

# Урок 11.

## Роутинг react-router-dom

<b>Введение в роутинг</b>	<b>2</b>
<b>Задание для закрепления</b>	<b>5</b>
<b>Настройка проекта</b>	<b>6</b>
<b>Задание для закрепления</b>	<b>9</b>
<b>Основные компоненты в React Router 6</b>	<b>10</b>
<b>Задание для закрепления</b>	<b>17</b>
<b>Хуки и их важность</b>	<b>18</b>
<b>useNavigate()</b>	<b>20</b>
<b>Задание для закрепления</b>	<b>23</b>
<b>useLocation()</b>	<b>25</b>
<b>Задание для закрепления</b>	<b>28</b>
<b>useParams()</b>	<b>31</b>
<b>Задание для закрепления</b>	<b>34</b>

## Введение в роутинг



**Роутинг – это процесс определения маршрутов (путей), по которым приложение может переходить в зависимости от URL.**

Пример: Веб-приложение имеет несколько страниц (Главная, О нас, Контакты). Когда пользователь вводит URL /about, приложение должно отобразить страницу "О нас".

Цель: Позволить пользователям легко навигировать по приложению.

Преимущества роутинга в одностраничных приложениях (SPA):

- Без перезагрузки страницы: Переход между различными частями приложения происходит без полной перезагрузки страницы, что улучшает пользовательский опыт и повышает производительность.
- Сохранение состояния: При переходе между страницами состояние приложения (форма, таблицы) сохраняется, и это делает использование приложения более удобным.
- SEO и Deep Linking: Возможность создания читаемых URL-адресов, которые можно использовать для "глубоких ссылок", что полезно для SEO и позволяет пользователям легко делиться ссылками на конкретные части приложения.
- Улучшенная производительность: Поскольку все ресурсы загружаются один раз, последующие переходы между страницами происходят быстрее.
- Управление доступом: Возможность создания приватных/защищенных маршрутов, доступ к которым может быть ограничен на основе роли пользователя или его состояния (например, аутентификации).



**react-router-dom – это библиотека, специально созданная для работы с роутингом в React-приложениях.**

Она позволяет легко создавать и настраивать маршруты, а также управлять ими. Предоставляет компоненты и хуки, которые упрощают процесс настройки роутинга в React-приложениях.

Ключевые особенности:

- Компоненты: Библиотека предоставляет ряд компонентов для определения маршрутов (<BrowserRouter>, <Routes>, <Route>, <Link>, <NavLink> и др.).

- Хуки: Также доступны хуки (useNavigate, useLocation, useParams, и др.), которые упрощают работу с роутингом посредством использования декларативного подхода.
- Динамические маршруты: Возможность создания маршрутов с параметрами, таких как /user/:id.
- Вложенные маршруты: Поддержка вложенных маршрутов, что облегчает создание многоуровневых навигационных структур.
- Программная навигация: Возможность выполнения навигации программным путем посредством хуков.

Разберем, как установить библиотеку react-router-dom и подготовить проект к использованию роутинга.



**npm (Node Package Manager) — это стандартный пакетный менеджер для Node.js, который используется для установки библиотек и пакетов.**

Команда для установки:

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

npm install react-router-dom

Этой командой вы добавите библиотеку react-router-dom в зависимости вашего проекта. Она будет установлена в папку node\_modules.

Проверка успешной установки:

Откройте файл package.json в корневой директории вашего проекта.

В разделе dependencies вы должны увидеть react-router-dom с указанной версией, например:



## ⭐ Задание для закрепления

Создание проекта с использованием Create React App и настройка React Router.

### 1. Создание нового React проекта

Перейдите в директорию, где вы хотите создать новый проект, и выполните команду для создания нового React-приложения с помощью `create-react-app`.

После завершения создания проекта, перейдите в его директорию.

### 2. Запуск проекта

Запустите проект с помощью команды для проверки, что все работает корректно. Откройте браузер и перейдите по указанному адресу, чтобы увидеть главное окно вашего нового React приложения.

### 3. Установка библиотеки React Router

В корневой директории вашего проекта выполните команду для установки библиотеки `react-router-dom`. ( `npm install react-router-dom` )

### 4. Проверка установки:

Откройте файл `package.json` в корневой директории вашего проекта и убедитесь, что `react-router-dom` добавлен в зависимости.

# Настройка проекта

## 1. Создание структуры проекта:

Обычная структура проекта для React-приложения может выглядеть следующим образом:



## 2. Настройка основного файла приложения:

В файле src/index.js вам нужно обернуть ваше приложение в компонент BrowserRouter, чтобы маршруты стали доступными во всем приложении. Пример:

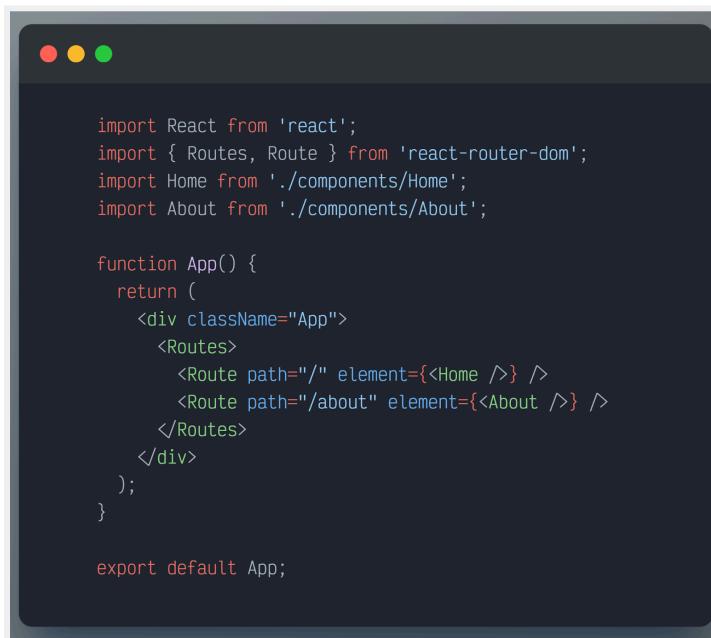
```
import React from 'react';
import ReactDOM from 'react-dom/client';
import { BrowserRouter } from 'react-router-dom';
import App from './App';

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

A screenshot of a Mac OS X terminal window. The title bar says "Terminal". The window displays the code for the main application entry point, src/index.js. It imports React, ReactDOM, and BrowserRouter from their respective packages. It then creates a root ReactDOM.createRoot instance and renders it,包裹着一个BrowserRouter组件，从而使得所有路由都能在应用中访问。

### 3. Создание компонентов для маршрутов:

Создаём файл `src/App.js` и добавляем базовую структуру роутинга:



```
import React from 'react';
import { Routes, Route } from 'react-router-dom';
import Home from './components/Home';
import About from './components/About';

