Routing in React

Engage and Think

You are a front-end developer at a growing e-commerce company. Customers have reported that navigating between pages causes the entire webpage to reload, resulting in slow load times and a poor user experience. Additionally, users lose their shopping cart data while switching between products pages.

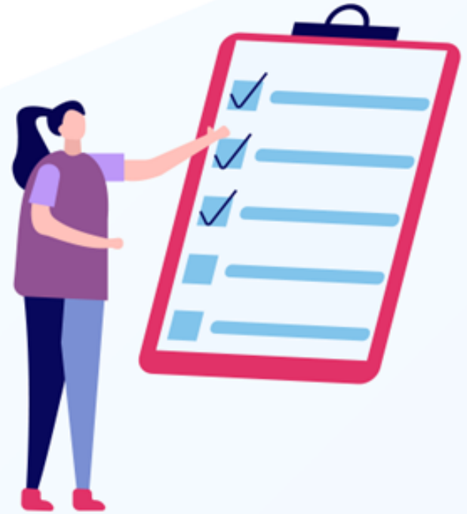
Your manager asks you to find a solution that allows seamless navigation without full-page reloads while maintaining state between pages.

How would you approach solving this issue to improve the user experience?

Learning Objectives

By the end of this lesson, you will be able to:

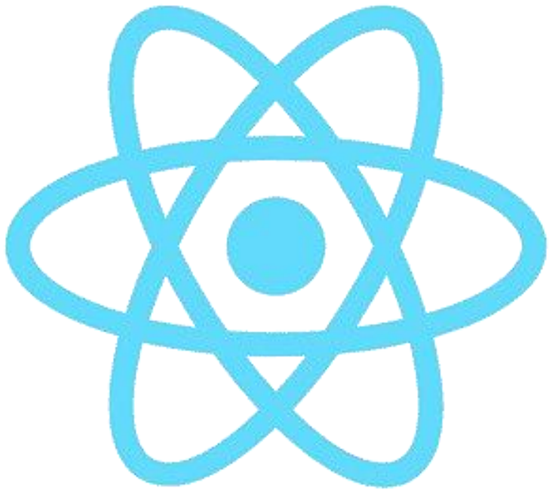
- Implement Route Guards for enhanced security and controlled access in React routes
- Demonstrate the steps in React routing for seamless client-side navigation in a single-page application
- Comprehend nested routes for organizing hierarchical user interfaces and managing complex application structures in React
- Work on the operation of passing and extracting parameters in React Route URLs for dynamic content and personalized user experiences



Introduction to Routing

What Is Routing in React?

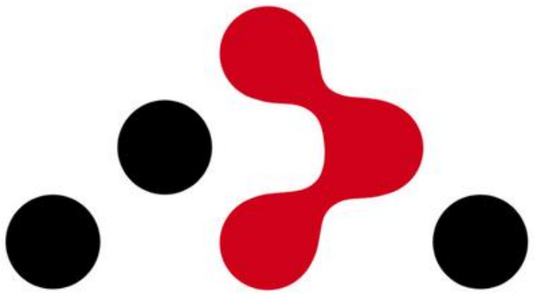
It refers to the process of navigating between different components or views in a React application based on the URL path in the browser.



It serves the foundation for navigation in React applications. React Router enables this navigation by managing dynamic and seamless transitions between views.

What Is React Router?

It is a powerful navigation library designed for building single-page applications (SPAs) in React.



- It allows users to interact with different parts of a web application’s interface and enables dynamic content updates without a full-page reload.
- It efficiently manages transitions by handling URL changes seamlessly within a React application.

Benefits of React Router

React Router offers the following key benefits for modern web applications:

- Enables single-page application behavior
- Promotes organized code structure
- Provides precise navigation control
- Enhances overall user experience

Key Components of React Routing

The following are the three main categories of components in React Router:

Routers

These components provide routing context and manage history (for example, <BrowserRouter> and <HashRouter>).

Route Matchers

These components match URLs and render views (for example, <Route>, <Switch>, and <Redirect>).

Navigation (or Route Changers)

These components enable navigation between routes (for example, <Link>, <NavLink>, and <Navigate>).

Setting Up React Router

To enable client-side routing in a React application, first install the react-router-dom package using npm, as shown below:

Step 1: Install React Router

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

This provides the necessary components for routing.

