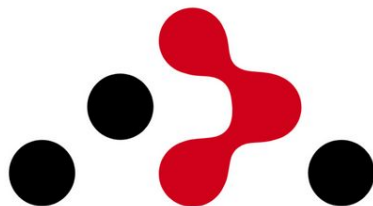


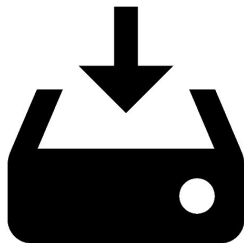
Тема занятия: React: React Router

React Router - это библиотека для управления маршрутизацией в веб-приложениях, разработанных с использованием библиотеки React. Она предоставляет средства для определения и отслеживания маршрутов (URL-адресов) в вашем приложении и для отображения соответствующих компонентов в зависимости от текущего URL.



Установка

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



Основные концепции

- **BrowserRouter** - настройка роутера:

Основной компонент, который оборачивает ваше приложение. Он используется для создания контекста, который позволяет другим компонентам React Router взаимодействовать с браузерной историей.

index.js

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

ReactDOM.render(
  <Router>
    {/* Ваше приложение */}
  </Router>,
  document.getElementById('root')
);
```

Основные концепции

• Определение маршрутов

Чтобы определить маршруты, нужно определить компонент Route для каждого маршрута в приложении, а затем поместить все эти компоненты Route в один компонент Routes.

Всякий раз, когда ваш URL-адрес изменяется, React Router будет просматривать маршруты, определенные в вашем компоненте Routes, и он будет отображать содержимое в пропсе element роута Route, который имеет path, соответствующий URL-адресу. В приведенном выше примере, если бы наш URL-адрес был /books, то отображался бы компонент BookList.

```
import { Route, Routes } from "react-router-dom"
import { Home } from "../Home"
import { BookList } from "../BookList"

export function App() {
  return (
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/books" element={<BookList />} />
    </Routes>
  )
}
```

Обычно это делается на верхнем уровне приложения, например в компоненте App

Основные концепции

• Управление навигацией

Последним шагом к React Router является обработка навигации. Обычно в приложении вы перемещаетесь с помощью тегов `<a>`, но React Router использует свой собственный кастомный компонент `Link` для обработки навигации. `Link` представляет собой просто оболочку вокруг тега `<a>`, которая помогает обеспечить правильную обработку всей маршрутизации

```
import { Route, Routes, Link } from "react-router-dom"
import { Home } from "../Home"
import { BookList } from "../BookList"

export function App() {
  return (
    <>
      <nav>
        <ul>
          <li><Link to="/">Home</Link></li>
          <li><Link to="/books">Books</Link></li>
        </ul>
      </nav>

      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/books" element={<BookList />} />
      </Routes>
    </>
  )
}
```


вместо href в ссылке

Основные концепции

Примечания

- Можно создать маршрут, на который будет выполнен переход, если не будет никаких других совпадений. Это делает его идеальным для таких вещей, как страница 404. Маршрут, содержащий *, также будет менее конкретным, чем все остальное, поэтому вы никогда случайно не сопоставите маршрут *, когда другой маршрут также совпал бы.

```
<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/books" element={<BookList />} />
  <Route path="/books/:id" element={<Book />} />
  <Route path="/books/new" element={<NewBook />} />
  <Route path="*" element={<NotFound />} />
</Routes>
```



- Вы можете расположить маршруты в любом порядке, и маршрутизатор автоматически определит лучший маршрут для текущего URL-адреса.

Дополнительные возможности

Навигация по ссылкам

Навигация по ссылкам - это самая простая и распространенная форма навигации., однако ссылки внутри Link могут быть сложнее

Представьте, что мы находимся на странице `/books/3`. Куда приведут нас показанные ниже ссылки?

```
1 <Link to="/">Home</Link>
2 <Link to="..">Back</Link>
3 <Link to="edit">Edit</Link>
```

1 - приведет к / маршруту, то есть странице Home

2 - приведет к маршруту /books, так как это относительная ссылка, которая поднимается на один уровень вверх от /books/3 к /books.

3 - перейдет на страницу /books/3/edit, так как она добавит путь в конец текущей ссылки, поскольку это относительная ссылка.

Дополнительные возможности

NavLink

Компонент **NavLink** работает точно так же, как компонент **Link**, но он предназначен специально для отображения активных состояний ссылок, например, в панелях навигации. По умолчанию, если проп `to` у **NavLink** совпадает с URL-адресом текущей страницы, к ссылке будет добавлен класс `active`, который можно использовать для стилизации.

```
<NavLink  
  to="/"  
  style={({ isActive }) => ({ color: isActive ? "red" : "black" })}  
>  
  Home  
</NavLink>
```

Дополнительные возможности

Ручная навигация

Иногда нужно вручную перемещаться на основе таких вещей, как отправка формы или отсутствие доступа к определенной странице. Для таких случаев можно использовать компонент `Navigate`.

Компонент `Navigate` — это очень простой компонент, который при визуализации автоматически перенаправляет пользователя на значение пропса `to` компонента, поэтому его обычно добавляют с некоторым условием, например для выполнения редиректа

```
<Route path="/" element={user ? <Home /> : <Register />} />
<Route path="/login" element={user ? <Navigate to="/" replace /> : <Login />} />
<Route path = "/register" element={user ? <Navigate to="/" replace /> : <Register />} />
```

Дополнительные возможности

Хук useNavigation

Хук useNavigation представляет собой хук, который не принимает никаких параметров и возвращает одну функцию navigate, которую вы можете использовать для перенаправления пользователя на определенные страницы. аналогично компоненту Navigation

```
const navigate = useNavigate()
function onSubmit() {
  // Отправка значения формы
  navigate("/books")
}
```

Дополнительные возможности

Данные состояния/местоположения

Вся эта информация в этом случае будет доступна через хук `useLocation`. Использовать этот хук очень просто, так как он возвращает одно значение и не принимает никаких параметров.

```
const location = useLocation()
```

Если у нас есть следующий URL `http://localhost/books?n=32#id` то возвращаемое значение `useLocation` будет выглядеть следующим образом.

```
{
  pathname: "/books",
  search: "?n=32",
  hash: "#id",
  key: "2JH3G3S",
  state: null
}
```