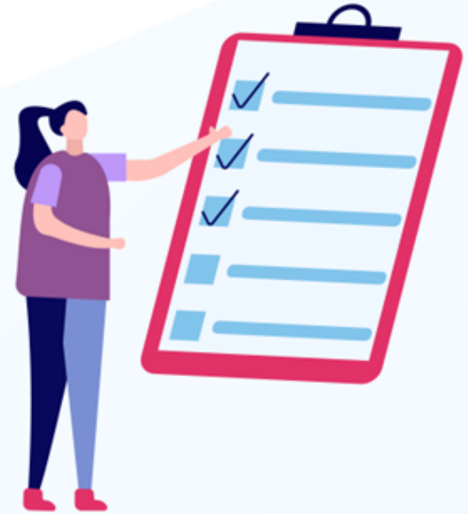
Design a Dynamic Frontend with React

Routing in React

Learning Objectives

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

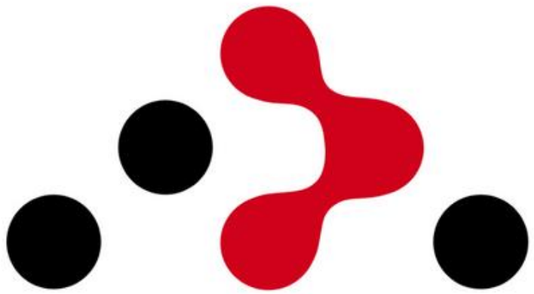
- Demonstrate the steps in React routing for seamless client-side navigation in a single page application
- Access a comprehensive guide that explains how to use React Router v6 in-depth
- Get an insight of the updates in React Router v6, delving into its new features and improvements
- Comprehend nested routes for organizing hierarchical user interfaces and managing complex application structures in React



Introduction to Routing

What Is React Routing?

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



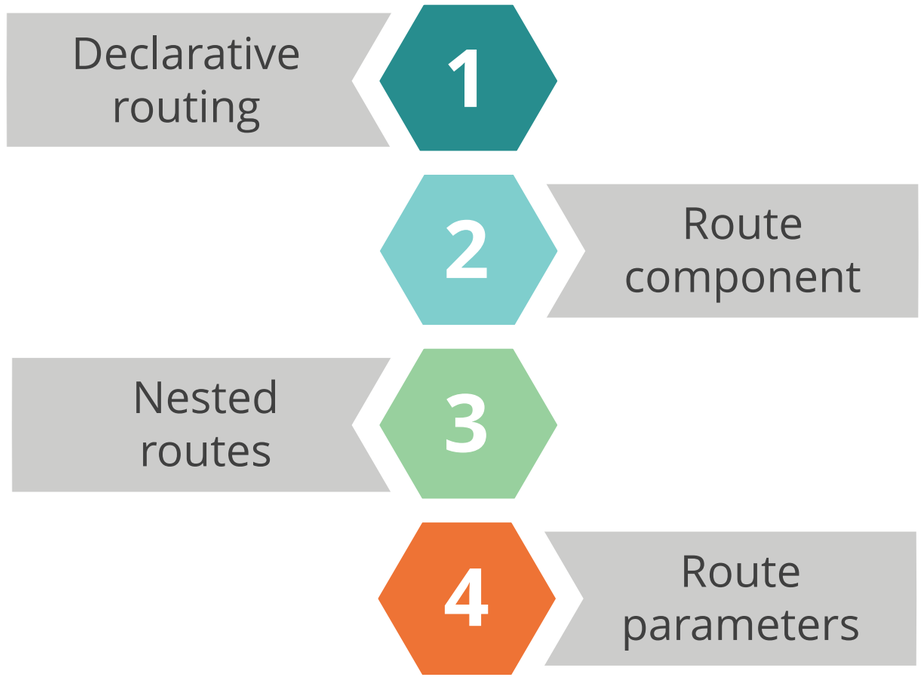
- In a typical web application, when users interact with different parts of the interface, the content changes without requiring a full page reload.
- React Router enables developers to manage these transitions and URL changes seamlessly within a React application.