Setting Up React Router

After installing react-router-dom, the next step is to enable routing by wrapping the application with BrowserRouter to manage navigation within the React component tree, as shown below:

Step 2: Set up BrowserRouter

```
import { BrowserRouter as Router } from 'react-
router-dom’;
ReactDOM.render( <Router> <App or> <orRouter>,
document.getElementById('root') );
```

Setting Up React Router

To define navigation paths in a React application, use the Route component to map specific URLs to their corresponding components, as shown below:

Step 3: Define Routes with Route component

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

const App = () => {
  return (
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/about" element={<About />} />
      {/* Add more routes as needed */}
    </Routes>
  );
};
```

Setting Up React Router

To enable navigation between different routes in a React application without a full page reload, use the Link component instead of traditional anchor <a> tags, as shown below:

Step 4: Enable navigation with Link

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

const Navigation = () => {
  return (
    <nav>
      <Link to="/">Home</Link>
      <Link to="/about">About</Link>
      {/* Add more navigation links as needed */}
    </nav>
  );
};
```

Setting Up React Router

In React Router, the Switch component ensures that only the first matching route is rendered. In React Router v6, it has been replaced with Routes for improved routing, as shown below:

Step 5: Implement Route structure

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

const App = () => {
  return (
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/about" element={<About />} />
      <Route path="/contact" element={<Contact />} />
    </Routes>
  );
};
```

Setting Up React Router

In React Router, dynamic route parameters allow capturing values from the URL, enabling the creation of user-specific or data-driven routes, as shown below:

Step 6: Handle dynamic Route parameters

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

const UserProfile = () => {
  const { id } = useParams();
  return <div>User ID: {id}</div>;
};
const App = () => (
  <BrowserRouter>
    <Routes>
      <Route path="/user/:id" element={<UserProfile />} />
    </Routes>
  </BrowserRouter>
);
```

Assisted Practice

Demonstrating Basic Routing in a React Application **Duration: 15 Min.**

Problem statement:
You have been asked to implement basic client-side navigation in a React application. This includes creating routes using React Router, rendering different components based on the route, and verifying navigation using a development server.

Outcome:
By the end of this task, you will be able to create and configure routes in a React app using React Router, navigate between components such as Home and About, and run the application using Vite to verify routing functionality.

Note: Refer to the demo document for detailed steps:
01_Demonstrating_Basic_Routing_in_a_React_Application

Assisted Practice: Guidelines

Steps to be followed:

1. Set up a new React project using Vite
2. Create React components
3. Import and use the components
4. Run and verify the application

Quick Check

You are building a React app with Home, About, and Contact pages. When navigating to */about*, both the Home and About components render.

How can you fix this issue?

A. Use <Switch> instead of <Routes>

B. Add *exact* to the Home route

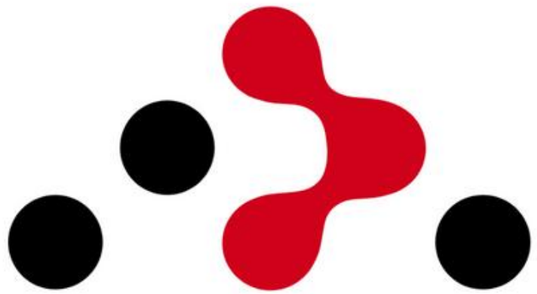
C. Replace Link with NavLink

D. Check window.location manually

Nested Routing

Nested Route: Overview

A route nested within another route forms a parent-child relationship. This structure enables hierarchical navigation, where child components render inside a shared parent layout.



It organizes paths by grouping related views, enabling seamless transitions within a shared container for better navigation management.

