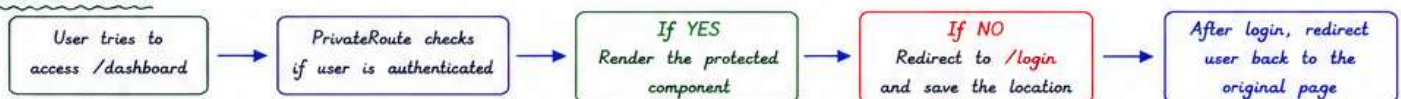
```
<Route path="/protected" element={<PrivateRoute>  
  <ProtectedComponent />  
</PrivateRoute>} />
```

PrivateRoute is a custom component that checks authentication and decides whether to allow access or not.

Example :

```
import { BrowserRouter, Routes, Route, Navigate, useLocation } from 'react-router-dom';  
import Login from './Login';  
import Dashboard from './Dashboard';  
import Profile from './Profile';  
import { useAuth } from './AuthContext'; // custom auth context  
  
// PrivateRoute Component  
const PrivateRoute = ({ children }) => {  
  const { isAuthenticated } = useAuth();  
  const location = useLocation();  
  
  if (!isAuthenticated) {  
    // Redirect to login, and save the current location  
    return <Navigate to="/login" state={{ from: location }} replace />;  
  }  
  return children;  
};  
  
function App() {  
  return (  
    <BrowserRouter>  
      <Routes>  
        <Route path="/login" element={<Login />} />  
        <Route  
          path="/dashboard"  
          element={  
            <PrivateRoute>  
              <Dashboard />  
            </PrivateRoute>  
          />  
        <Route  
          path="/profile"  
          element={  
            <PrivateRoute>  
              <Profile />  
            </PrivateRoute>  
          />  
        <Route path="/" element={<Navigate to="/dashboard" replace />} />  
      </Routes>  
    </BrowserRouter>  
  );  
}  
export default App;
```

How it Works ?



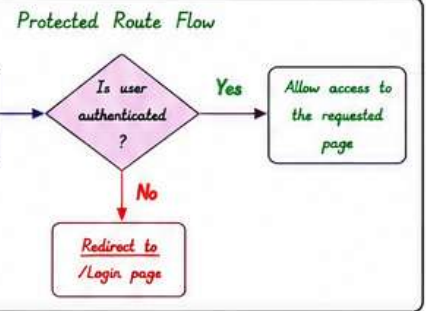
Key Points :

- ✓ Protected Routes improve security and user experience.
- ✓ Use a wrapper component (**PrivateRoute**) to protect routes.
- ✓ Unauthenticated users are redirected to **login** page.
- ✓ After login, users are redirected back to the page they originally requested.
- ✓ Useful for dashboards, profiles, settings, and other private pages.

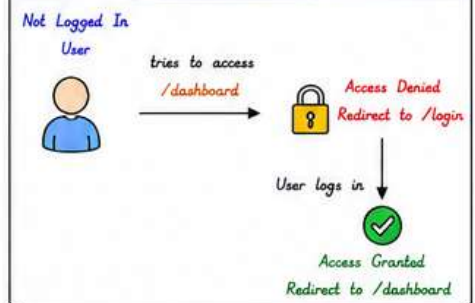
Key Takeaway :



Protected Routes ensure that only authenticated users can access sensitive pages. They help in building secure and professional applications.



Example Scenario



AuthContext (example)

```
import { createContext, useContext, useState } from 'react';  
const AuthContext = createContext();  
  
export const AuthProvider = ({ children }) => {  
  const [isAuthenticated, setIsAuthenticated] = useState(false);  
  return (  
    <AuthContext.Provider value={{ isAuthenticated, setIsAuthenticated }}>  
      {children}  
    </AuthContext.Provider>  
  );  
};  
export const useAuth = () => useContext(AuthContext);
```

- ✓ **useLocation()** is used to store the page the user tried to access.
- ✓ After successful login, redirect the user back to that page.