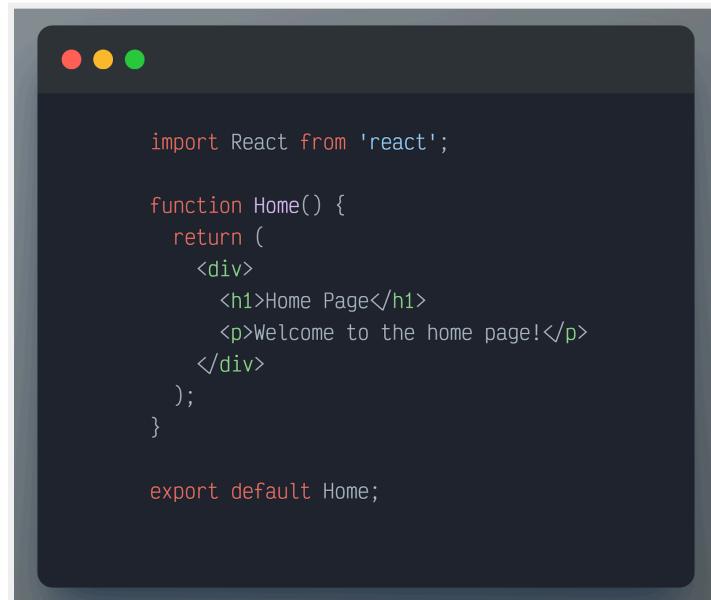
function App() {
  return (
    <div className="App">
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
      </Routes>
    </div>
  );
}

export default App;
```

### 4. Создание компонентов для маршрутов (например, Home и About):

В папке `src/components` создаем файлы `Home.js` и `About.js`:

`src/components/Home.js`:

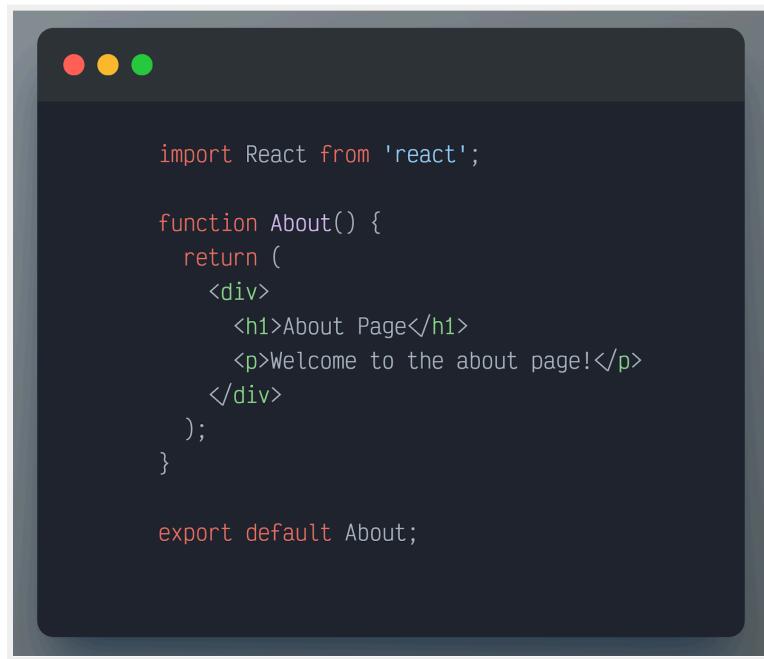


```
import React from 'react';

function Home() {
  return (
    <div>
      <h1>Home Page</h1>
      <p>Welcome to the home page!</p>
    </div>
  );
}

export default Home;
```

src/components/About.js:



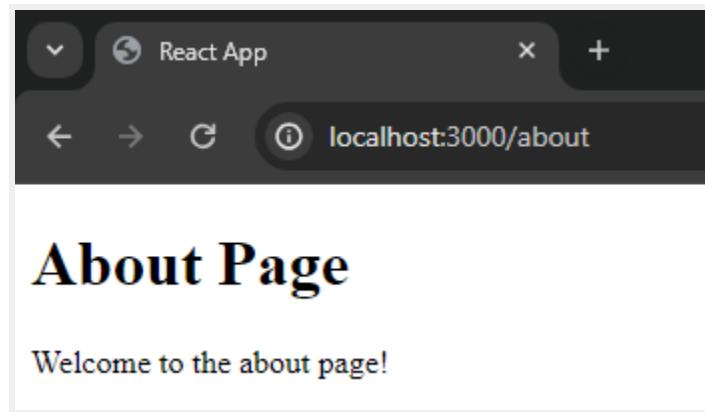
```
import React from 'react';

function About() {
  return (
    <div>
      <h1>About Page</h1>
      <p>Welcome to the about page!</p>
    </div>
  );
}

export default About;
```

### Заключение этапа установки

Теперь наше React-приложение настроено для работы с роутингом при помощи библиотеки react-router-dom.



## ☆ Задание для закрепления

Настройка React-приложения с использованием библиотеки React Router:

1. Создайте папку components в директории src.

В папке components создайте два файла: Home.js и About.js:

В Home.js создайте функциональный компонент, отображающий заголовок и текст главной страницы.

В About.js создайте функциональный компонент, отображающий заголовок и текст страницы "О нас".

2. Настройка index.js

Настройте файл index.js так, чтобы в приложении работал роутинг, а именно, импортируйте BrowserRouter из 'react-router-dom' и оберните <App /> в <BrowserRouter>.

3. Настройка основного файла приложения (App.js):

Откройте файл src/App.js и настройте маршруты с использованием компонентов Routes, Route из react-router-dom:

Импортируйте компоненты Home и About.

Создайте навигационное меню с помощью Link, чтобы можно было переходить на соответствующие страницы.

Настройте маршруты для отображения компонентов Home и About.

4. Запуск проекта для проверки работы маршрутов:

Запустите проект снова и проверьте, что маршруты работают корректно, перейдя по различным URL вашего приложения, соответствующим компонентам Home и About.

После выполнения этого задания у вас будет настроенное React-приложение с базовым роутингом, использующим библиотеку react-router-dom. Это будет отличной отправной точкой для дальнейшего изучения более сложных аспектов роутинга в React.

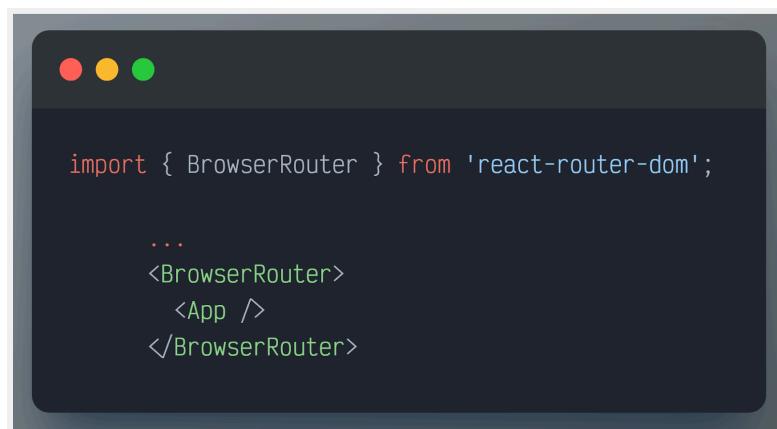
## Основные компоненты в React Router 6

Разберем основные компоненты, которые предоставляет библиотека React Router 6 для настройки маршрутов в приложении.

1. <BrowserRouter>: Основной оберывающий компонент

Это компонент, который обеспечивает доступ ко всем функциям роутинга в рамках своего контекста.

Пример использования:

A screenshot of a macOS application window, likely a terminal or code editor. The window has a dark theme with red, yellow, and green close buttons at the top. The main area contains the following code:

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

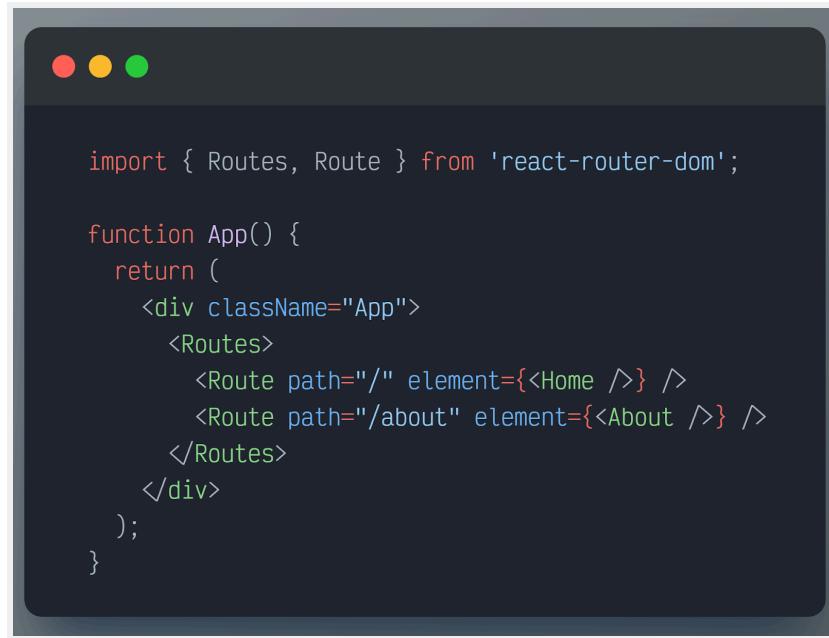
...
<BrowserRouter>
  <App />
</BrowserRouter>
```