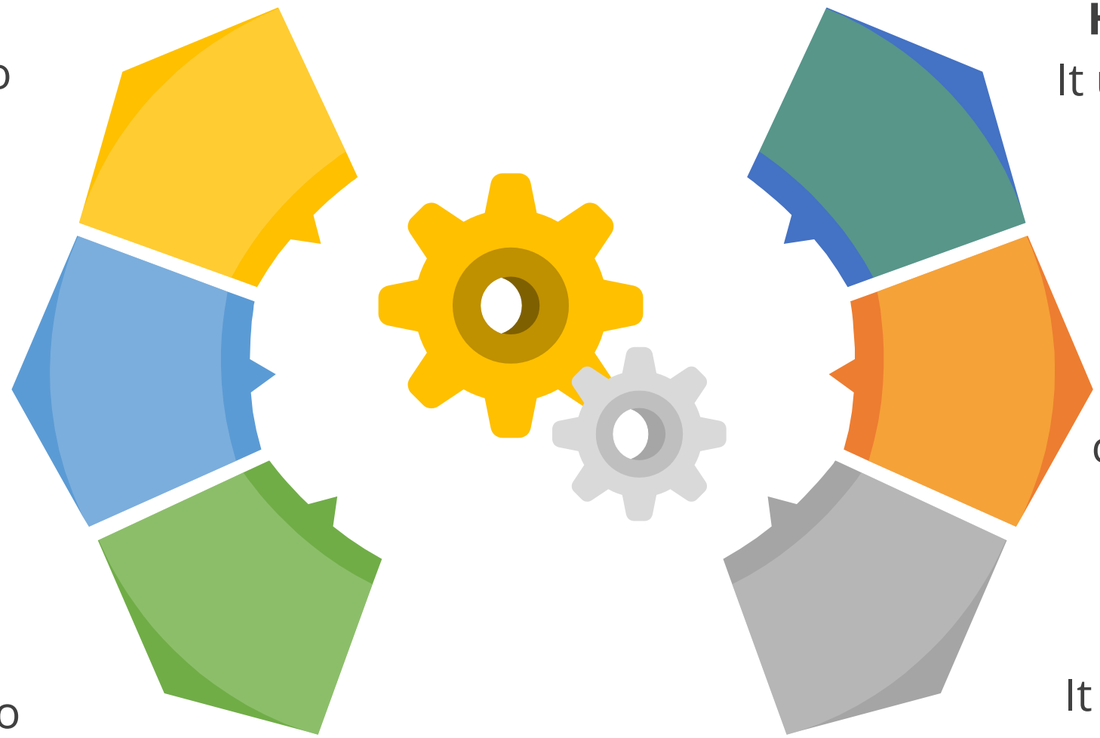
Nested Route: Key Concepts

The following are the key concepts of nested routes:

Dynamic segments
They use parameters to define variable paths.

Dynamic loading
It renders components based on the route.

Code organization
It groups related views to improve maintainability.



Hierarchical structure
It uses a nested format for structured navigation.

Parent-child components
They enable child components to inherit context from the parent.

Path concatenation
It automatically appends child paths to parents.

Assisted Practice

Demonstrating Nested Routing in a React Application **Duration: 15 Min.**

Problem statement:
You have been asked to create a structured and hierarchical navigation system in a React application using nested routes. This involves separating the layout and content components and using React Router to control route-based rendering.

Outcome:
By the end of this task, you will be able to implement nested routing in a React application, organize components into structured folders, and manage navigation between routes like Home and Dashboard using React Router and Vite.

Note: Refer to the demo document for detailed steps:
02_Demonstrating_Nested_Routing_in_a_React_Application

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Set up a new React project using Vite
2. Create React components
3. Import and use the components
4. Run and verify the application

Quick Check

A company wants to build a web application where the dashboard has multiple sections like overview, reports, and settings, all under a common layout. They need a way to structure these sections efficiently while keeping the header and sidebar consistent.

What is the best approach?

A. Create separate pages for each section

B. Use a single page and toggle visibility

C. Reload the page when switching sections

D. Use standalone routes for each section

Route Parameters

What Are Route Parameters?

They are dynamic segments in a URL that allow passing values to route handlers, enabling retrieval of specific data.

They are commonly used in web frameworks to capture and process variable parts of a URL, such as user IDs and product categories.

Route Parameters: Steps

The first step in using route parameters is to define a route with a dynamic parameter, as shown below:

Step 1: Define a Route with parameters

```
<Route path="/user/:id" component={UserProfile} />
```

This allows components to capture and use values from the URL dynamically.