Login Component (Redirect back after login)

```
import { useLocation, useNavigate } from 'react-router-dom';  
import { useAuth } from './AuthContext';  
const Login = () => {  
  const { setIsAuthenticated } = useAuth();  
  const location = useLocation();  
  const navigate = useNavigate();  
  const from = location.state?.from?.pathname || '/dashboard';  
  
  const handleLogin = () => {  
    setIsAuthenticated(true);  
    navigate(from, { replace: true });  
  };  
  return (  
    <div>  
      <h2>Login Page</h2>  
      <button onClick={handleLogin}>Login</button>  
    </div>  
  );  
};
```



9.13 Redirects

- * Redirects are used to navigate users from **one route to another**.
- * It is useful when a route is **old, not valid**, or **access is restricted**.
- * In React Router, we use **<Navigate />** component to redirect.
- * It is also used after login, logout, or form submission.
- * It improves user experience and maintains correct navigation flow.

What are Redirects ?

Redirects allow us to send users from one URL (route) to another URL.

It helps in handling invalid routes, old routes, or after authentication actions.

Syntax :

```
<Navigate to="/path" replace={true | false} />
```

Navigate is a special component used to redirect to another route.

Example :

```
import { BrowserRouter, Routes, Route, Navigate } from 'react-router-dom';
import Login from './Login';
import Dashboard from './Dashboard';
import NotFound from './NotFound';

function App() {
  const isLoggedIn = false; // change to true after login

  return (
    <BrowserRouter>
      <Routes>
        <Route path="/login" element={<Login />} />
        <Route path="/dashboard" element={
          isLoggedIn ? <Dashboard /> : <Navigate to="/login" replace />
        } />

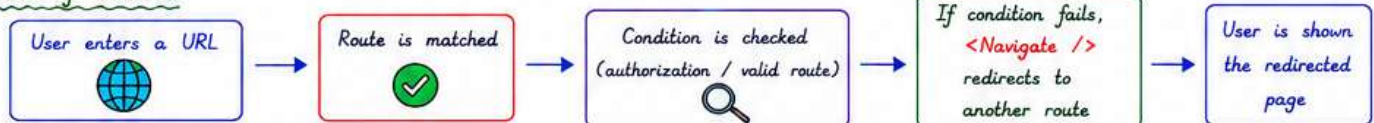
        /* Old route redirecting to new route */
        <Route path="/home" element={<Navigate to="/dashboard" replace />} />

        /* 404 Page for invalid routes */
        <Route path="*" element={<NotFound />} />
      </Routes>
    </BrowserRouter>
  );
}
export default App;
```

replace={true} vs replace={false}

- **replace={false}** (default): Adds the new page to history. User can go back to the previous page.
- **replace={true}**: Replaces the current page in history. User cannot go back to the previous page.

Flow of Redirect



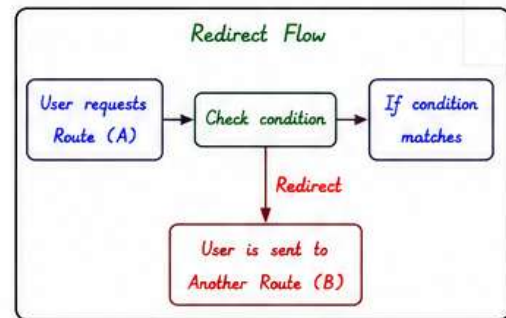
Key Points :

- ✓ Use **<Navigate />** component to redirect users.
- ✓ Redirects help in handling authentication, invalid routes and old routes.
- ✓ Use **replace={true}** to prevent users from going back to the restricted page.
- ✓ Redirects create a smooth and logical navigation experience.

Key Takeaway :



Redirects guide users to the correct destination and prevent them from accessing unwanted or invalid pages.

