**Введение в React Router**

Раньше переход на другую страницу сайта всегда был связан с загрузкой нового html-документа. При переходе с главной страницы на внутреннюю, браузер запрашивал с сервера новый html-файл и отрисовывал его после получения.

В прошлом спринте вы познакомились с концепцией SPA. В одностраничном приложении есть всего один html-файл, но это не значит, что в нём может быть всего одна страница.

Страниц в SPA может быть сколько угодно. Но загрузки новых html-файлов с сервера не происходит: за перерисовку интерфейса отвечает JS.

Представьте, вы находитесь на главной странице сайта и переходите по ссылке «О проекте». При этом страница перерисовывается, а путь в строке URL меняется на /about. После этого вы перемещаете курсор в URL-строку, стираете /about и вводите /contacts. Интерфейс меняется и на странице отображается контактная информация.

Ничего особенного, кроме того, что всё это время за вами следил JavaScript. JS делал так, чтобы при изменении URL-адреса менялся и интерфейс. И наоборот: чтобы при клике по ссылке менялся адрес в строке URL. И всё это внутри одного html-файла.

В приложениях на «Реакте» для синхронизации URL-строки с тем, что отображается на странице, обычно используют библиотеку React Router. Эта тема как раз про неё.

# Установка React Router

В этом уроке вы научитесь устанавливать и настраивать React Router в проекте.

Создадим новый проект с помощью CRA. Для этого в командной строке введём npx create-react-app emoji-critic. Это будет приложение для любителей эмодзи:

После установки внутри src/ создадим папку с именем components/ и переместим в неё App.js со всеми сопутствующими файлами. Также удалим из проекта всё, что было добавлено по умолчанию. В итоге App.js должен выглядеть так:

Скопировать кодJSX

*// App.js*

import React from 'react';

import Header from './Header';

import './App.css';

function App() {

return (

<div className="App">

<Header />

</div>

);

}

export default App;

Внутри директории components/ создадим новый компонент с именем Dashboard. Это будет простой функциональный компонент, который возвращает JSX с текстом:

Скопировать кодJSX

*// Dashboard.js*

import React from 'react';

import './Dashboard.css';

function Dashboard () {

return (

<div className="dashboard">

<h2>Emoji Critic — всё об эмодзи</h2>

<p>

#1 среди авторов обзоров на эмодзи в этом году!

</p>

</div>

)

}

export default Dashboard;

# Добавление React Router в проект

Чтобы установить библиотеку React Router в проект на «Реакте», откройте главную директорию проекта и введите npm i --save react-router-dom. Эта версия React Router предназначена для маршрутизации в браузерных приложениях.

Первое, с чем вы столкнётесь при работе с React Router, — с компонентом Route. Это главный строительный блок. Чтобы он работал правильно, нужно убедиться, что каждый Route вложен в компонент BrowserRouter.

Компонент BrowserRouter отслеживает историю навигации в процессе работы React Router. Когда пользователь переходит назад или вперёд в браузере, BrowserRouter синхронизирует отображаемый контент.

В файле index.js, который расположен внутри каталога src/, обернём основной компонент App в компонент BrowserRouter:

Скопировать кодJSX

*// index.js*

import React from 'react';

import ReactDOM from 'react-dom';

import { BrowserRouter } from 'react-router-dom'; *// импортируем BrowserRouter*

import App from './components/App';

import './index.css';

*// теперь обернём компонент App в BrowserRouter*

ReactDOM.render(

<React.StrictMode>

<BrowserRouter>

<App />

</BrowserRouter>

</React.StrictMode>,

document.getElementById('root')

);

# Создаём первый компонент Route

Приложение должно отображать компонент Dashboard, когда URL в адресной строке указывает на localhost:3000/.

Для этого импортируем компоненты Route и Dashboard в файл App.js. Внутрь Route вложим компонент, который будет отрисовываться при каждом обращении к URL-пути, передаваемому пропсу path:

Скопировать кодJSX

*// App.js*

import React from 'react';

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

import Dashboard from './Dashboard';

import Header from './Header';

import './App.css';

function App() {

return (

<div className="App">

<Header />

<Route path="/">

<Dashboard />

</Route>

</div>

);

}

export default App;

В нашем примере Dashboard будет отрисован при каждом обращении к корневому URL /.