Route Parameters: Steps

The second step is to create a Link that includes route parameters, allowing users to navigate dynamically to parameterized routes, as shown below:

Step 2: Link with parameters

```
<Link to="/user/123">User Profile</Link>
```

Route Parameters: Steps

The third step is to extract route parameters within a component, allowing dynamic content rendering based on values from the URL, as shown below:

Step 3: Access parameters in a component

```
import { useParams } from "react-router-dom";

const UserProfile = () => {
    const { id } = useParams();
    return <div>User ID: {id}</div>;
};
```

Route Parameters: Example

After defining routes, linking parameters, and extracting values, the following example shows a complete implementation using React Router:

Complete example of passing and extracting parameters

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

const UserProfile = () => <div>User ID: {useParams().id}</div>;

const App = () => (
    <Router>
        <nav><Link to="/user/123">User Profile</Link></nav>
        <Routes><Route path="/user/:id" element={<UserProfile />}
/></Routes>
    </Router>
);
```

Assisted Practice

Implementing Route Parameters in a React Application **Duration: 15 Min.**

Problem statement:
You have been asked to implement route parameters in a React application. This requires setting up dynamic routes using React Router so components can read and display route-specific data such as user IDs.

Outcome:
By the end of this task, you will be able to use React Router to define and consume route parameters, dynamically render content based on the route (for example., /user/:userId), and simulate user navigation with dynamic URLs in a Vite-based React project.

Note: Refer to the demo document for detailed steps:
03_Implementing_Route_Parameters_in_a_React_Application

Assisted Practice: Guidelines

Steps to be followed:

1. Set up a new React project using Vite
2. Create React components
3. Import and use the components
4. Run and verify the application

Quick Check

An online bookstore wants to display details of a book when a user clicks on a book. Instead of creating separate pages for each book, they prefer dynamic URL like /books/:id.

What feature should they use to achieve this?

A. Static routes

B. Route parameters

C. Query strings

D. Page refreshes

Programmatic Navigation

What Is Programmatic Navigation?

It allows applications to redirect users dynamically through code instead of manual clicks, enhancing control over navigation flow.

It is commonly used in Single Page Applications (SPAs) for authentication redirects, form submissions, and conditional routing based on user actions or system logic.