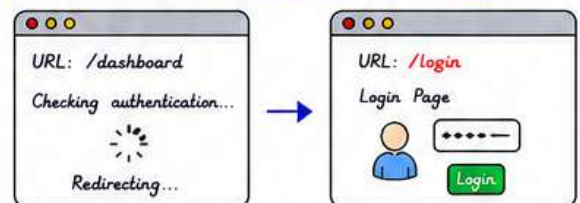


Common Use Cases

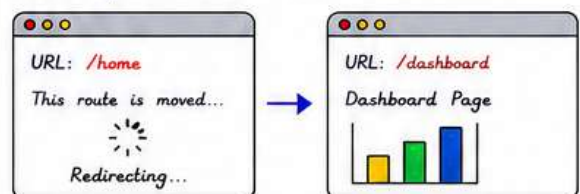
- Redirect after login
- Redirect after logout
- Redirect from old routes to new routes
- Handle invalid URLs (404 pages)
- Prevent unauthorized access

How it Works ?

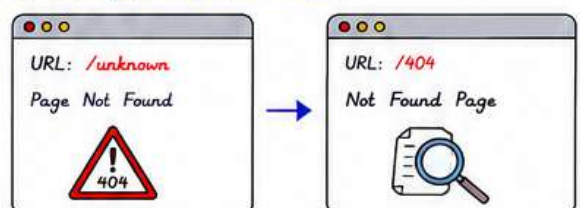
1. User tries to access /dashboard without login



2. User opens old route /home



3. User opens an invalid route



9.14 404 Pages

- * A 404 Page (Not Found Page) is displayed when a user tries to access a route that **does not exist**.
- * It improves **user experience** by showing a custom message instead of a blank page or error.
- * In React Router, 404 Pages are implemented using a **catch-all route (*)**.
- * It is usually the **last route** in the `<Routes>`.

What is 404 Page ?

A 404 Page is a page shown to the user when the requested URL does not match any defined route. It tells the user that the page does not exist.

Syntax :

```
<Route path="*" element={ <NotFound /> } />
```

Example :

```
import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';
import Home from './Home';
import About from './About';
import Contact from './Contact';
import NotFound from './NotFound';

function App() {
  return (
    <BrowserRouter>
      <nav style={{ padding: '15px', background: '#f0f0f0' }}>
        <Link to="/" style={{ marginRight: '15px' }}>Home</Link> |
        <Link to="/about" style={{ marginRight: '15px' }}>About</Link> |
        <Link to="/contact">Contact</Link>
      </nav>
      <hr />
      <Routes>
        <Route path="/" element={ <Home /> } />
        <Route path="/about" element={ <About /> } />
        <Route path="/contact" element={ <Contact /> } />
        { /* 404 Route - Must be the last route */ }
        <Route path="*" element={ <NotFound /> } />
      </Routes>
    </BrowserRouter>
  );
}
export default App;
```

NotFound.js

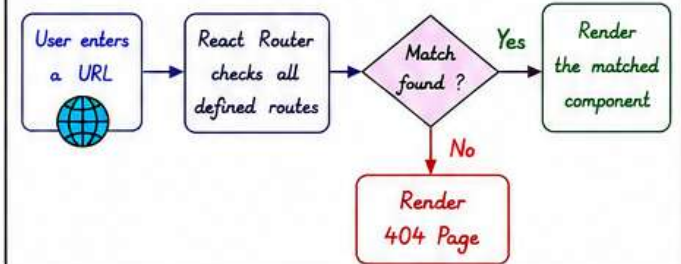
```
import { Link } from 'react-router-dom';

function NotFound() {
  return (
    <div style={{ textAlign: 'center', padding: '50px' }}>
      <h1 style={{ color: 'red', fontSize: '72px' }}>404</h1>
      <h2>Oops! Page Not Found</h2>
      <p>The page you are looking for does not exist.</p>
      <Link to="/" className="btn">Go to Home</Link>
    </div>
  );
}
export default NotFound;
```

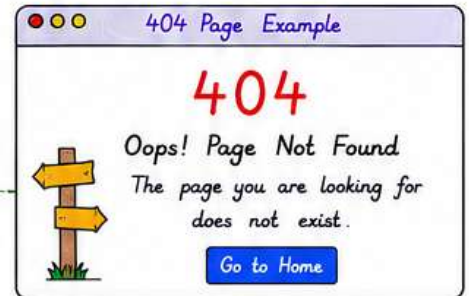
Home.js (example)

```
function Home() {
  return (
    <div style={{ padding: '30px' }}>
      <h2>Home Page</h2>
      <p>Welcome to our website!</p>
    </div>
  );
}
export default Home;
```

How 404 Page Works ?