В данном примере, App компонент будет иметь доступ ко всем функциям и компонентам React Router.

2. <Routes>: Контейнер для маршрутов

Это контейнер, который рассматривает каждый вложенный <Route> и рендерит первый совпадающий маршрут.

Пример использования:

A screenshot of a terminal window with a dark background. At the top, there are three small colored circles (red, yellow, green) representing window control buttons. Below them is a command-line interface (CLI) prompt. The code displayed is a JavaScript file using the syntax of the 'react-router-dom' library. It defines a function named 'App' which returns a component structure. This structure includes a 'div' element with the class name 'App', followed by a '' component. Inside '', there are two '' components: one for the root path '/' pointing to the 'Home' component, and another for the path '/about' pointing to the 'About' component. The code ends with a closing brace for the 'App' function.

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

function App() {
  return (
    <div className="App">
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
      </Routes>
    </div>
  );
}


```

Здесь <Routes> содержит два маршрута: для корневого ("") и для маршрута "/about", каждый из которыхрендерит соответствующий компонент (Home и About).

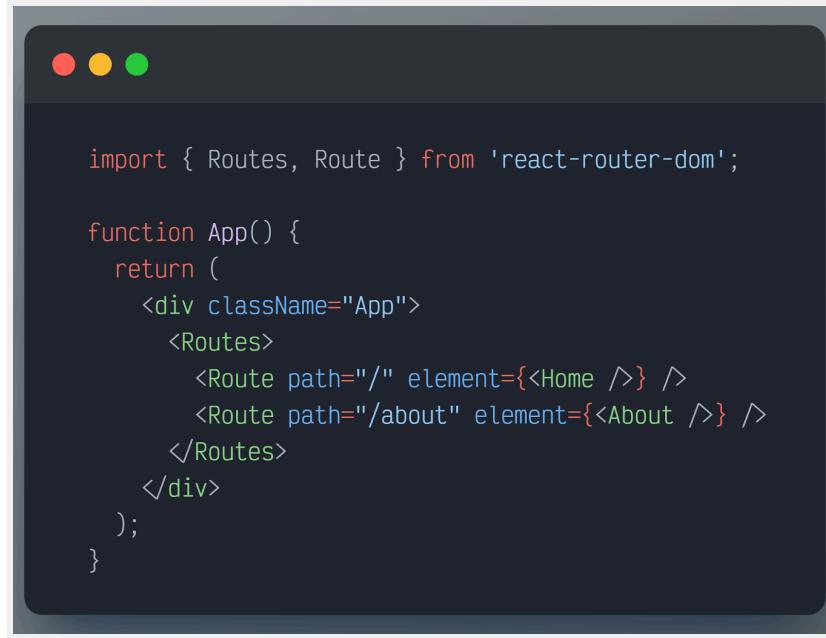
### 3. <Route>: Определение маршрута и его привязка к компоненту

Используется для определения маршрута. Он указывает, какой компонент должен быть отображен, когда URL совпадает с указанным путем.

Атрибуты:

- **path:** определяет путь URL, который должен совпадать для рендеринга данного компонента.
- **element:** указывает компонент, который будет рендериться при совпадении пути.

Пример использования:



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

function App() {
  return (
    <div className="App">
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
      </Routes>
    </div>
  );
}


```

В данном примере `path="/"` рендерит компонент `Home`, когда пользователь находится на главной странице. `path="/about"` рендерит компонент `About`, когда пользователь находится на странице "О нас".

Поговорим о навигации внутри React-приложения с использованием компонентов `Link` и `NavLink` из библиотеки `react-router-dom`. Эти компоненты позволят вам создавать навигационные меню и активные ссылки для удобного перехода между различными страницами приложения.

Использование `<Link>` для навигации между страницами:



**<Link> заменяет обычные HTML-ссылки (<a>) в React-приложениях и предназначен для навигации без перезагрузки страницы.**

Этот компонент автоматически предотвращает полное обновление страницы, что делает пользовательский опыт более плавным.



```
import { Link } from 'react-router-dom';
...
<Link to="/">Home</Link>
...
```

Импортируем компонент Link из react-router-dom.

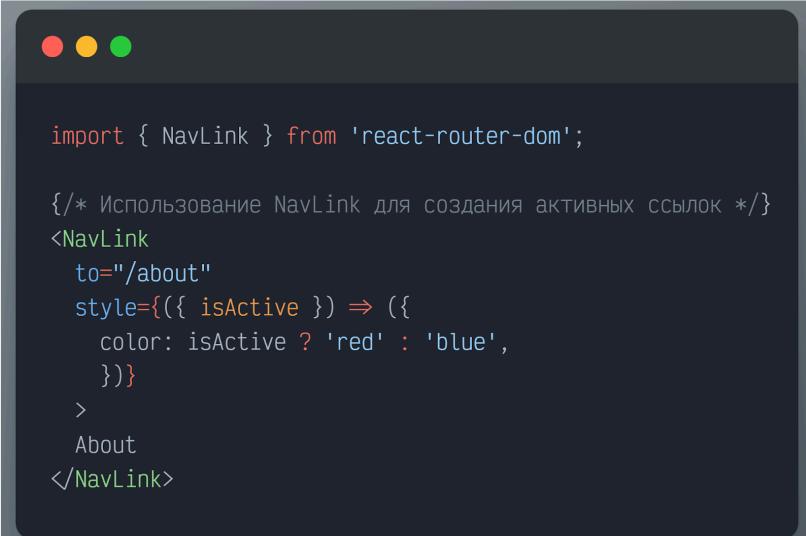
Используем Link в навигационном меню, указывая атрибут to для определения маршрута, на который нужно перейти.

Использование <NavLink> для создания активных ссылок и стилей:



**<NavLink> — это специальный вид Link, который позволяет автоматически добавлять классы к активным ссылкам для стилизованного отображения.**

Этот компонент полезен для создания элементов навигации, где требуется отображать активное состояние текущего маршрута.