How Programmatic Navigation Works with SPAs?

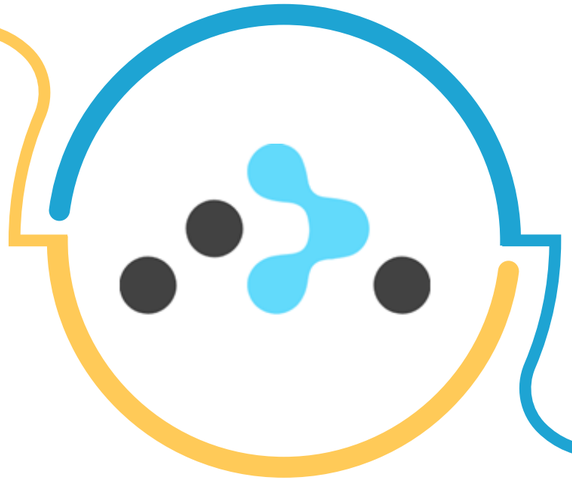
Programmatic navigation in SPAs functions by modifying the URL and loading new content dynamically, eliminating the need for a full-page reload.

This allows a seamless user experience similar to traditional multi-page apps without reloading the page.

Implementing Programmatic Navigation

React Router provides hooks to programmatically navigate and access route information within an application. The following are the commonly used hooks:

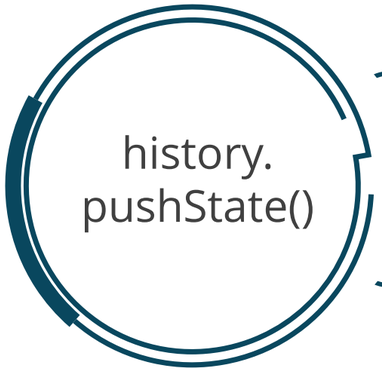
useNavigate
Redirects users programmatically



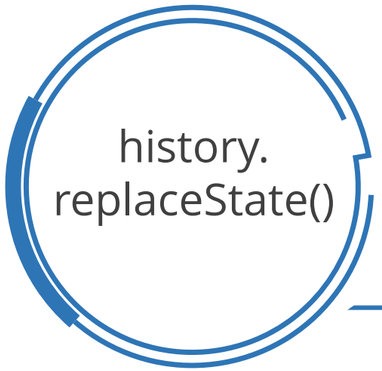
useLocation
Accesses the current route’s details

Programmatic Navigation in Browser

It is achieved by interacting with the browser's History API in the following ways:

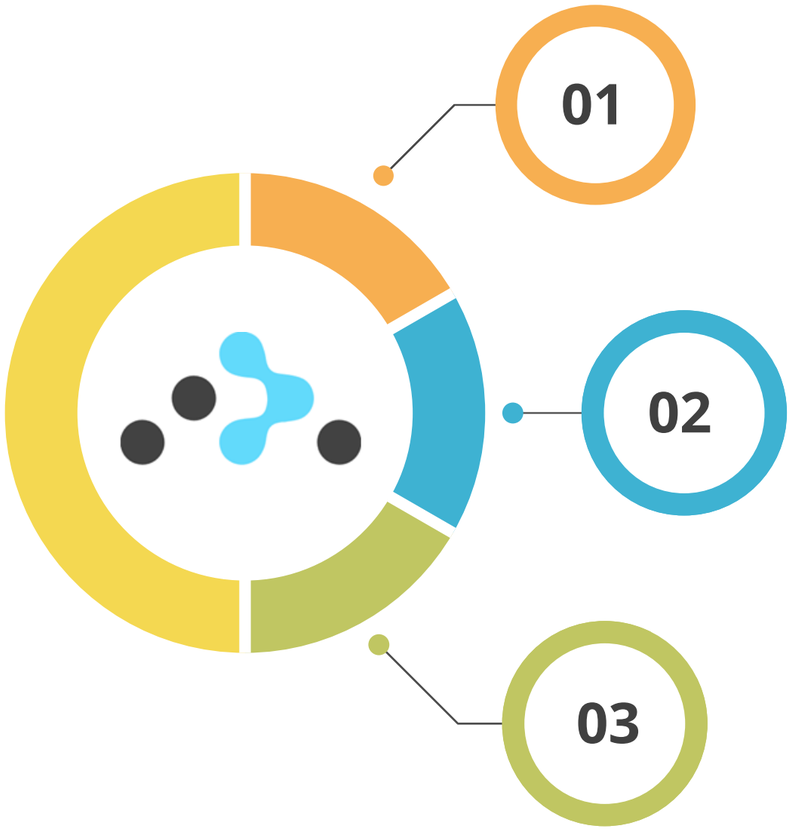


Adds a new entry to the browser's history stack, updating the URL without reloading the page



Modifies the current history entry, replacing it with the provided state object and URL without reloading the page

Programmatic Navigation: Use Cases



User redirection
Navigates users after actions such as form submissions, login, or logout

Route protection
Restricts access based on authentication or permissions

Dynamic routing
Adjusts navigation based on user interactions or app state

Assisted Practice

Demonstrating Programmatic Navigation in a React Application **Duration: 15 Min.**

Problem statement:
You have been asked to implement programmatic navigation in a React application. This includes redirecting users to specific routes using JavaScript logic rather than navigation links.

Outcome:
By the end of this task, you will be able to programmatically navigate users between routes using useNavigate() from React Router, triggered by user actions, such as button clicks, in a Vite-powered React application.

Note: Refer to the demo document for detailed steps:
04_Demonstrating_Programmatic_Navigation_in_a_React_Application

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Set up a new React project using Vite
2. Create React components
3. Import and use the components
4. Run and verify the application

Quick Check

You are developing a single-page-application (SPA) for a news website. Users should be able to navigate between different news categories (for example, sports, politics, and technology) without experiencing a full-page reload. The navigation should update the URL dynamically while displaying the relevant content instantly.

Which method should you use to achieve this seamless navigation?

A. Programmatic navigation

B. Static HTML pages

