Лекция 9
Определение маршрутов для навигации

1. Определение маршрутов для навигации

Теперь, когда мы обернули наше приложение **BrowserRouter** для включения маршрутизации, следующим шагом является указание маршрутов (путей), которые будет включать наше приложение. Эти маршруты определяют общую структуру приложения и управляют тем, какой контент отображается в зависимости от URL-адреса.

React Router предоставляет два основных компонента для определения маршрутов: **Routes** и **Route**. Компонент Routes выступает в качестве контейнера для всех определений маршрутов, а компонент Route представляет собой конкретную конфигурацию маршрута.

Вот синтаксис для определения маршрутов:

```
<Routes>
  <Route path="<path_value>" element={<component_to_render>} />
  <Route path="<path_value>" element={<component_to_render>} />
  <Route path="<path_value>" element={<component_to_render>} />
</Routes>
```

В этой структуре:

- **<path_value>**: Задает конфигурацию URL-адреса для маршрута;
- **<component_to_render>**: Определяет компонент, который будет отображаться при переходе пользователя по указанному пути.

Пример. Давайте применим эту концепцию к нашему проекту. Откройте **App.jsx** файл и импортируйте Routeскомпоненты Routeиз react-router-domбиблиотеки:

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

Внутри кода **return** компонента **App** определите маршруты для нашего приложения:

```
1.const App = () => {
2.  return (
3.    <>
4.      <Routes>
5.        <Route path="/" element={<HomePage />} />
6.        <Route path="/about" element={<AboutPage />} />
7.        <Route path="/contacts" element={<ContactsPage />} />
8.      </Routes>
9.    </>
10.  );
11.};
```

Пояснение к коду:

- Строки 4, 8: <Routes>: Этот компонент служит контейнером для определения маршрутов в приложении, выступая в качестве родительского компонента для определения отдельных маршрутов;
- Строка 5: <Route>: Представляет собой определение маршрута. Она указывает, что при совпадении URL-пути /компонент <HomePage />должен быть отображен;
- Строка 6: <Route>: Определяет еще один маршрут. Когда путь URL совпадает /about, компонент <AboutPage />будет отображен;
- Строка 7: <Route>: Определяет другой маршрут. Если путь URL совпадает /contacts, <ContactsPage />компонент будет отображен.

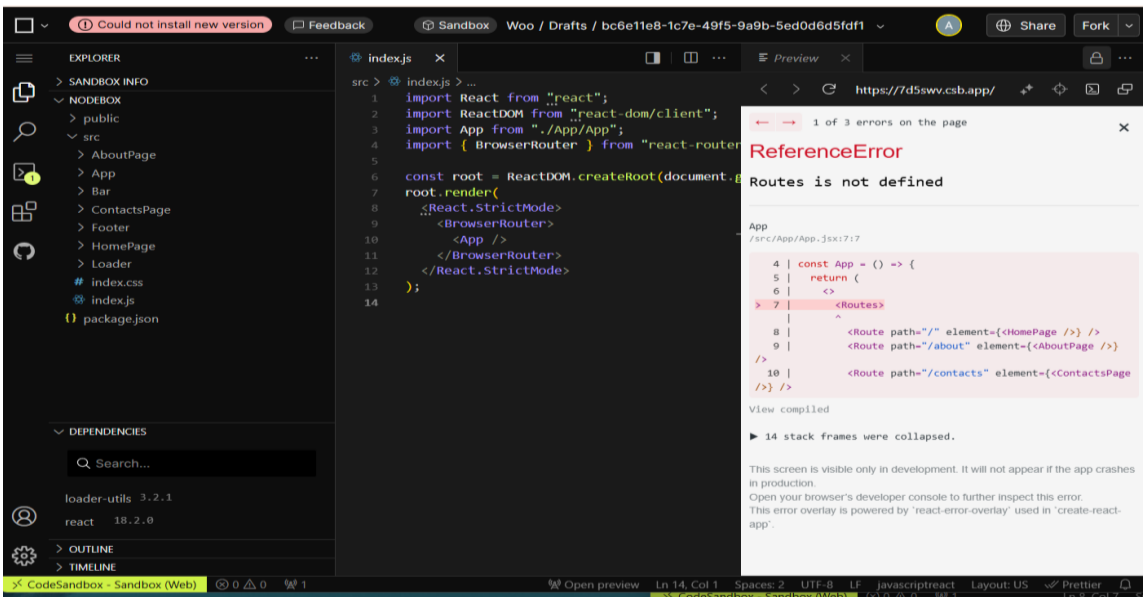
Давайте на мгновение остановимся и подведем итоги того, чего мы уже достигли в нашем проекте. Крайне важно обеспечить четкое понимание рассмотренных концепций и проделанной работы.