Убедимся, что всё работает. Запустите проект командой npm run start и откройте в браузере localhost:3000/. Вы увидите, что заголовок Emoji Critic отображается между тегами <h1> правильно. А Dashboard отрисован именно там, где мы разместили компонент Route в JSX-коде.

# Всё хорошо, что хорошо маршрутизируется

Чтобы использовать React Router в проекте, нужно импортировать необходимые компоненты. Если хотите добавить Route в приложение, убедитесь, что он обёрнут в компонент BrowserRouter. Компонент Route устанавливает связь между путём, который указан в пропсе path, и URL-адресом, который в данный момент используется браузером. При каждом обращении к этому URL будет отображаться компонент внутри маршрута.

Одного маршрута недостаточно, чтобы почувствовать преимущества библиотеки React Router. В следующем уроке научимся добавлять несколько маршрутов.

Пример App.js

import React from 'react';

import './App.css';

import Header from './Header';

import Diary from './Diary';

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

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

function App() {

return (

<BrowserRouter>

<Header />

<Route path="/">

<Diary />

</Route>

</BrowserRouter>

);

}

export default App;

# Работа с несколькими маршрутами

Сейчас приложение содержит три маршрута:

Скопировать кодJSX

// маршруты внутри файла App.js

<Route path="/">

<Dashboard />

</Route>

<Route path="/reviews">

<Reviews />

</Route>

<Route path="/about-me">

<AboutMe />

</Route>

Когда мы переходим по localhost:3000/, отображается компонент Dashboard — так и должно быть. Но есть проблема: если указать адрес localhost:3000/review, помимо компонента Reviews, на странице отобразится Dashboard, который относится к маршруту /. То же самое случится, если перейти по ссылке localhost:3000/about-me.

Это происходит потому, что значение пропса path сравнивается с путём в URL не по принципу равенства, а по принципу «начинается с». Иными словами, маршрут с путём / будет активным для любого пути в URL, так как любой путь всегда начинается с /. Даже если перейти по localhost:3000/french-fries, всё равно отобразится Dashboard, ведь в названии маршрута тоже стоит /.

Простое решение — добавить пропс exact к первому маршруту:

Скопировать кодJSX

<Route exact path="/">

<Dashboard />

</Route>

Теперь компонент Dashboard будет отображаться только при переходе по маршруту /. Пропс exact гарантирует, что значение пропса path будет сравниваться с путём в URL по принципу полного равенства.

# Компонент Switch

Попробуем решить задачу другим способом — применим компонент Switch. Если разместить несколько компонентов Route внутри Switch, отрисуется только один из них.

Импортируем этот компонент и обернём в него маршруты:

Скопировать кодJSX

*// App.js*

import React from 'react';

import { Route, Switch } from 'react-router-dom'; *// импортируем Switch*

import Dashboard from './Dashboard';

import Header from './Header';

import Reviews from './Reviews';

import AboutMe from './AboutMe';

import './App.css';

function App() {

return (

<div className="App">

<Header />

<Switch>

<Route path="/">

<Dashboard />

</Route>

<Route path="/reviews">

<Reviews />

</Route>

<Route path="/about-me">

<AboutMe />

</Route>

</Switch>

</div>

);

}

export default App;

Всё на своих местах.

Перейдя по localhost:3000/, увидим только Dashboard. Отлично. А вот если мы перейдём по localhost:3000/reviews или localhost:3000/about-me, компонент Dashboard снова отрисуется на странице. Почему?

Компонент Switch проверяет маршруты сверху вниз и рендерит самый первый из подходящих. Поскольку оба пути /reviews и /about-me всё так же начинаются с /, компонент Switch возвращает Dashboard.

Компонент Switch не решает нашу проблему, но решает свою задачу — за раз отображает один компонент.

## Пропс exact

Если / — первый маршрут внутри Switch, нужно просто добавить ему пропс exact:

Скопировать кодJSX

// изменим компонент Route в файле App.js

<Route exact path="/">

<Dashboard />

</Route>

Теперь все ссылки отображаются корректно.