C. Server-side redirects

D. Full-page reload

Route Guard

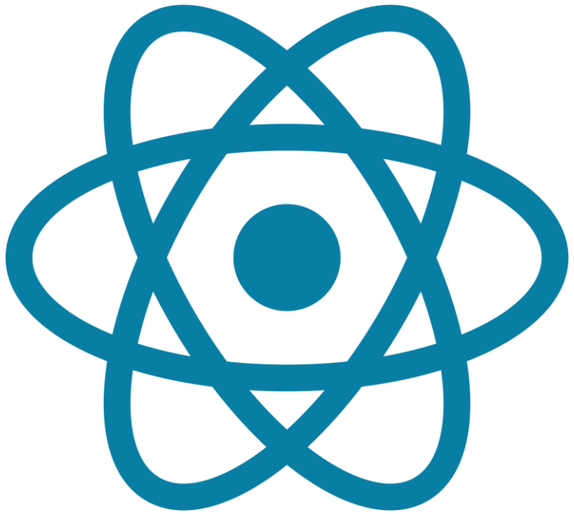
What Is Route Guard?

It is a security mechanism in web applications that controls access to specific routes by checking conditions such as authentication or user roles.

It intercepts navigation requests and either allows or restricts access based on predefined criteria, ensuring secure and controlled routing.

How Route Guards Works With SPAs?

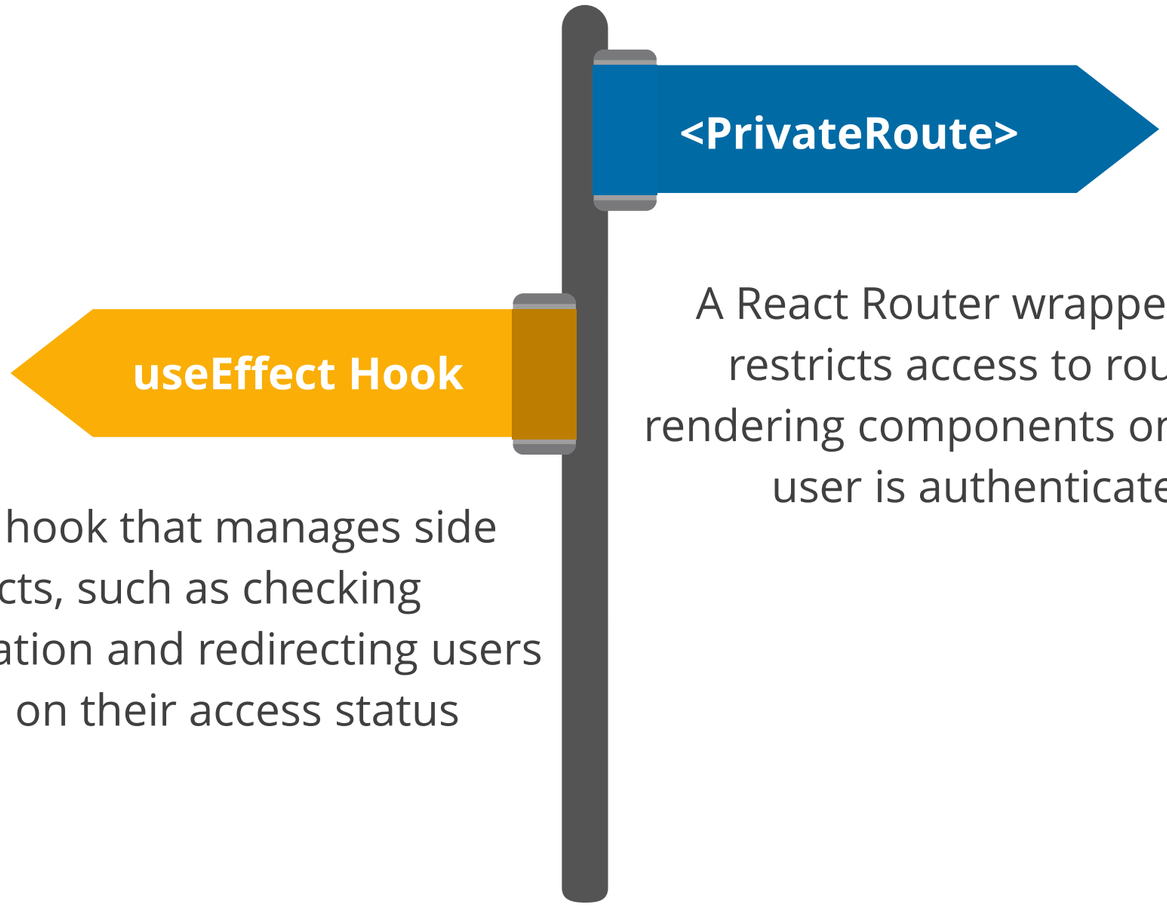
In SPAs, routing is managed on the client side, enabling smooth navigation without server requests.



Route guards intercept routing events programmatically to control access based on authentication, permissions, or other conditions.

Implementing Route Guards

In React, route guards can be implemented using the following methods to enforce authentication and authorization, ensuring secure and seamless navigation:



Route Guards: Use Cases

The following use cases highlight the importance of route guards:

Blocking unauthenticated users from accessing restricted routes

Preventing navigation when there are unsaved changes or pending actions

Preloading essential data before activating the destination route

404 Error Handling

What Is 404 Error?

It is an HTTP status code that indicates the requested page or resource could not be found on the server.

It typically appears when a URL is incorrect or the resource no longer exists.

Custom 404 Error Page Using React Router

In React Router, a custom 404 page can be created by defining a **<Route>** with a path of ***** or **404**.

This specific route is generally placed at the end of all route definitions, serving as a catch-all for unrecognized URLs.

Importance of Custom 404 Error Page

01 **Enhancing user experience:** Provides a consistent and user-friendly experience compared to default error messages

02 **Improving navigation flow:** Guides users to return to relevant sections through helpful links and directions

03 **Reinforcing brand identity:** Maintains brand consistency and creates a positive impression, even during errors

Importance of Custom 404 Error Page

04 **Enabling user feedback:** Allows users to report broken links, supporting site maintenance and error tracking

05 **Maintaining SEO health:** Prevents search engines from indexing non-existent pages to protect SEO performance

06 **Encouraging creative engagement:** Offers space for humor or creativity, making the site more memorable and shareable

Creating a Custom 404 Page with React Router

Follow these simple steps with code examples to create a custom 404 page in React using React Router:

Step 1: Create the 404 component