Benefits of Routing

- Single-page application experience
- Organized code structure
- Navigation control
- Improved user flow

Key Components of React Routing

React router enables developers to manage the transitions and URL changes seamlessly within a React application. Here are its key components:



Implementing a React Routing Application

Problem Statement:

Duration: 10 min

You have been assigned the task to implement react routing for navigating between various components.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Create a React.js project
2. Install routing dependencies and its features
3. Create three components: Home, About Us, and Contact Us
4. Create NavBar.js component file
5. Add the Bootstrap features to the project
6. Test the application

React Router V6

React Router V6

It introduces several powerful new features, as well as improved compatibility with the latest versions of React. Below are the commands that will help you upgrade to v6 from v5:

Upgrade to React v16.8: npm install --save react@^16.8.0 react-dom@^16.8.0

Upgrade to React Router v5.1: npm install react-router-dom@5.1.0

Upgrade to React Router v6: npm install react-router@6 react-router-dom@6

Setting Up React Router

Step 1: Installation
Install **react-router-dom** using npm in the terminal

Example:

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

Setting Up React Router

Step 2: Router component
Wrap your application with **BrowserRouter**.

Example:

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

Step 3: Route component
Use the **Route** component to define routes in your application

Example:

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

oconst App = () => {
return (
 <div>
 <Route path="or" exact component={Home}or>
<Route path="orabout" component={About}or>
{or*Add more routes as needed *or}
<ordiv>
 );
 };
```

Setting Up React Router

Step 4: Link component
Utilize the **Link** component for navigation

Example:

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

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

Step 5: Switch component (optional)
Use the **switch** component to render only the first matching route

Example:

```
import { Switch, Route } from 'react-router-dom';
const App = () => {
  return (
    <div>
      <Switch>
        <Route path="or" exact component={Home}
or>
        <Route path="orabout" component={About}
or>
        <Route path="orcontact"
component={Contact} or>
      <orSwitch>
    <ordiv>
  );};
```

Setting Up React Router

Step 6: Route parameters
For dynamic routes, use route parameters to capture values from the URL.

Example:

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

const UserProfile = ({ match }) => {
  return <div>User ID: {match.params.id}<ordiv>;
};

const App = () => {
  return (
    <div>
      <Route path="Ruberoid" component={UserProfile} or>
    <ordiv>
  );
};
```

Creating a Simple Route for Different Sections of the App

01 **Home section**

Is the default landing page of the application

02 **About section**

Has information about the application or the team behind it

03 **Contact section**

Is a page with contact information or a contact form

04 **Production**

Displays a list of products or services offered

05 **Product details**

Shows detailed information about a specific product based on its ID

06 **404 not found**

Renders when none of the specified routes match, indicating a page not found

Example: Set of Routes

Here is the code representation of simple routes:

Example:

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

const App = () => {
  return (
    <Router>
      <Switch>
        <Route path="/" exact component={Home} />
        <Route path="/about" component={About} />
        <Route path="/contact" component={Contact} />
        <Route path="/products" exact component={Products} />
        <Route path="/products/:id" component={ProductDetails} />
        <Route component={NotFound} />
      </Switch>
    </Router>
  );
};
```

Advanced features of React Router 6

Data Fetchers: Overview

They facilitate interaction with route actions and loaders, preserving standard benefits such as error handling and race condition management without triggering browser navigation.

- Define loader functions by creating functions that fetch data using the web's fetch API
- Attach loaders to routes by assigning loader functions to the loader property of route objects
- Access data using the **useloaderdata** Hook within components to retrieve and utilize fetched data

Data Fetchers: Advantages

Automatic refetching

Centralized data management

Error handling

Improved code organization

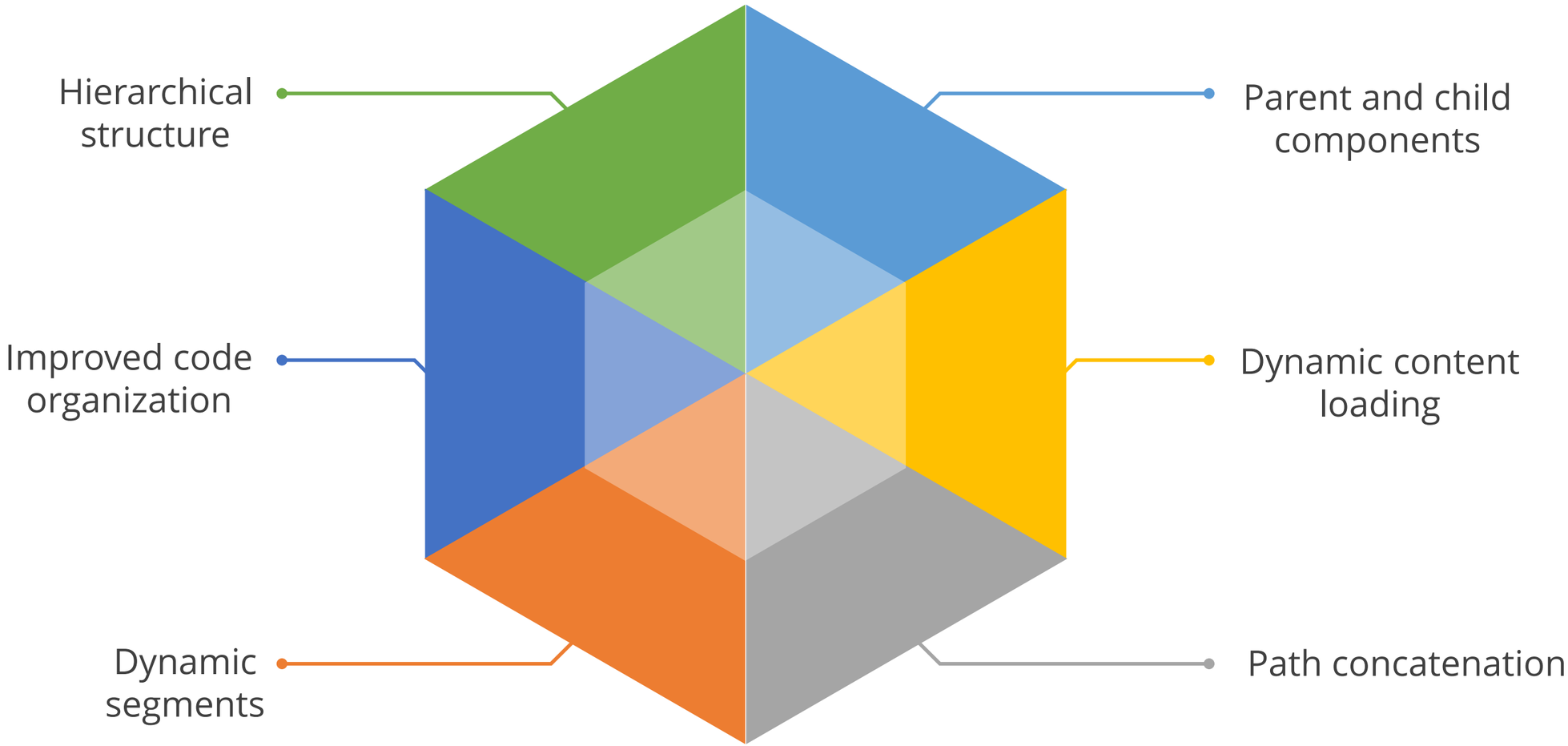
Nested Route: Overview

Nested routes are one of React’s unique features that allow users to swap out specific view fragments based on the current path.

- Nested routes in React Router provide a way to organize and structure the routing logic for complex applications.
- Instead of having a flat route structure, where all routes are at the top level, nested routes allow you to create a hierarchy of routes.
- Nested routes provide a robust framework for designing intricate user interfaces in a React application.

Nested Route: Overview

The concepts included in Nested Route are:



Implementing Programmatic Routing with Login Page and Nested Routing

Problem Statement:

Duration: 10 min

You have been assigned a task to demonstrate react routing by navigating to the home component after a successful login with credentials and incorporating nested concepts within the home component.

Assisted Practice: Guidelines

Steps to be followed:

1. Set up a React project
2. Create files for the home component
3. Test the application

Introduction to Route Parameter

Route parameters are powerful features that allow you to capture dynamic values from the URL, enabling the creation of dynamic and data-driven components.

They enhance the flexibility of your React application by allowing components to respond dynamically to different URL patterns.

Passing Parameters In React Router URLs

Passing and extracting parameters in route URLs involves using dynamic segments in the route path.

1. Define a Route with a parameter:

Example:

```
<Route path="Ruberoid" component={UserProfile} or>
```

Passing Parameters In React Router URLs

2. Link with parameters

Example:

```
<Link to="oruseror123">User Profile<orLink>
```

Extracting Parameters In React Router URLs

Extracting the parameter involves isolating and obtaining specific variables or values for further analysis or use in a process.

3. Accessing parameters in component

Example:

```
const UserProfile = ({ match }) => {
  return <div>User ID:{match.params.id}
<ordiv>;
};
```

Extracting Parameters in React Router URLs

Here is an example of passing and extracting parameters in React Router URLs:

Example:

```
import { BrowserRouter as Router, Route, Link } from 'react-router-dom';
const UserProfile = ({ match }) => {
  return <div>User ID: {match.params.id}<ordiv>;
};
const App = () => {
  return (
    <Router>
      <div>
        <nav>
          <ul>
            <li><Link to="oruseror123">User Profile<orLink><orli>
          <orul>
        <ornav>
       <Route path="Ruberoid" component={UserProfile} or>
      <ordiv>
    <orRouter>
  );
};
```

What Is Programmatic Navigation?

Programmatic Navigation in JavaScript for web and single-page applications(SPAs) allows navigating through the application via code instead of user clicks.

It controls the flow of the application based on logic defined in the code.

How Programmatic Navigation Works with SPAs?

In single-page applications (SPAs), programmatic navigation is crucial because the entire page does not reload when moving between different views or components.

This type of navigation allows the application to update the URL and display the appropriate content without needing a full-page refresh.

Implementing Programmatic Navigation

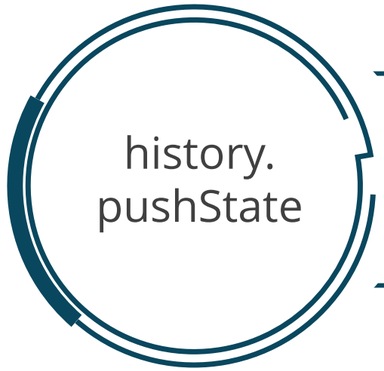
Programmatic navigation can be implemented by using the following Hooks:

useHistory:
Offers a way to manage session history

useNavigate:
Simplifies navigation commands

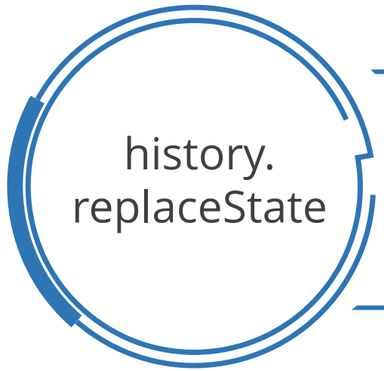
Programmatic Navigation in Browser

Programmatic navigation is achieved by interacting with the browser's history API in the following ways:



history.
pushState

This method adds an entry to the browser's session history stack, changing the URL without reloading the page.



history.
replaceState

This method changes the current URL in the browser's history stack without a page reload, replacing the current history entry instead of adding a new one.

Programmatic Navigation: Use Cases

01 Redirecting users after certain actions (like form submissions, log in or log out processes)

02 Guarding routes (like preventing access to certain routes if the user is not authenticated)

03 Handling routes based on application state or user interactions dynamically

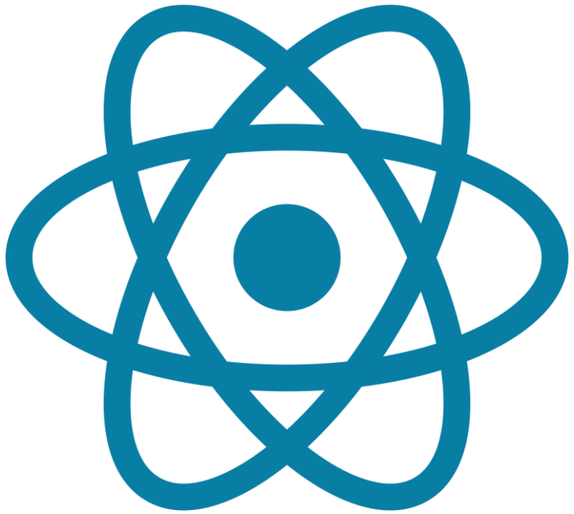
What Is Route Guard?

A Route Guard is a security feature in web application frameworks that regulates user access to different routes or pages within an application.

It functions by evaluating whether a user meets certain criteria, like authentication status or permissions, to navigate to or from a specific route.

How Route Guards Works With SPAs?

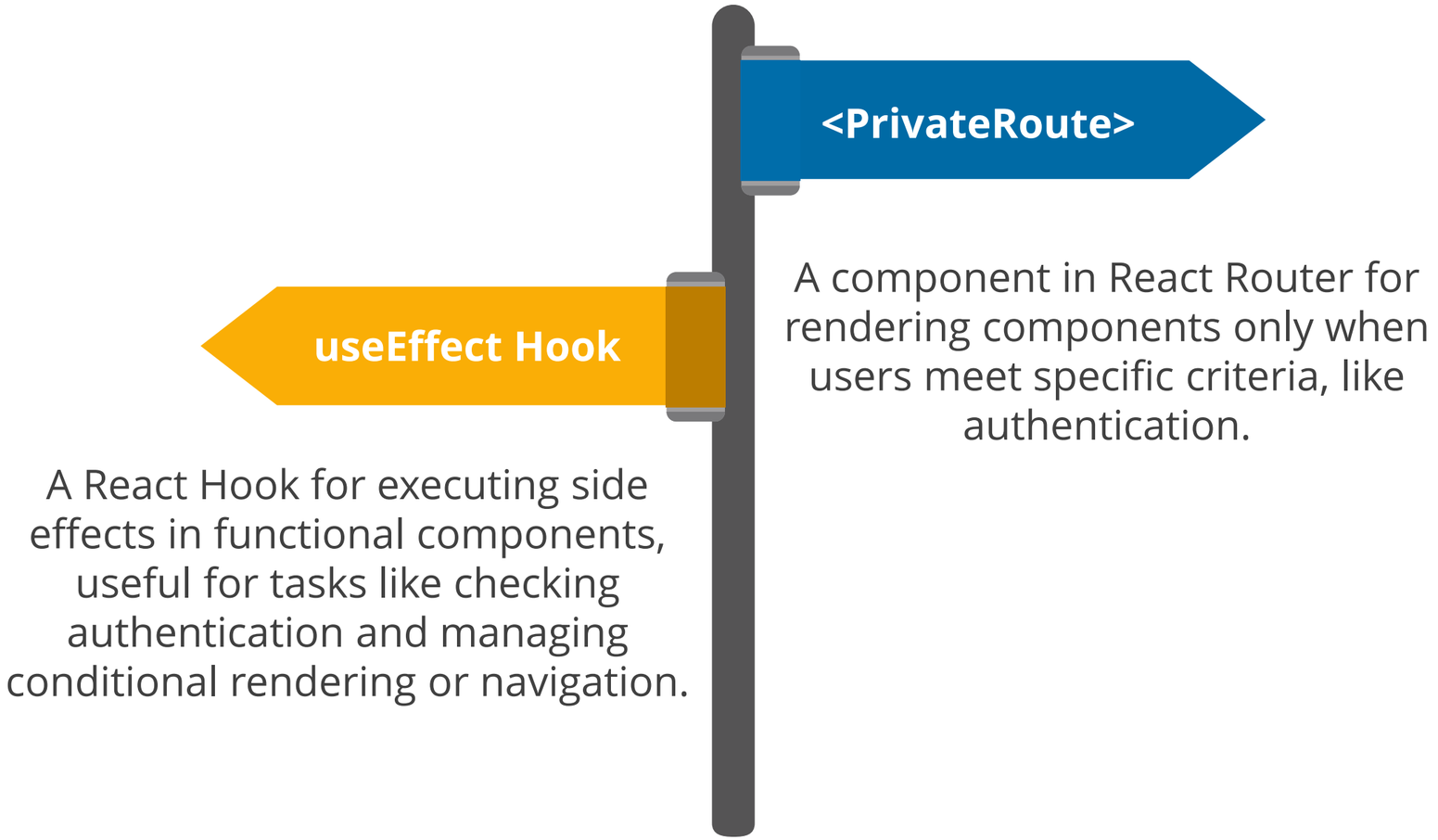
In single-page applications (SPAs), routing is managed on the client side, allowing for seamless page transitions without server requests.



Route Guards in SPAs intercept routing events programmatically to determine if a particular route should be activated based on specific logic.

Implementing Route Guards

Route guards in React can be implemented by using the following:



Types of Route Guards

01

Before Enter or Leave: To check conditions before entering or leaving a route

