React Router — это библиотека для управления маршрутизацией в приложениях на React. Она помогает разработчикам создавать одностраничные приложения (SPA), которые реагируют на смену URL без перезагрузки страницы. Это позволяет приложению быть быстрым, а навигацию — плавной.

Давайте разберем React Router по шагам, начиная с основ и постепенно углубляясь.

**1. Что такое маршрутизация и зачем она нужна?**

В традиционных веб-приложениях каждая страница имеет свой HTML-файл. Делая переход на другую страницу, браузер запрашивает сервер для получения нового HTML-файла. В одностраничных приложениях (SPA) такого нет — у нас есть один HTML-файл, который меняется динамически с помощью JavaScript.

React Router помогает:

Менять представление (UI) на экране при изменении URL.

Управлять навигацией между "страницами" без реальной перезагрузки.

Позволять глубоко внедренные ссылки (например, /profile/settings).

Сохранять историю переходов, чтобы пользователь мог нажимать "Назад" или "Вперед".

**2. Установка React Router**

Чтобы начать работу, нужно установить пакет библиотеки.

npm install react-router-dom

Если у вас версия React 18 (или выше), используйте совместимую версию React Router v6 или выше.

**3. Основные компоненты React Router**

Библиотека React Router предоставляет несколько важных компонентов для работы:

BrowserRouter:

   Это главный компонент, который оборачивает ваше приложение.

   Он отвечает за слежение за URL, используя историю браузера.

Routes и Route:

   Routes служит контейнером для маршрутов.

   Route описывает один маршрут, у которого есть путь (path) и компонент, который рендерится при совпадении пути.

Link:

   Это компонент, с помощью которого пользователи могут перемещаться между страницами.

   Работа Link похожа на HTML ``, но без перезагрузки страницы.

useNavigate:

   Это хук, который позволяет программно перенаправлять пользователя на другой маршрут.

**4. Как создать базовый пример**

Вот процесс создания простого приложения с React Router:

**1. Настройка BrowserRouter**

import React from 'react';

import { BrowserRouter } from 'react-router-dom';

function App() {

  return (

    <BrowserRouter>

      <div>

        <h1>Моё приложение</h1>

      </div>

    </BrowserRouter>

  );

}

export default App;

BrowserRouter оборачивает ваше приложение, чтобы включить маршрутизацию. Теперь мы можем добавлять маршруты с помощью Routes и Route.

**2. Добавление маршрутов с Routes и Route**

Допустим, у нас есть три "страницы": Главная, О нас и Контакты. Создадим их как отдельные компоненты:

import React from 'react';

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

// создаем наши страницы (компоненты)

const Home = () => <h2>Главная</h2>;

const About = () => <h2>О нас</h2>;

const Contact = () => <h2>Контакты</h2>;

// компоненты маршрутизации

function App() {

  return (

    <BrowserRouter>

      <div>

        <h1>Моё приложение</h1>

        <Routes>

          <Route path="/" element={<Home />} />

          <Route path="/about" element={<About />} />

          <Route path="/contact" element={<Contact />} />

        </Routes>

      </div>

    </BrowserRouter>

  );

}

export default App;

Что здесь происходит:

``:

   Оборачиваем приложение, чтобы включить маршрутизацию (обязательно).

` и `:

   Внутри ` мы определяем маршруты с помощью `.

   Каждый ` связывает path (путь) с компонентом (element`):

     / — это корневой путь (главная страница).

     /about — путь для "О нас".

     /contact — путь для "Контакты".

element={}:

   Указываем, какой компонент должен отображаться, если URL совпадает с path.

Теперь, если вы откроете приложение, по разным URL'ам будут рендериться соответствующие компоненты:

/\*\* покажет компонент Home.

/about\*\* покажет компонент About.

/contact\*\* покажет компонент Contact.