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

/* Использование NavLink для создания активных ссылок */
<NavLink
  to="/about"
  style={({ isActive }) => ({
    color: isActive ? 'red' : 'blue',
  })}
>
  About
</NavLink>
```

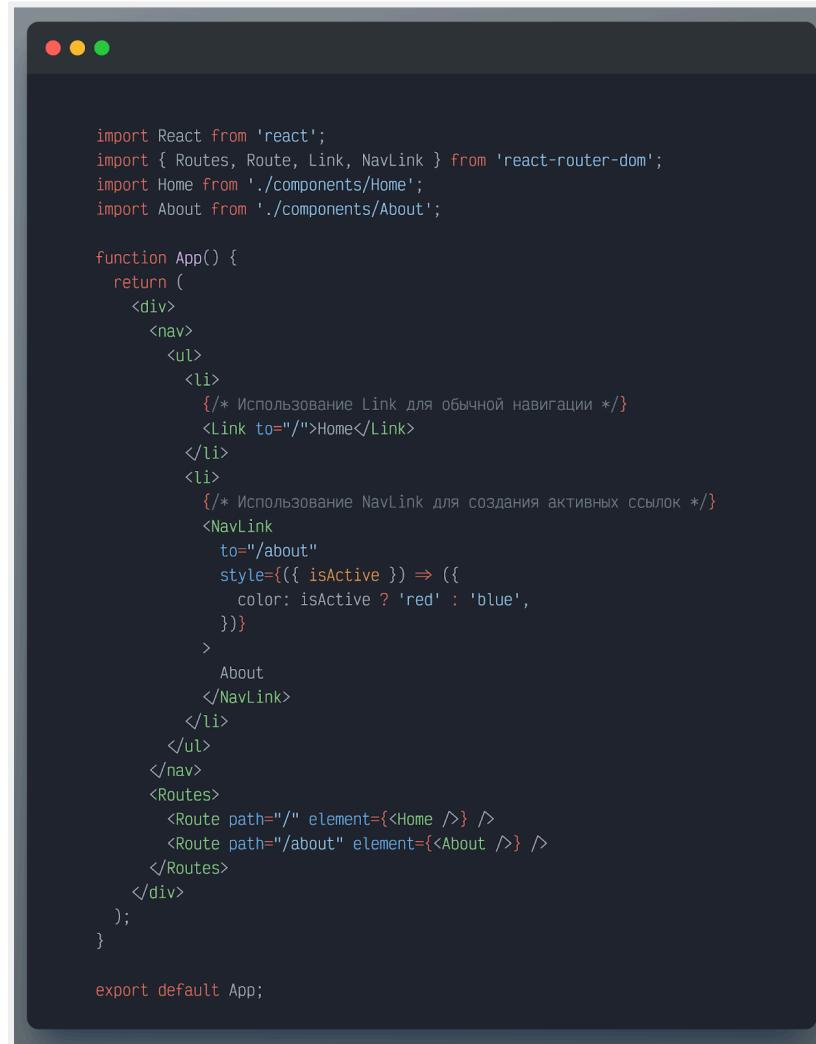
Импортируем компонент NavLink из react-router-dom.

Используем NavLink в навигационном меню, добавляя атрибут to для определения маршрута и className (или style) для определения стилей активного состояния.

Пример меню навигации с использованием компонентов Link и NavLink:

1. Создание базового навигационного меню:

Откроем файл src/App.js и изменим навигационное меню следующим образом:



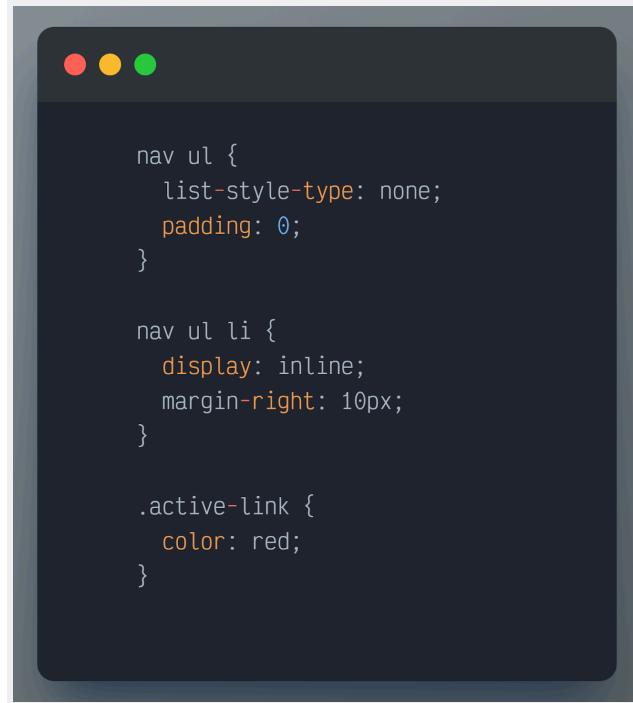
```
import React from 'react';
import { Routes, Route, Link, NavLink } from 'react-router-dom';
import Home from './components/Home';
import About from './components/About';

function App() {
  return (
    <div>
      <nav>
        <ul>
          <li>
            {/* Использование Link для обычной навигации */}
            <Link to="/">Home</Link>
          </li>
          <li>
            {/* Использование NavLink для создания активных ссылок */}
            <NavLink
              to="/about"
              style={({ isActive }) => ({{
                color: isActive ? 'red' : 'blue',
              }})}
            >
              About
            </NavLink>
          </li>
        </ul>
      </nav>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
      </Routes>
    </div>
  );
}

export default App;
```

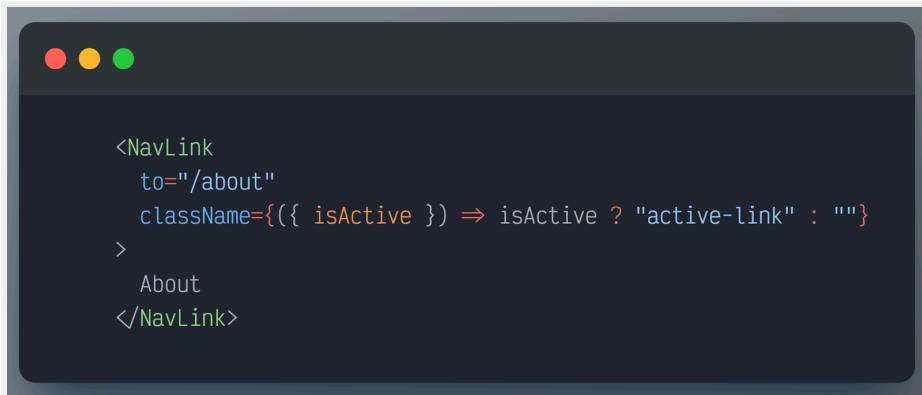
2. Стилизация меню навигации:

Мы можем добавить стили для навигационного меню, чтобы лучше отображать активные ссылки. Откроем файл src/App.css (создаем его, если он отсутствует) и добавляем следующие стили:



```
nav ul {  
    list-style-type: none;  
    padding: 0;  
}  
  
nav ul li {  
    display: inline;  
    margin-right: 10px;  
}  
  
.active-link {  
    color: red;  
}
```

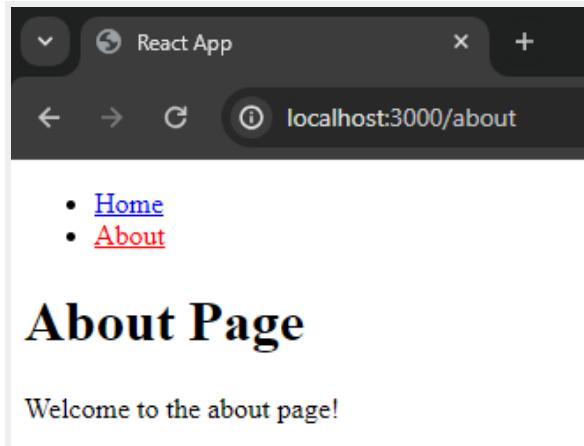
Затем изменим NavLink в src/App.js, чтобы использовать класс active-link:



```
<NavLink  
    to="/about"  
    className={({ isActive }) => isActive ? "active-link" : ""}  
>  
    About  
</NavLink>
```

### 3. Запуск и проверка приложения:

Запустим проект, протестируем навигацию, перейдя по ссылкам "Home" и "About". Убедимся, что при выборе ссылки "About" её цвет меняется на красный, указывая, что это активная ссылка.



Вы узнали, как создавать обычные навигационные ссылки и активные ссылки с уникальными стилями, что значительно улучшает пользовательский интерфейс и опыт работы с приложением.

## ☆ Задание для закрепления

Доработка существующего компонента App вашего React-приложения, чтобы добавить в него навигационное меню с помощью компонентов Link и NavLink.

### 1. Импорт компонентов Link и NavLink

В файле src/App.js импортируйте компоненты Link и NavLink из библиотеки react-router-dom.

### 2. Создание навигационного меню

В компоненте App создайте навигационное меню. Используйте компонент Link для создания обычных навигационных ссылок и компонент NavLink для создания ссылок с активным состоянием.

Настройте атрибут to для каждого Link и NavLink, чтобы определить маршрут, на который должна вести ссылка.

Для NavLink используйте шаблонные строки или функции для присвоения классов активной ссылке, что позволит вам применять стили к активным ссылкам.

### 3. Добавление стилей для активных ссылок

Создайте или откройте файл стилей (например, App.css), и добавьте стили, которые будут применяться к активным ссылкам. Убедитесь, что эти стили применяются к классам или свойствам, установленным в компоненте NavLink.

### 4. Тестирование работоспособности:

Запустите проект, проверьте работу навигации, переходя по различным ссылкам в вашем навигационном меню. Убедитесь, что ссылки работают корректно и активные ссылки отображаются с применением заданных стилей.

## ХУКИ И ИХ ВАЖНОСТЬ

Вспомним, что такое хуки.



**Хуки (hooks) — это специальные функции в React, которые позволяют использовать состояние и другие возможности React в функциональных компонентах.**

Они были представлены в версии React 16.8 и служат для упрощения и улучшения функциональных компонентов.

Основные концепции хуков:

- Управление состоянием: Раньше состояние можно было хранить только в классовых компонентах с помощью `this.state`. Теперь, используя `useState`, можно управлять состоянием прямо внутри функциональных компонентов.
- Побочные эффекты: С помощью `useEffect` можно выполнять побочные эффекты, такие как получение данных с сервера, подписка на события или ручная манипуляция с DOM.
- Контекст: `useContext` позволяет функциональным компонентам подписываться на изменения контекста, что делает передачу данных через дерево компонентов более удобной и прямой.

Почему хуки полезны?

- Читаемость и простота: Функциональные компоненты с хуками в большинстве случаев проще для понимания и поддержки.
- Переиспользование: Логику состояния и побочные эффекты можно повторно использовать через пользовательские хуки (custom hooks).
- Без классов: Устраняется необходимость использования классов, что упрощает синтаксис и делает код более лаконичным.

React Router является популярной библиотекой для управления маршрутизацией в приложениях React. Версия библиотеки 5 и выше внедрила поддержку hooks, которая позволяет разработчикам работать с маршрутизацией более интуитивно и эффективно.

Преимущества использования хуков в react-router-dom:

- Простота интеграции: Позволяет легко интегрировать возможности маршрутизации внутри функциональных компонентов без необходимости использования классов или дополнительных оберток.

- Доступ к информации маршрута: Предоставляет функциональным компонентам доступ к информации о текущем маршруте, параметрах и состоянии через простые вызовы функций.
- Гибкость навигации: Хуки позволяют выполнять программную навигацию и получать данные о местоположении и параметрах URL, что делает роутинг более динамичным и гибким.

Основные хуки, предоставляемые react-router-dom:

- `useNavigate`: Позволяет выполнять программную навигацию. С его помощью можно перенаправлять пользователей на разные страницы на основе логики приложения.
- `useLocation`: Предоставляет информацию о текущем URL, включая путь и параметры запроса, что помогает определять текущий контекст страницы.
- `useParams`: Позволяет получать параметры маршрута, такие как идентификаторы или другие динамические части URL, что облегчает работу с динамическими данными в маршрутах.
- `useRoutes`: Позволяет создавать маршруты на основе конфигурации прямо внутри компонента, поддерживая динамическое формирование маршрутов.
- `useMatch`: Используется для проверки, соответствует ли текущий URL определенному маршруту, что полезно для условного рендеринга или получения данных.
- `useSearchParams`: Позволяет работать с параметрами поиска в URL, что упрощает получение и изменение query-параметров.

Эти хуки предоставляют функциональные возможности, которые делают управление маршрутизацией в React-приложениях более удобным, гибким и мощным, значительно упрощая разработку и поддержку кода.

## useNavigate()



useNavigate — это хук, предоставляемый библиотекой react-router-dom, который позволяет выполнять программную навигацию, то есть перемещаться между страницами приложения с помощью кода, а не только через пользовательские действия, такие как нажатие на ссылки.

Программная навигация бывает полезна в различных сценариях:

- После выполнения действий: Например, после успешной аутентификации или отправки формы, вы можете автоматически перенаправить пользователя на другую страницу.
- Условная навигация: Можно перемещать пользователя в зависимости от выполнения каких-то условий (например, если пользователь не авторизован, перенаправить его на страницу логина).
- Упрощение пользовательского интерфейса: Иногда проще и логичнее инициировать переход с использованием кода, чем полагаться исключительно на ссылки.

Как использовать useNavigate:

1. Импорт хуков: Сначала необходимо импортировать нужный хук из библиотеки react-router-dom.

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

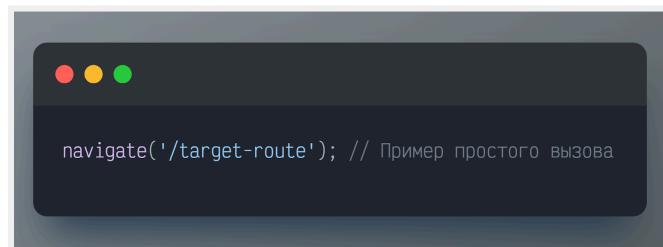
```
import { useNavigate } from 'react-router-dom';
```

2. Получение функции navigate: Внутри вашего компонента вызываем useNavigate для получения функции navigate.