The ***** (wildcard) matches any path that is not already matched by other routes.



When 404 Page is shown ?

- ✓ When user enters an invalid URL.
- ✓ When the URL does not match any defined route.
- ✓ When a user enters a wrong path (e.g., `/proucts` instead of `/products`).
- ✓ When a user tries to access a deleted or unavailable page.

Example URLs

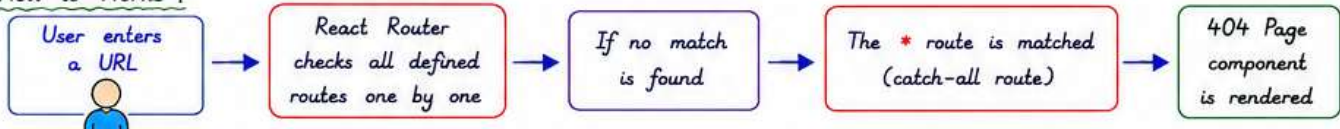
- `/unknown`
- `/about/team`
- `/products/123`
- `/any/random/path`



Output (Invalid URL)



How it Works ?



Key Points :

- ✓ Use `path="*"` to handle all unmatched routes.
- ✓ Always keep the 404 route as the **last route** in `<Routes>`.
- ✓ Create a user-friendly 404 page with options to go back or go to home.
- ✓ 404 Pages improve usability and look professional.



Key Takeaway :

- 404 Pages ensure users are informed when a page is not found and guide them back to the right place.



9.15 Layout Routes

- * Layout Routes are used to create **common UI** (header, footer, navbar) that is shared across multiple pages.
- * We create a layout component and render child routes inside it using `<Outlet />`.
- * This avoids repetition and keeps the code **clean and modular**.
- * Very useful for **dashboards**, **admin panels**, and **multi-page** applications.

What is Layout Route ?

A Layout Route is a route that wraps child routes and provides a shared UI.

It uses `<Outlet />` as a placeholder to render child components.

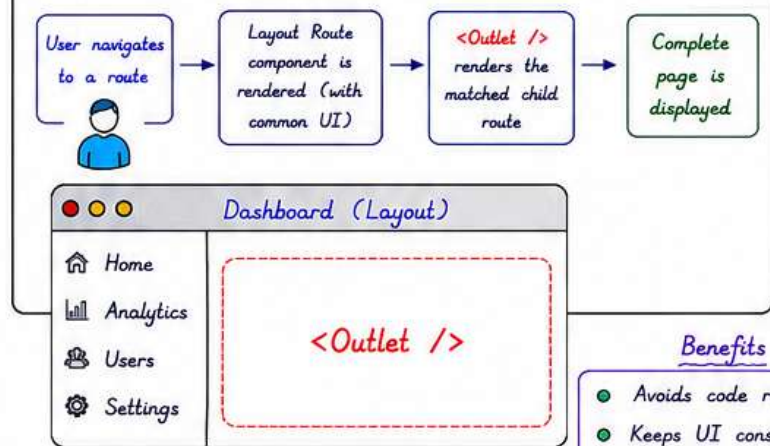
Syntax :

```
<Route path="/parent" element={<Layout />} >
  <Route path="child1" element={<Child1 />} />
  <Route path="child2" element={<Child2 />} />
</Route>
```