# Долгий, странный путь

В этом уроке вы познакомились с принципом работы компонента Switch. Теперь разберём случай, когда его нужно использовать.

Например, с его помощью можно определить приоритет маршрутов. Даже если несколько маршрутов «подходит» под текущий URL, отрисуется только первый из них:

Скопировать кодJSX

*// JSX приложения онлайн-магазина*

<NavBar />

<MainContent />

<SideBar>

<Switch>

<Route path="/sales">

<RecommendedProducts />

</Route>

<Route path="/">

<ShoppingCart />

</Route>

</Switch>

</SideBar>

<Footer />

Компонент Switch применяют в тех случаях, когда нужно отрисовать только один маршрут.

В следующем уроке вы узнаете, как в React Router работать со ссылками и навигацией внутри приложения.

Пример

import React from 'react';

import './App.css';

import Header from './Header';

import Diary from './Diary';

import Tips from './Tips';

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

function App() {

return (

<BrowserRouter>

<Header />

<Switch>

<Route exact path="/">

<Diary />

</Route>

<Route path="/tips">

<Tips />

</Route>

</Switch>

</BrowserRouter>

);

}

export default App;

# Навигация и ссылки

Надо признать, что принуждать пользователей вводить вручную URL-адрес, чтобы перемещаться по приложению, — не самое мудрое решение. Вы уже знаете о теге <a> и о том, как его можно использовать, чтобы помочь пользователям находить нужную информацию.

React Router предлагает свой набор инструментов навигации, и в этом уроке мы их рассмотрим.

# Недостающая ссылка

React Router предоставляет компонент Link, который выполняет операцию, аналогичную знакомым тегам <a> из стандартного HTML. В действительности Link –– это тег <a> с дополнительными характеристиками для работы с React Router.

Импортируем компонент Link из react-router-dom. Затем добавим ссылку на компонент Reviews, чтобы пользователи могли легко просматривать все последние обзоры эмодзи:

Скопировать кодJSX

*// Dashboard.js*

import React from 'react';

import './Dashboard.css';

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

function Dashboard () {

return (

<div className="dashboard">

<h2>Emoji Critic — всё об эмодзи</h2>

<p>

#1 среди авторов обзоров на эмодзи в 2020 году!

</p>

<Link to="/reviews">

Нажмите, чтобы увидеть самые свежие ревью!

</Link>

</div>

)

}

export default Dashboard;

Мы добавили Link внутри JSX под тегом <p>. У компонента Link есть пропс to для описания URL, по которому следует перейти.

## Компонент NavLink

Ещё один удобный компонент для работы со ссылками в React Router — NavLink. На самом деле, этот компонент практически идентичен стандартному компоненту Link, но у него есть дополнительные функции, которые полезны при создании списков и меню. Чтобы понять, как именно работает NavLink, создадим ссылку на Dashboard и рассмотрим синтаксис компонента:

Скопировать кодJSX

<NavLink to="/" activeClassName="nav\_\_link\_active">Dashboard</NavLink>

Этот компонент отличается от Link тем, что у него есть пропс activeClassName. Он принимает в качестве значения селектор CSS-класса. Указанный в activeClassName CSS-класс будет добавлен к тому элементу NavLink, маршрут которого совпадает с текущим URL в браузере. В приведённом выше примере, если пользователь перейдёт к /, элемент ссылки получит класс nav\_\_link\_active. Так вы сможете применять собственные стили к ссылкам для текущей активной страницы. Это полезно в панели навигации, поскольку помогает пользователю проследить своё текущее местоположение в приложении.

# NavLink в деле

Создадим небольшую панель навигации, чтобы лучше понять принцип работы NavLink. Сделаем это внутри NavBar.js:

Скопировать кодJSX

*// NavBar.js*

import React from 'react';

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

import './NavBar.css';

function NavBar () {

return (

<nav className="menu">

<NavLink exact to="/" className="menu\_\_link">Домой</NavLink>

<NavLink to="/reviews" className="menu\_\_link">Обзоры эмодзи</NavLink>

<NavLink to="/about-me" className="menu\_\_link">Обо мне</NavLink>

</nav>

)

}

export default NavBar;

Мы разместили три компонента NavLink в элементе <nav>. Теперь у пользователя появились ссылки в панели навигации, по которым можно быстро переходить от страницы к странице.