02

Resolve: To resolve data before the route is activated

03

After Enter or Leave: To perform actions after entering or leaving a route

Route Guards: Use Cases

01

Route guards prevents non-authenticated users from accessing certain routes.

02

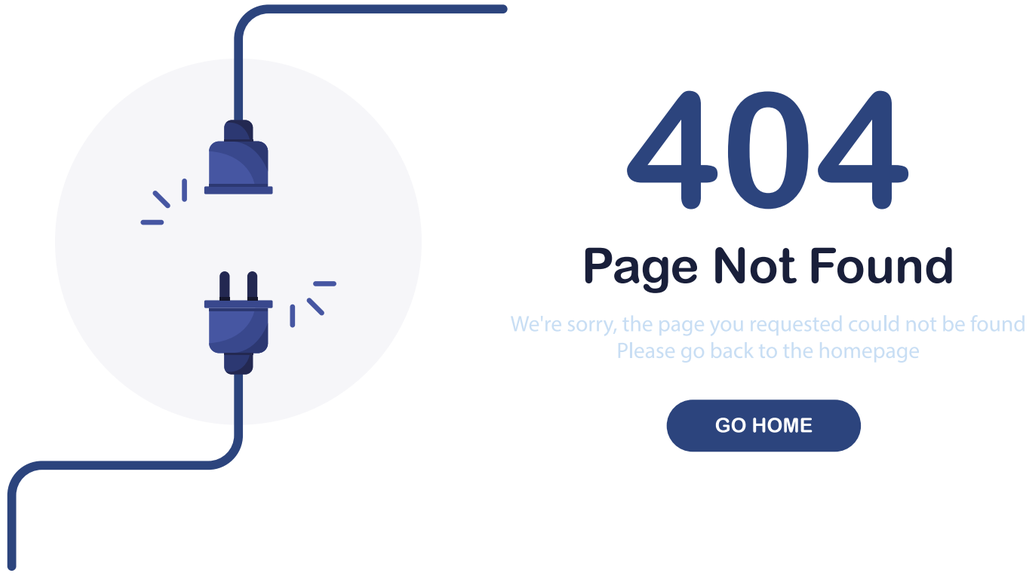
It prevents users from navigating away from the current view based on unsaved changes or pending operations.

03

It helps in loading necessary data before the route is activated.

What Is 404 Error?

A 404 page is shown when a user tries to access a route or resource that is not available.



The 404 is an HTTP status code representing **Not Found**.

Custom 404 Error Page Using React Router

In React, React Router allows users to define a **<Route>** with a path of ***** or **404**, which is used to render a custom **404 component**.

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

Importance of Custom 404 Error Page

01 **User Experience:** Custom 404 pages provide a consistent and less intimidating user experience compared to generic error messages.

02 **Navigation Aid:** They offer helpful links and guidance back to the site's main sections, keeping users engaged.

03 **Brand Image:** They reinforce brand consistency and can leave a positive impression, even in error scenarios.

Importance of Custom 404 Error Page

04 **Feedback Channel:** They encourage users to report broken links, aiding in site maintenance and error resolution.

05 **SEO Maintenance:** Properly configured 404 pages help maintain SEO health by preventing search engines from indexing non-existent pages.

06 **Creative Outlet:** They offer an opportunity for creativity and humor, making the site 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 with React Router:

Step 1: Create the 404 component