```
const navigate = useNavigate();
```

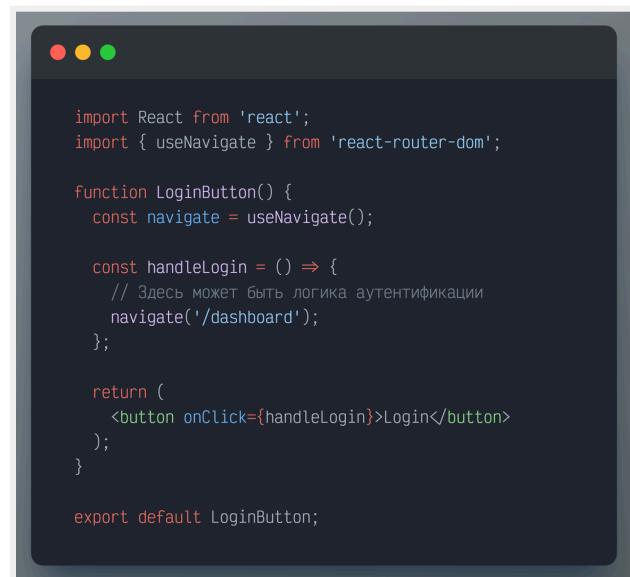
3. Использование `navigate` для перемещения: Используйте полученную функцию `navigate` для программного перенаправления пользователя на другой маршрут.



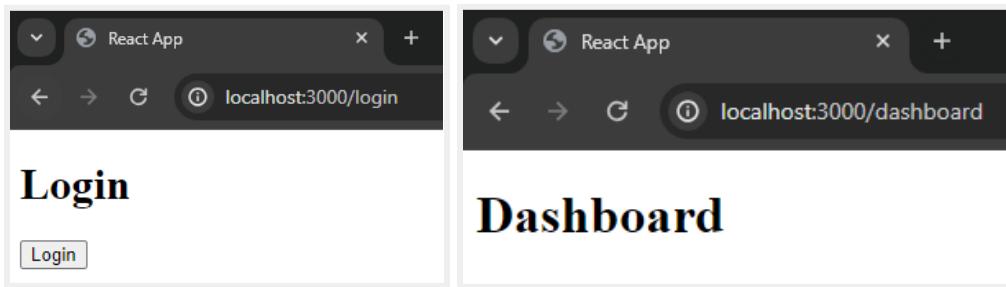
```
navigate('/target-route'); // Пример простого вызова
```

Пример с кнопкой "Login":

В этом примере при нажатии на кнопку "Login" выполняется переход на страницу "Dashboard":

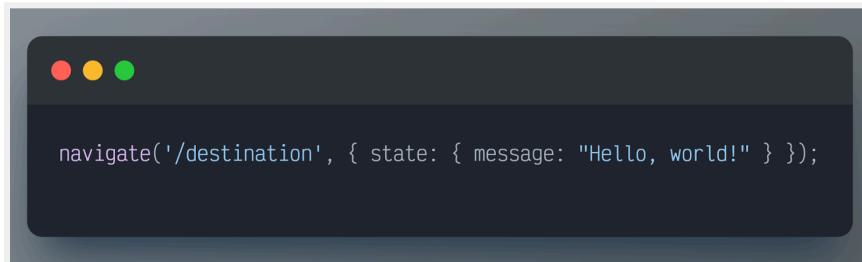


Результат:



Особенности useNavigate:

Передача состояния: Вы также можете передать состояние вместе с навигацией. Это полезно для передачи данных между страницами.



`navigate('/destination', { state: { message: "Hello, world!" } });`

Замена истории: По умолчанию `navigate` добавляет новый элемент в историю браузера. Если нужно заменить текущий элемент (что аналогично `window.location.replace`), передайте второй аргумент `{ replace: true }`.



`navigate('/new-url', { replace: true });`

Таким образом, `useNavigate` предоставляет мощные и гибкие возможности для выполнения программной навигации в ваших React-приложениях, что делает процесс управления маршрутами более удобным и интуитивно понятным.

## ⭐ Задание для закрепления

Создание простого React-приложения с использованием `useNavigate` из `react-router-dom`:

### 1. Создание нового проекта

Откройте терминал или командную строку.

Создайте новый проект с помощью Create React App.

Перейдите в директорию проекта.

### 2. Установка библиотеки react-router-dom

Установите `react-router-dom`: `npm install react-router-dom`.

### 3. Настройка начального роутинга

Создайте файл `App.js` и настройте его так, чтобы он обрабатывал маршруты. Вместо конкретного кода, создайте каркас для маршрутизации (`Routes` и `Route`).

### 4. Создание компонентов

Создайте два новых компонента: `Home` и `About`.

В компоненте `Home` добавьте кнопку, которая будет использовать `useNavigate` для перенаправления на страницу `About`.

В компоненте `About` добавьте простой заголовок или текст для отображения.

### 5. Внедрение роутинга в корневой файл

Откройте файл `index.js` и оберните корневой компонент (например, `App`) в `BrowserRouter`.

### 6. Запуск приложения и тестирование

Убедитесь, что все необходимые маршруты и компоненты соответствующим образом связаны.



Запустите приложение.

Откройте браузер и перейдите на главную страницу вашего приложения.

Проверьте, что на странице Home отображается кнопка "Go to About". При нажатии вы должны быть перенаправлены на страницу About.

## useLocation()



**useLocation** — это хук из библиотеки `react-router-dom`, который позволяет получать информацию о текущем местоположении (URL) в вашем React-приложении.

Этот хук возвращает объект `location`, содержащий информацию о текущем пути, включая `pathname`, `search` и `hash`.

Пример: Отображение текущего URL

В этом примере мы создадим простой компонент, который будет отображать текущий URL, используя `useLocation`.

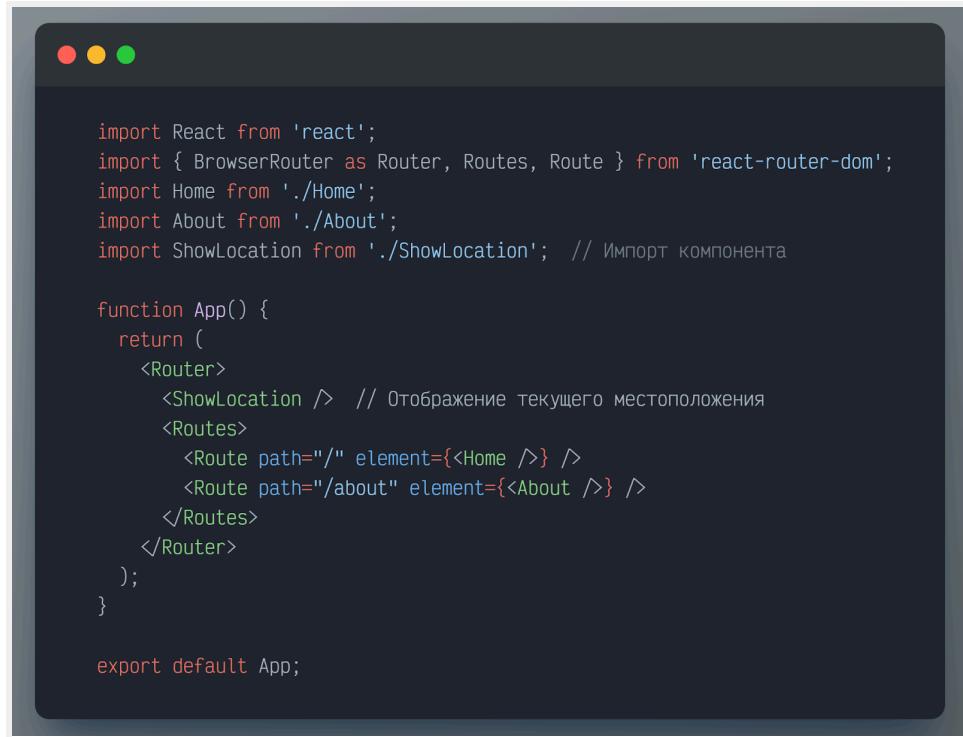
```
import React from 'react';
import { useLocation } from 'react-router-dom';