Example :

```
import { BrowserRouter, Routes, Route, Link, Outlet } from 'react-router-dom';
// Layout Component
function Layout() {
  return (
    <div>
      <header style={{ background: '#1976d2', color: 'white', padding: '15px' }}>
        <h2>My Dashboard</h2>
      </header>
      <div style={{ display: 'flex' }}>
        <nav style={{ width: '200px', background: '#f0f0f0', padding: '15px', minHeight: '80vh' }}>
          <ul style={{ listStyle: 'none', padding: 0 }}>
            <li style={{ margin: '10px 0' }}><Link to="/">Home</Link></li>
            <li style={{ margin: '10px 0' }}><Link to="/analytics">Analytics</Link></li>
            <li style={{ margin: '10px 0' }}><Link to="/users">Users</Link></li>
            <li style={{ margin: '10px 0' }}><Link to="/settings">Settings</Link></li>
          </ul>
        </nav>
        <main style={{ flex: 1, padding: '20px' }}>
          { /* Child routes will render here */ }
          <Outlet />
        </main>
      </div>
      <footer style={{ background: '#1976d2', color: 'white', textAlign: 'center', padding: '10px' }}>
        © 2025 My Dashboard
      </footer>
    </div>
  );
}
// Child Components
const Home = () => <h2>Welcome to Home Page</h2>;
const Analytics = () => <h2>Analytics Page Content</h2>;
const Users = () => <h2>Users Page Content</h2>;
const Settings = () => <h2>Settings Page Content</h2>;
function App() {
  return (
    <BrowserRouter>
      <Routes>
        { /* Layout Route (Parent) */ }
        <Route element={<Layout />}>
          <Route path="/" element={<Home />} />
          <Route path="/analytics" element={<Analytics />} />
          <Route path="/users" element={<Users />} />
          <Route path="/settings" element={<Settings />} />
        </Route>
        { /* 404 Page */ }
        <Route path="*" element={<h2>Page Not Found</h2>} />
      </Routes>
    </BrowserRouter>
  );
}
export default App;
```

How Layout Routes Work ?



Benefits

- Avoids code repetition.
- Keeps UI consistent.
- Easy to manage navigation.
- Improves code structure.
- Perfect for dashboards, admin panels, and apps with multiple pages.



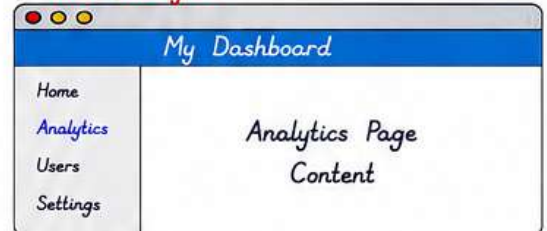
The parent route (Layout) renders its children inside `<Outlet />`.

URL Structure

<code>/</code>	→ Home (inside Layout)
<code>/analytics</code>	→ Analytics Page
<code>/users</code>	→ Users Page
<code>/settings</code>	→ Settings Page

How it Looks

URL: `/analytics`



URL: `/users`



How it Works ?



Key Takeaways :

- ✓ Layout Routes help create shared UI for multiple pages.
- ✓ They use `<Outlet />` to render child routes.
- ✓ Improves reusability, maintainability, and user experience.
- ✓ Essential for dashboards, admin panels, and complex apps.

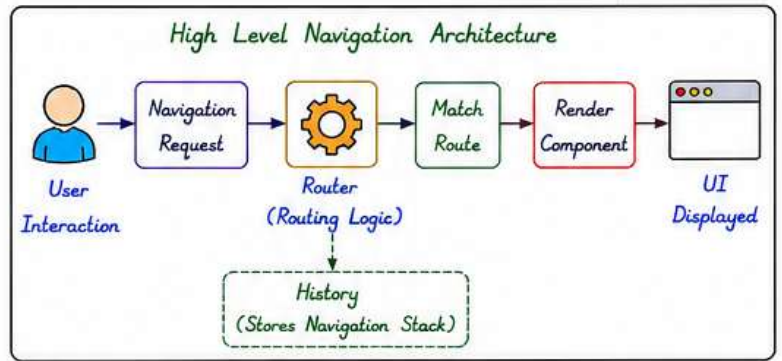


Use Layout Routes to build consistent and professional applications. Keep your components modular and your code clean!



9.16 Navigation Architecture

- * Navigation Architecture in React Router defines how the application handles **user navigation and route changes**.
- * It involves **components, routes, links, history**, and rendering working together.
- * Understanding this architecture helps in building **scalable** and **maintainable** React applications.
- * React Router uses a **declarative routing** approach.
- * The architecture is built around a few core concepts.