```
oror NotFoundPage.js

import React from 'react';

const NotFoundPage = () => {
  return (
    <div>
      <h1>404 Not Found<orh1>
      <p>The page you are looking for doesn't exist.<orp>
    <ordiv>
  );
};

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

```
oror App.js

import React from 'react';
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';
import HomePage from '.orHomePage'; oror Import your home page component
import AboutPage from '.orAboutPage'; oror Import your about page component
import NotFoundPage from '.orNotFoundPage'; oror Import your 404 page
component

const App = () => {
  return (
    <Router>
      <Switch>
        <Route exact path="or" component={HomePage} or>
        <Route path="orabout" component={AboutPage} or>
        {or* Define other routes as needed *or}

        {or* Catch-all Route for 404 Not Found *or}
        <Route component={NotFoundPage} or>
      <orSwitch>
    <orRouter>
  );
};

export default App;
```

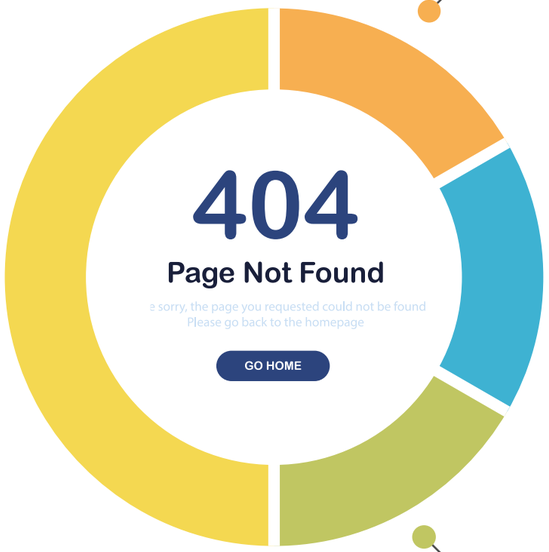
Creating a Custom 404 Page with React Router

Step 4: Test your 404 page

```
npm start
```

This configuration displays a custom 404 page for undefined routes in your React app, enhancing user experience and managing navigation errors smoothly.

404 Error: Use Cases



01

Preventing non-authenticated users from accessing certain routes.

02

Preventing users from navigating away from the current view based on unsaved changes or pending operations.

03

Loading necessary data before the route is activated.

Creating a React Router

Problem Statement:

Duration: 10 min

You have been assigned a task to create a React routing with protected features to ensure that only authenticated user can access specific components.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Create and set up a React project
2. Update index.js file
3. Create the required files for the src folder
4. Add the bootstrap features
5. Test the application

ASSISTED PRACTICE

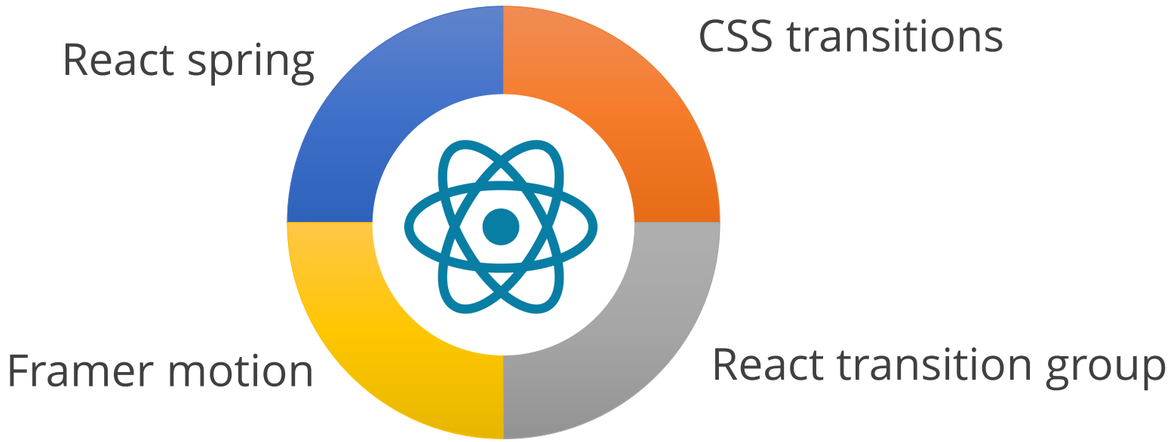
What Are Route Animations?