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

  return (
    <div>
      <h2>Current URL: {location.pathname}</h2>
    </div>
  );
}

export default ShowLocation;
```

В основном файле приложения (App.js) вы можете добавить этот компонент, чтобы увидеть текущий URL на странице:

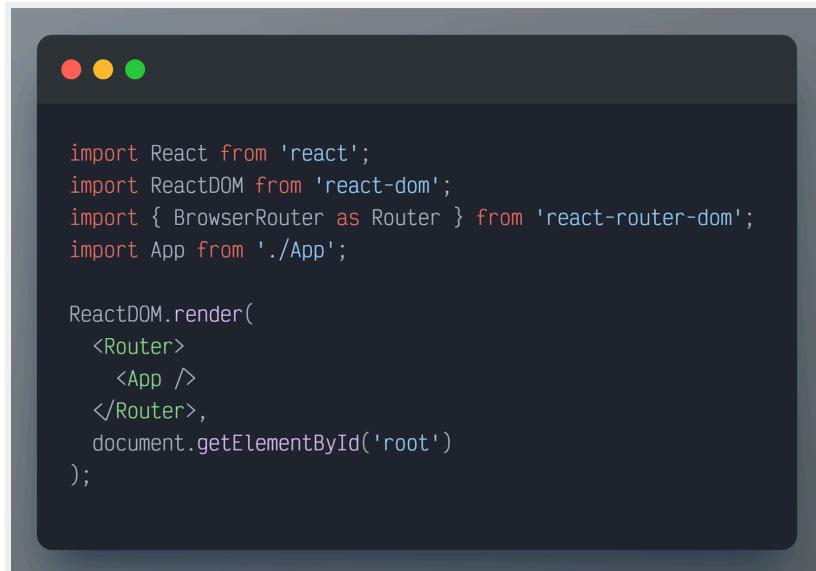


```
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import Home from './Home';
import About from './About';
import ShowLocation from './ShowLocation'; // Импорт компонента

function App() {
  return (
    <Router>
      <ShowLocation /> // Отображение текущего местоположения
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
      </Routes>
    </Router>
  );
}

export default App;
```

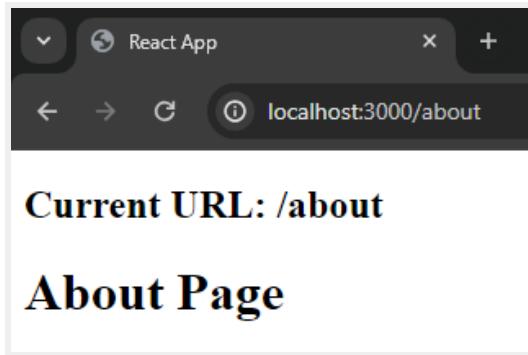
Код index.js:



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

ReactDOM.render(
  <Router>
    <App />
  </Router>,
  document.getElementById('root')
);
```

Результат:



`useLocation` — это инструмент для получения и использования информации о текущем маршруте в вашем React-приложении. Он позволяет вам легко получать текущий путь, параметры строки запроса и другие детали текущего местоположения, что помогает создавать более динамичные и интерактивные веб-приложения.

## ⭐ Задание для закрепления

Создание React-приложения с использованием useLocation.

### 1. Настройка роутинга

Откройте файл index.js.

Импортируйте BrowserRouter из react-router-dom.

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

ReactDOM.render(
  <Router>
    <App />
  </Router>,
  document.getElementById('root')
);
```

Оберните компонент App в BrowserRouter.

### 2. Настройка файла App.js

Импортируйте компоненты Routes и Route из react-router-dom.

Создайте маршрутизацию для нескольких страниц: Home, Profile, Settings.

```
JavaScript
import React from 'react';
import { Routes, Route } from 'react-router-dom';
import Home from './components/Home';
import Profile from './components/Profile';
```

```
import Settings from './components/Settings';

function App() {
  return (
    <div>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/profile" element={<Profile />} />
        <Route path="/settings" element={<Settings />} />
      </Routes>
    </div>
  );
}

export default App;
```

### 3. Создание страниц

Создайте файл Home.js.

Импортируйте useLocation из react-router-dom.

Используйте useLocation для получения текущего пути и отображения его на странице.

Повторите шаги для создания страниц Profile.js и Settings.js.

```
JavaScript
import React from 'react';
import { useLocation } from 'react-router-dom';

function Home() {
  const location = useLocation();
```

```
return (
  <div>
    <h1>Home Page</h1>
    <p>Current URL: {location.pathname}</p>
  </div>
);

}

export default Home;
```

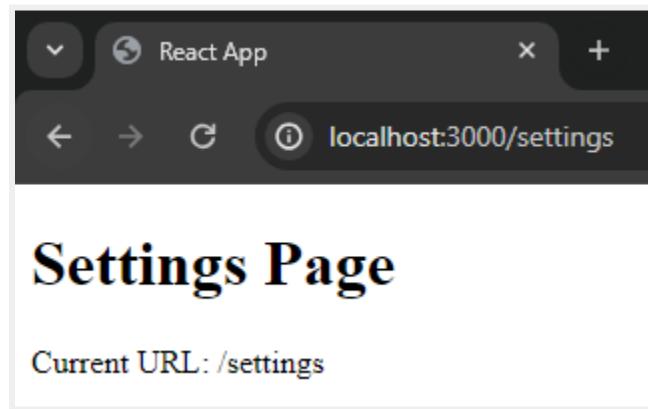
#### 4. Запуск приложения

Убедитесь, что все необходимые файлы созданы и правильно импортированы.

Вернитесь в корневую директорию приложения.

Запустите приложение с помощью команды npm start.

Откройте браузер и проверьте приложение. Перейдите по разным URL (/ , /profile, /settings) и убедитесь, что текущий путь отображается на каждой странице.



1.

## useParams()



**useParams** — это хук из `react-router-dom`, который позволяет вам получать параметры текущего маршрута.



**Параметры маршрута** — это динамические сегменты URL, которые могут содержать значения, определенные пользователем.

Эти параметры полезны, когда вам нужно передать данные через URL, такие как идентификаторы пользователей или статьи.

`useParams` возвращает объект с ключами, соответствующими именам параметров маршрута, и значениями, соответствующими значениям этих параметров в текущем URL.

Пример использования:

Предположим, у вас есть приложение, в котором вы хотите отображать профиль пользователя на основе его идентификатора, переданного через URL. Например, URL для страницы профиля пользователя может выглядеть так: `/profile/:userId`.

Вот как можно использовать `useParams` для получения параметров маршрута:

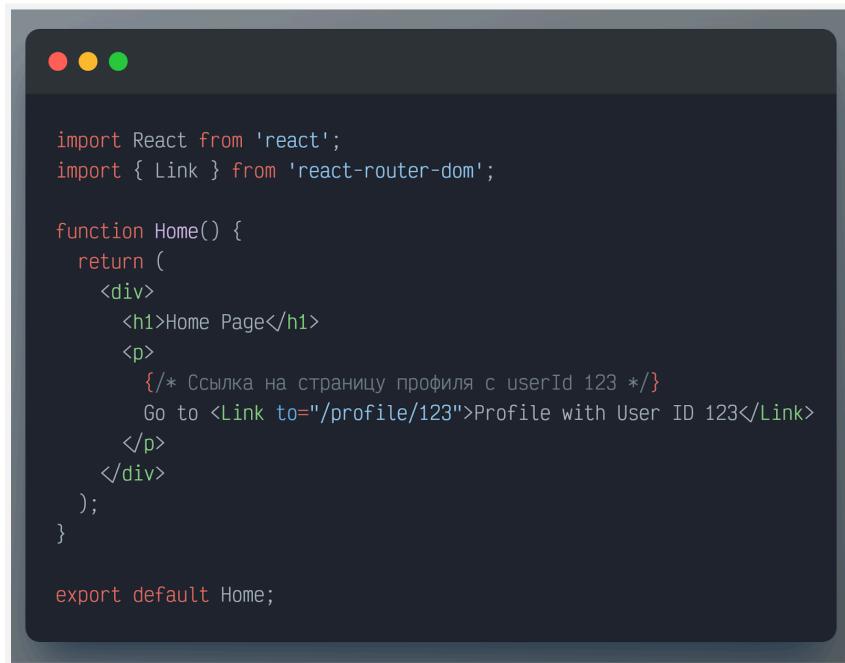
App.js