What is Navigation Architecture?

It is the structure that manages how users move between pages (routes) and how the corresponding components are rendered in the UI.

It ensures smooth navigation, proper state handling, and a great user experience.

Core Concepts

- **Routes** : Define mapping between URL and Component.
- **Link/NavLink** : Used to navigate between routes.
- **Router** : Core component that enables routing.
- **Outlet** : Placeholder to render matched child routes.
- **History** : Keeps track of navigation actions (back/forward).
- **Navigation** : Programmatic way to navigate.
- **Match** : Determines which route matches the current URL.



Good navigation architecture leads to **clean, fast, and user-friendly** apps.

Architecture Flow

1. User clicks a link or enters a URL in the browser.
2. The Router captures the navigation request.
3. The Router checks the route configuration.
4. It matches the URL with the defined routes.
5. If matched, the corresponding component is rendered.
6. History is updated to keep track of the navigation.
7. UI is updated and shown to the user.



Example Architecture in React Router

```
import { BrowserRouter, Routes, Route, Link, Outlet, useNavigate } from 'react-router-dom';

function App() {
  return (
    <BrowserRouter>
      <Layout /> { /* Parent Layout */ }
    </BrowserRouter>
  );
}

function Layout() {
  return (
    <div>
      <header style={{ background: '#1976d2', color: 'white', padding: '15px' }}>
        <h2>My App</h2>
        <nav>
          <Link to="/" style={{ marginRight: '15px', color: 'white' }}>Home</Link>
          <Link to="/about" style={{ marginRight: '15px', color: 'white' }}>About</Link>
          <Link to="/contact" style={{ color: 'white' }}>Contact</Link>
        </nav>
      </header>

      <main style={{ padding: '20px' }}>
        { /* Child routes render here */ }
        <Outlet />
      </main>

      <footer style={{ padding: '10px', background: '#f2f2f2', textAlign: 'center' }}>
        © 2025 My App
      </footer>
    </div>
  );
}

// Pages
const Home = () => <h2>Home Page</h2>;
const About = () => <h2>About Page</h2>;
const Contact = () => <h2>Contact Page</h2>;

function AppRoutes() {
  return (
    <Routes path="/" element={<Layout />}>
      <Route index element={<Home />} />
      <Route path="about" element={<About />} />
      <Route path="contact" element={<Contact />} />
    </Routes>
  );
}

export default AppRoutes;
```

Components of Architecture

1. **Router**
Provides routing context to the application.
2. **Routes**
Holds all the Route components.
3. **Route**
Defines a path and the component to render.
4. **Link / NavLink**
Used for navigation without page reload.
5. **Outlet**
Renders matched child route in parent layout.
6. **History**
Manages navigation state (back, forward).

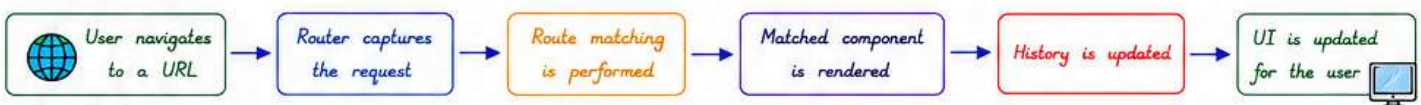


Navigation Types

- **Declarative Navigation** → Using Link and NavLink.
- **Programmatic Navigation** → Using useNavigate() hook.
- **Browser Navigation** → Back, Forward, Refresh.



How it Works?



Key Points :

- ✓ Navigation Architecture connects URLs to UI components.
- ✓ Router is the heart of the architecture.
- ✓ Routes define the structure of the application.
- ✓ Outlet enables nested routing and layout sharing.
- ✓ History manages all navigation actions.
- ✓ Good architecture improves scalability and maintainability.



Key Takeaway :

A strong Navigation Architecture ensures smooth navigation, clean code, and better user experience in React applications.



☆ Plan your routes, layouts, and navigation flow properly to build powerful and user-friendly React apps!