Важно отметить, что код, предоставленный в CodeSandbox, содержит ошибки. Ошибки возникают из-за отсутствия импорта необходимых компонентов для рендеринга, таких как **HomePage, AboutPage**, и **ContactsPage**. В следующей главе мы рассмотрим эту проблему и продемонстрируем правильный синтаксис для импорта компонентов.

index.js:

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App/App";
import { BrowserRouter } from "react-router-dom";

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <React.StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </React.StrictMode>
);
```



2. Оптимизация импорта с помощью отложенной загрузки.

Как уже упоминалось ранее, нам необходимо импортировать компоненты, которые будут отображаться при переходе пользователя по определенным путям в нашем приложении. Для этого мы воспользуемся функцией `lazy` из библиотеки React.

Синтаксис использования lazy функции для импорта компонента выглядит следующим образом:

```
import { lazy } from 'react';
const View = lazy(() => import('./View.js'));
```

Вот что делает каждая часть этого синтаксиса:

- Строка 1: Импортирует **lazy** функцию из библиотеки React. Эта функция позволяет разделять код и выполнять отложенную загрузку компонентов;
- Строка 3: Объявляет переменную (**View** в данном случае) с помощью **lazy** функции. Функция **lazy** принимает функцию в качестве аргумента, и эта функция возвращает динамический оператор импорта. Оператор импорта асинхронно импортирует указанный файл/модуль (в данном **случае View.js**) с помощью import ключевого слова. Обернув оператор импорта ключевым словом **lazy**, мы включаем отложенную загрузку компонента **View.js.**

Пример. Давайте применим эту концепцию к нашему проекту, импортировав компоненты для различных представлений. К этим представлениям относятся главная страница, страница «О нас» и страница «Контакты». Важно отметить, что эти представления уже созданы, и наша текущая задача — использовать их в нашей маршрутизации:

```
import { lazy } from "react";

const HomePage = lazy(() => import("../HomePage/HomePage"));
const AboutPage = lazy(() => import("../AboutPage/AboutPage"));
const ContactsPage = lazy(() => import("../ContactsPage/ContactsPage"));
```

Мы используем **lazy** функцию из библиотеки React для импорта компонентов для различных представлений. Каждый компонент импортируется динамически с помощью оператора **import**, и мы указываем относительные пути к файлам компонентов **(../HomePage/HomePage, ../AboutPage/AboutPage, ../ContactsPage/ContactsPage**).

Благодаря импорту этих компонентов с отложенной загрузкой мы гарантируем, что они загружаются только тогда, когда это необходимо, что способствует более эффективному и отзывчивому взаимодействию с пользователем.

3. Реализация резервного компонента

Давайте сосредоточимся на реализации резервного компонента. Этот шаг имеет решающее значение для улучшения пользовательского опыта за счет предоставления визуальной обратной связи во время загрузки определенных веб-страниц.

React предоставляет **Suspense** компонент для этой цели. Компонент **Suspense** позволяет указать резервный компонент, который отображается во время загрузки основного контента. Вот синтаксис для реализации этой функции:

```
1. import { Suspense } from "react";
2.
3. const App = () => {
4.  return (
5.    <Suspense fallback={<LoaderComponent />}>
6.      {/* Routes */}
7.    </Suspense>
8.  );
9. };
```

- Строка 1: Мы импортируем Suspenseкомпонент из библиотеки React;
- Строки 5 и 7: Мы оборачиваем все маршруты приложения компонентом Suspense. Это позволяет нам отображать резервный компонент, пока загружается любой маршрут;
- Строка 5: SuspenseКомпонент принимает fallbackсвойство. В данном случае fallbackсвойство устанавливается в значение <LoaderComponent />, которое определяет содержимое, отображаемое во время загрузки основного контента.

Пример. Теперь давайте интегрируем в наше приложение механизм резервного копирования:

```
import React, { Suspense, lazy } from "react";
import { Routes, Route } from "react-router-dom";
import Loader from "../Loader/Loader";
import "../index.css";