Route animations in React are visual effects that occur during the transition between routes in a Single Page Application (SPA), providing a dynamic change rather than a static page swap.

These animations create a smooth visual flow as users navigate, improving interactivity and delivering a polished navigation experience.

How to Implement Route Animations?

React requires external libraries for animations, with several options available to add route transitions some of them are as follows:



Example: Using Route Animation

Here is an example demonstrating the use of React Transition Group for route transition animations:

Example:

```
import { Route, Switch, useLocation } from 'react-router-dom';
import { TransitionGroup, CSSTransition } from 'react-transition-group';

function AnimatedSwitch() {
  const location = useLocation();

  return (
    <TransitionGroup>
      <CSSTransition key={location.key} classNames="fade" timeout={300}>
        <Switch location={location}>
          <Route exact path="or" component={Home} or>
          <Route path="orabout" component={About} or>
          {or* other routes *or}
        <orSwitch>
      <orCSSTransition>
    <orTransitionGroup>
  );
}
```

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.

Knowledge Check

Knowledge Check 1

What is the primary purpose of setting up React Router in a React application?

A. To manage state globally across the application

B. To enable navigation between different components without reloading the page

C. To increase the performance of the application

D. To store data in the browser's local storage

Knowledge Check 1

What is the primary purpose of setting up React Router in a React application?

A. To manage state globally across the application

B. To enable navigation between different components without reloading the page

C. To increase the performance of the application

D. To store data in the browser's local storage

The correct answer is **B**

React Router enables seamless navigation between different components in React apps without page reloads, enhancing the user experience in single-page applications (SPAs).

Knowledge Check 2

What is the primary use of route parameters in React Router?

A. To style different components

B. To pass and display dynamic content based on the URL

C. To redirect users to external websites

D. To encrypt user data

Knowledge Check 2

What is the primary use of route parameters in React Router?

A. To style different components

B. To pass and display dynamic content based on the URL

C. To redirect users to external websites

D. To encrypt user data

The correct answer is **B**

Route parameters in React Router allow for passing and retrieving values from the URL, enabling the display of dynamic content like user information based on URL parameters.

Knowledge Check 3

In React, which Hook is used for programmatically navigating between routes in newer versions of React Router?

A. useState

B. useEffect

C. useNavigate

D. useContext

Knowledge Check 3

In React, which Hook is used for programmatically navigating between routes in newer versions of React Router?

A. useState

B. useEffect

C. useNavigate

D. useContext

The correct answer is **C**

The useNavigate Hook in newer versions of React Router is specifically designed for programmatic navigation, allowing developers to navigate between routes in a React application.

Knowledge Check 4

Which method is part of the browser's History API and is used to change the URL without reloading the page in programmatic navigation?

A. document.pushState()

B. window.replaceState()

C. history.pushState()

D. location.changeState()

Knowledge Check 4

Which method is part of the browser's History API and is used to change the URL without reloading the page in programmatic navigation?

A. document.pushState()

B. window.replaceState()

C. history.pushState()

D. location.changeState()

The correct answer is **C**

The history.pushState() method is used in the History API to add a new entry to the browser's history stack, changing the URL without a full page reload.

Knowledge Check 5

Which common use case of Route Guards involves ensuring that data required for a route is loaded before the route is activated?

A. Preventing non-authenticated users from accessing certain routes

B. Preventing users from navigating away from a route with unsaved changes

C. Loading necessary data before the route is activated

D. Logging and analytics to track navigation behavior

Knowledge Check 5

Which common use case of Route Guards involves ensuring that data required for a route is loaded before the route is activated?

A. Preventing non-authenticated users from accessing certain routes

B. Preventing users from navigating away from a route with unsaved changes

C. Loading necessary data before the route is activated

D. Logging and analytics to track navigation behavior

The correct answer is **C**

A common use for Route Guards is to preload essential data before activating a route, ensuring all necessary information is ready for the user, thereby enhancing the user experience and avoiding incomplete or loading screens.