```
import React from 'react';

const NotFoundPage = () => {
  return (
    <div>
      <h1>404 Not Found</h1>
      <p>The page you are looking for doesn't exist.</p>
    </div>
  );
};

export default NotFoundPage;
```

Creating a Custom 404 Page with React Router

Step 2: Set up React Router

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

Creating a Custom 404 Page with React Router

Step 3: Define Routes in your application

```
import React from 'react';
import { BrowserRouter as Router, Route, Switch } from 'react-
router-dom';
import HomePage from './HomePage';
import AboutPage from './AboutPage';
import NotFoundPage from './NotFoundPage';

const App = () => {
  return (
    <Router>
      <Switch>
        <Route exact path="/" component={HomePage} />
        <Route path="/about" component={AboutPage} />

        {/* Catch-all route for unmatched paths */}
        <Route component={NotFoundPage} />
      </Switch>
    </Router>
  );
};

export default App;
```

Creating a Custom 404 Page with React Router

Step 4: Test your 404 page

```
npm run dev
```

This configuration displays a custom 404 page for undefined routes in your React application, improving user experience and gracefully handling navigation errors.

Assisted Practice

Implementing Protected Routes in a React Application **Duration: 15 Min.**

Problem statement:
You have been asked to restrict access to certain routes in a React application based on user authentication. This involves setting up protected routes using a custom component and React Router to manage secure navigation.

Outcome:
By the end of this task, you will be able to configure protected routes in a React app. This includes setting conditional access using a flag (isSignedIn) and redirecting unauthorized users to the homepage.

Note: Refer to the demo document for detailed steps:
05_Implementing_Protected_Routes_in_a_React_Application

Assisted Practice: Guidelines

Steps to be followed:

1. Set up a new React project using Vite
2. Create React components
3. Import and use the components
4. Run and verify the application

Quick Check

Your team is developing a web application with role-based access control (RBAC). A user without authentication tries to access a restricted route, but instead of seeing the expected page, they receive a 404 error.

What is the likely reason for this error?

A. Non-existent route

B. Unauthorized access

C. Data not loaded

D. Unsaved changes

Route Animation

What Are Route Animations?

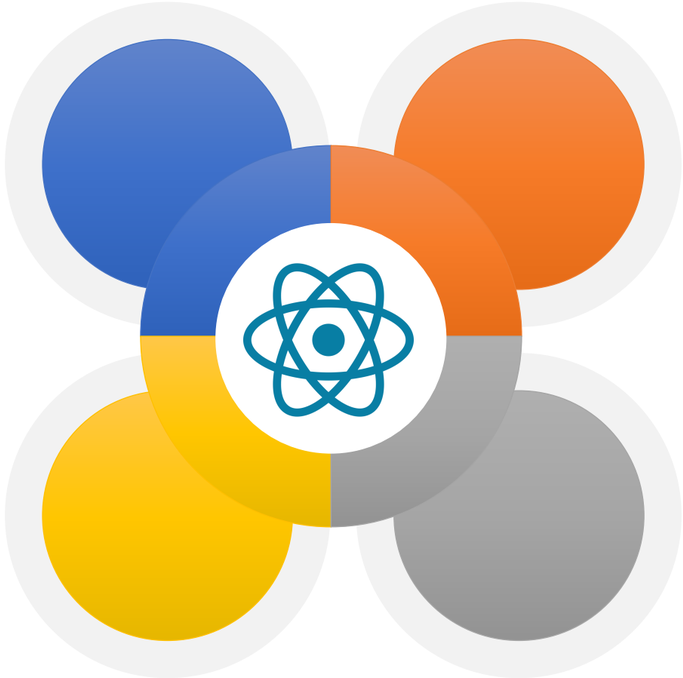
They are visual effects that occur during transitions between routes in a SPA, creating a dynamic shift instead of a static page reload.

These animations enable a smooth visual flow during navigation, enhancing interactivity and delivering a more polished user experience.

How to Implement Route Animations?

To implement animations in React, external libraries are typically used. Here are some popular options for adding route transitions:

React Spring
Creates smooth, physics-based animations in React



CSS transitions
Adds simple style transitions using CSS