**3. Добавление навигации с помощью ``**

Теперь добавим меню для навигации между страницами. Вместо обычного тега ` в React Router используется компонент `, который обеспечивает плавную навигацию без перезагрузки страницы.

Обновленный код:

import React from 'react';

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

const Home = () => <h2>Главная</h2>;

const About = () => <h2>О нас</h2>;

const Contact = () => <h2>Контакты</h2>;

function App() {

  return (

    <BrowserRouter>

      <div>

        <h1>Моё приложение</h1>

        {/\* Навигационное меню \*/}

        <nav>

          <Link to="/">Главная</Link> | {' '}

          <Link to="/about">О нас</Link> | {' '}

          <Link to="/contact">Контакты</Link>

        </nav>

        {/\* Определение маршрутов \*/}

        <Routes>

          <Route path="/" element={<Home />} />

          <Route path="/about" element={<About />} />

          <Route path="/contact" element={<Contact />} />

        </Routes>

      </div>

    </BrowserRouter>

  );

}

export default App;

Теперь сверху у нас есть навигационное меню:

Ссылки Главная, О нас, Контакты используют компонент ``, который обновляет URL без перезагрузки страницы.

**4. Программная навигация с useNavigate**

Хук useNavigate позволяет программно менять маршрут. Например, после выполнения действия или обработки данных вы хотите перенаправить пользователя на другую страницу.

Пример:

import React from "react";

import { useNavigate } from "react-router-dom";

const Home = () => {

  const navigate = useNavigate();

  const goToAbout = () => {

    navigate('/about'); // программное перенаправление

  };

  return (

    <div>

      <h2>Главная</h2>

      <button onClick={goToAbout}>Перейти на страницу "О нас"</button>

    </div>

  );

};

export default Home;

Когда пользователь нажимает кнопку, его автоматически перенаправляют на /about.

**5. Управление "не найдено" (страница 404)**

Чтобы обработать случай, когда пользователь вводит несуществующий URL, можно добавить специальный ` без path`. Он будет отображаться, если никто из остальных маршрутов не подходит.

Пример:

<Routes>

  <Route path="/" element={<Home />} />

  <Route path="/about" element={<About />} />

  <Route path="/contact" element={<Contact />} />

  <Route path="\*" element={<h2>Страница не найдена - 404</h2>} />

</Routes>

В этом случае, если пользователь зайдет на, например, /random, отобразится текст "Страница не найдена - 404".

**6. Вложенные маршруты**

React Router позволяет создавать вложенные маршруты. Это полезно, если у вас есть общая структура страницы (например, профиль пользователя с подпунктами).

Пример:

const Profile = () => (

  <div>

    <h2>Профиль</h2>

    <nav>

      <Link to="settings">Настройки</Link> | <Link to="info">Инфо</Link>

    </nav>

    <Routes>

      <Route path="settings" element={<h3>Настройки профиля</h3>} />

      <Route path="info" element={<h3>Информация о пользователе</h3>} />

    </Routes>

  </div>

);

<Routes>

  <Route path="/profile/\*" element={<Profile />} />

</Routes>;

Теперь, перейдя на /profile/settings или /profile/info, будут отображаться вложенные маршруты.

**7. Передача параметров в маршрут**

React Router позволяет добавлять динамические параметры в URL. Например, вы хотите отобразить информацию о пользователе по его ID.

Пример:

import { useParams } from 'react-router-dom';

const User = () => {

  const { id } = useParams(); // Получение параметра из URL

  return <h2>Пользователь ID: {id}</h2>;

};

<Routes>

  <Route path="/user/:id" element={<User />} />

</Routes>;

Если вы откроете /user/123, то на странице отобразится: "Пользователь ID: 123".

**8. Заключение**

React Router — мощный инструмент для навигации внутри React-приложений. Для начала важно понять базовые компоненты (BrowserRouter, Routes, Route, Link), а затем освоить более сложные механики, такие как вложенные маршруты, параметры в пути или программная навигация.