```
import React from 'react';
import { Routes, Route } from 'react-router-dom';
import Home from './Home';
import Profile from './Profile';

function App() {
  return (
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/profile/:userId" element={<Profile />} />
    </Routes>
  );
}

export default App;
```

## Home.js

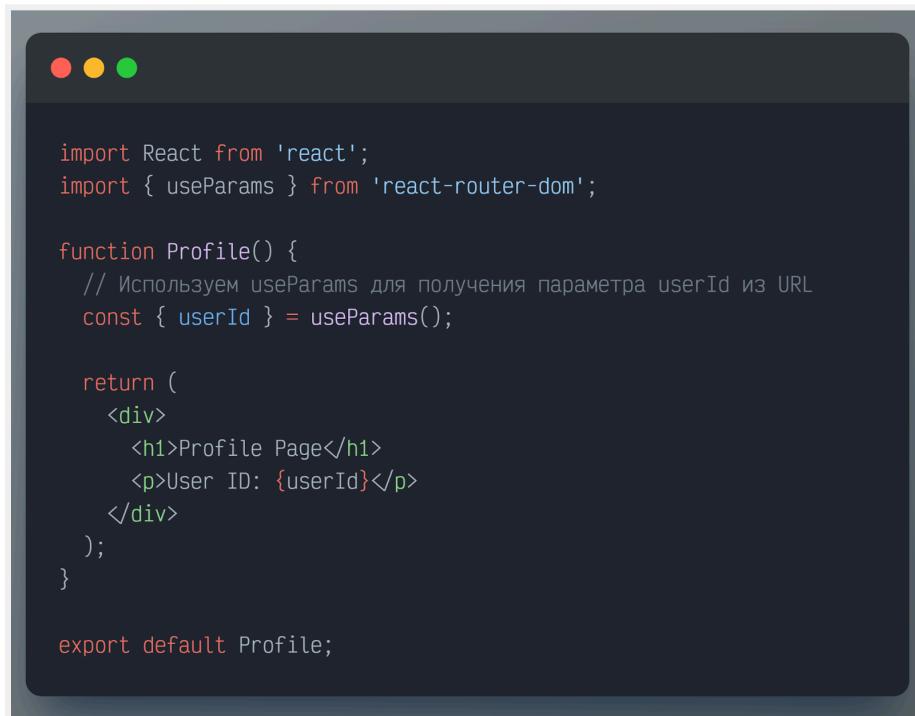


```
import React from 'react';
import { Link } from 'react-router-dom';

function Home() {
  return (
    <div>
      <h1>Home Page</h1>
      <p>
        {/* Ссылка на страницу профиля с userId 123 */}
        Go to <Link to="/profile/123">Profile with User ID 123</Link>
      </p>
    </div>
  );
}

export default Home;
```

## Profile.js



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

function Profile() {
  // Используем useParams для получения параметра userId из URL
  const { userId } = useParams();

  return (
    <div>
      <h1>Profile Page</h1>
      <p>User ID: {userId}</p>
    </div>
  );
}

export default Profile;
```

## 1. Создаем маршруты для приложения:

В index.js, оборачиваем App в BrowserRouter.

В App.js, создаем маршруты для / и /profile/:userId.

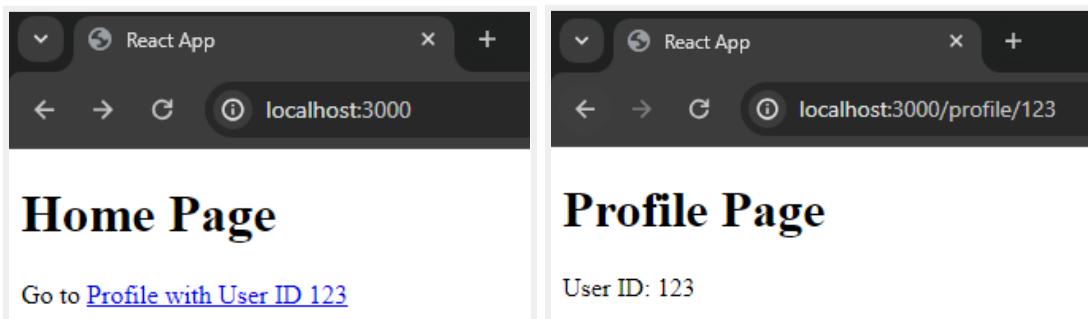
## 2. Компонент Home:

В Home.js, создаем базовый компонент с ссылкой на страницу профиля. Для этого используем компонент Link из react-router-dom.

## 3. Компонент Profile:

В Profile.js, используем хук useParams для получения параметра userId из URL.

Отображаем значение userId на странице.



Используя такой подход, мы можем легко создавать динамические маршруты и передавать параметры через URL, что значительно упрощает работу с различными ресурсами в нашем приложении.

## ⭐ Задание для закрепления

Использование хука useParams в React-приложении.

1. Настройте маршруты в вашем приложении

Откройте файл index.js.

Импортируйте BrowserRouter из react-router-dom и оберните компонент App в BrowserRouter.

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

ReactDOM.render(
  <Router>
    <App />
  </Router>,
  document.getElementById('root')
);
```

В App.js настройте маршруты для страниц Home и User.

```
JavaScript
import React from 'react';
import { Routes, Route } from 'react-router-dom';
import Home from './Home';
import User from './User';

function App() {
  return (
    <div>
```

```
<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/user/:userId" element={<User />} />
</Routes>
</div>
);
}

export default App;
```

## 2. Создайте файл Home.js для страницы Home.

На странице Home.js создайте ссылку на страницу пользователя с заданным ID.

JavaScript

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

function Home() {
  return (
    <div>
      <h1>Home Page</h1>
      <p>
        {/* Ссылка на страницу пользователя с ID 1 */}
        Go to <Link to="/user/1">User with ID 1</Link>
      </p>
    </div>
  );
}

export default Home;
```

## 3. Создайте файл User.js для страницы пользователя.

В User.js используйте хук useParams для получения параметра userId из URL.

```
JavaScript
import React from 'react';
import { useParams } from 'react-router-dom';

function User() {
    // Используем useParams для получения параметра userId из URL
    const { userId } = useParams();

    return (
        <div>
            <h1>User Page</h1>
            <p>User ID: {userId}</p>
        </div>
    );
}

export default User;
```

#### 4. Запустите приложение

Убедитесь, что все необходимые файлы созданы и правильно импортированы.

Откройте браузер и перейдите по адресу <http://localhost:3000>.

На главной странице (/), вы увидите ссылку на страницу пользователя с ID 1. Нажмите на ссылку.

Перейдите на страницу User (/user/1) и проверьте, что отображается соответствующий ID пользователя.