Framer motion
Enables advanced and intuitive animations in React

React transition group
Manages enter and exit animations for components

Route Animation: Example

The following example demonstrates how to use React transition group for implementing route transition animations:

```
import { Routes, Route, useLocation } from 'react-router-dom';
import { CSSTransition, TransitionGroup } from 'react-transition-group';
import Home from './Home';
import About from './About';

export default function AnimatedRoutes() {
  const location = useLocation();

  return (
    <TransitionGroup>
      <CSSTransition key={location.key} classNames="fade" timeout={300}>
        <Routes location={location}>
          <Route path="/" element={<Home />} />
          <Route path="/about" element={<About />} />
        </Routes>
      </CSSTransition>
    </TransitionGroup>
  );
}
```

Implementing Advanced React Routing Features

Duration: 25 Mins.

Project agenda: To develop a React application with advanced routing capabilities using Vite, focusing on nested routes, dynamic URL parameters, custom error handling, and component-level styling using Styled Components. The project will ensure structured navigation, improved user experience, and reliable access management.

Description: You are tasked with building a React application that integrates advanced routing features to optimize navigation and access management. This includes implementing nested routes, dynamic URL parameters, custom error handling with a 404 page, and component-level styling using Styled Components.

LESSON-END PROJECT

Implementing Advanced React Routing Features

Duration: 25 Min.

Perform the following:

1. Set up a new React project using Vite
2. Structure the application
3. Implement routing features
4. Set up context for user management
5. Set up routing in App.jsx
6. Update main.jsx
7. Run and verify the application

Expected deliverables: A fully functional React-based application built with Vite, incorporating advanced routing capabilities. The application will support structured navigation using nested routes, dynamic route parameters for personalized content, and programmatic navigation for seamless redirection. It will include a custom 404 error page for handling invalid routes and component-level styling using Styled Components to enhance the user experience.

LESSON-END PROJECT

Key Takeaways

- React Router enhances application navigation by structuring code more effectively, giving developers better control over the navigation flow.
- It is vital for enabling smooth navigation and URL management in single-page React applications, enhancing user experience.
- It supports dynamic route parameters, allowing for flexible, data-driven components based on URL changes.
- Implementing route guards is crucial for controlling access based on criteria like authentication and ensuring secure and appropriate user access to routes.
- Creating custom 404 error pages in React Router improves user experience and maintains brand consistency during navigation errors.

Thank You