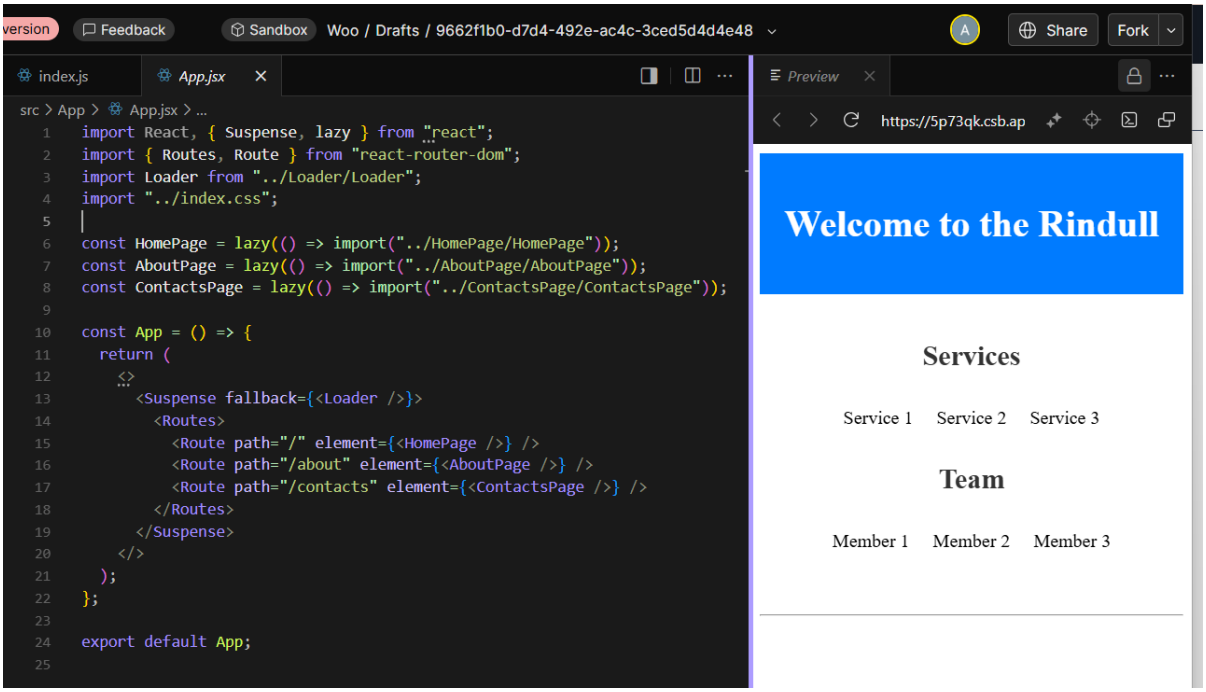
const HomePage = lazy(() => import("../HomePage/HomePage"));
const AboutPage = lazy(() => import("../AboutPage/AboutPage"));
const ContactsPage = lazy(() => import("../ContactsPage/ContactsPage"));

const App = () => {
  return (
    <>
      <Suspense fallback={<Loader />}>
        <Routes>
          <Route path="/" element={<HomePage />} />
          <Route path="/about" element={<AboutPage />} />
          <Route path="/contacts" element={<ContactsPage />} />
        </Routes>
      </Suspense>
    </>
  );
};

export default App;
```

Приведенный код демонстрирует интеграцию механизма резервного копирования в **App** компонент. Используя **Suspense** компонент с **fallback** указанным параметром **<Loader />,** мы гарантируем отображение визуальной обратной связи во время загрузки маршрутов.

Исходный код:
https://codesandbox.io/p/sandbox/9662f1b0-d7d4-492e-ac4c-3ced5d4d4e48-5p73qk?file=%2Fsrc%2FApp%2FApp.jsx%3A5%2C1&from-embed



Приведенный код демонстрирует интеграцию механизма резервного копирования в **App** компонент. Используя **Suspense** компонент с **fallback** указанным параметром **<Loader />**, мы гарантируем отображение визуальной обратной связи во время загрузки маршрутов.

Давайте сделаем паузу и оценим прогресс, достигнутый нами на данный момент в рамках проекта.

Один из важных аспектов, который нам необходимо решить, — это отсутствие кнопок навигации, позволяющих пользователям легко перемещаться между различными страницами. Хотя наша функция маршрутизации реализована, в настоящее время пользователи не могут активировать эти маршруты.

В следующей главе мы рассмотрим этот вопрос, представив навигационные кнопки и продемонстрировав, как использовать Linkкомпонент из React Router.