Воспользуемся пропсом activeClassName. Мы уже добавили пропс exact в компонент NavLink с маршрутом /. Если не добавить exact, к этой ссылке всегда будет применяться activeClassName:

Скопировать кодJSX

*// NavBar.js*

import React from 'react';

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

import './NavBar.css';

function NavBar () {

return (

<nav className="menu">

<NavLink exact to="/" activeClassName="menu\_\_link\_active" className="menu\_\_link">Домой</NavLink>

<NavLink to="/reviews" activeClassName="menu\_\_link\_active" className="menu\_\_link">Обзоры эмодзи</NavLink>

<NavLink to="/about-me" activeClassName="menu\_\_link\_active" className="menu\_\_link">Обо мне</NavLink>

</nav>

)

}

export default NavBar;

Осталось описать CSS-правило для класса, указанного в activeClassName. Сделаем активные ссылки внутри App.css зелёными.

Скопировать кодCSS

*/\* Добавим новый селектор в NavBar.css \*/*

.menu\_\_link\_active {

color: #4AEA37;

}

Теперь в приложении ссылка на текущую страницу будет зелёной.

# Не отклоняйтесь от маршрута

Вы уже научились реализовывать базовую маршрутизацию и навигацию в одностраничных приложениях. Чтобы настроить маршруты, нужно использовать BrowserRouter и Route. Можно сделать так, чтобы только один соответствующий маршрут отрисовывался с помощью компонента Switch, а с помощью exact указать, что путь должен быть точным. Также в этом уроке вы узнали, как с помощью компонентов Link и NavLink сделать удобную навигацию.

Пора усложнять задачу. В следующих уроках вы реализуете динамические вложенные маршруты в приложении.

Пример

import React from 'react';

import './NavBar.css';

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

function NavBar() {

return (

<nav className="menu">

<NavLink exact to="/" activeClassName="menu\_\_item\_active" className="menu\_\_item">Домой</NavLink>

<NavLink to="/tips" activeClassName="menu\_\_item\_active" className="menu\_\_item">Советы</NavLink>

</nav>

);

}

export default NavBar;

# Вложенные маршруты

Приложение Emoji Critic содержит четыре маршрута в файле App.js:

Скопировать кодJSX

// маршруты внутри файла App.js

<Switch>

<Route path="/">

<Dashboard />

</Route>

<Route path="/reviews">

<Reviews />

</Route>

<Route path="/about-me">

<AboutMe />

</Route>

<Route path="/about-us">

<AboutUs />

</Route>

</Switch>;

Усложним компонент AboutMe. Добавим три новых компонента: MyStory с биографией автора, Hobbies с информацией об увлечениях и Contact с контактными данными. Вы уже знаете, что в любом проекте нужно следить за структурой. Поэтому создадим папку about-me/ внутри директории components/. Переместим в неё AboutMe и три новых компонента.

Теперь нужно сделать так, чтобы пользователь мог переходить по вкладкам внутри страницы “About Me”. Для этого создадим новые маршруты. В идеале, если пользователь добавит localhost:3000/about-me/my-story в закладки, а потом откроет её из закладок, он увидит страницу с отрисованным компонентом MyStory. То же самое и с остальными добавленными компонентами.

Мы можем добавить новые маршруты прямо внутри AboutMe.js:

Скопировать кодJSX

import React from "react";

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

import MyStory from "./MyStory";

import Hobbies from "./Hobbies";

import Contact from "./Contact";

import "./AboutMe.css";

function AboutMe() {

return (

<div>

<ul className="links">

<li>

<Link to="/my-story">Моя история</Link>

</li>

<li>

<Link to="/hobbies">Хобби</Link>

</li>

<li>

<Link to="/contact">Контактная информация</Link>

</li>

</ul>

<Route path="/my-story">

<MyStory />

</Route>

<Route path="/hobbies">

<Hobbies />

</Route>

<Route path="/contact">

<Contact />

</Route>

</div>

);

}

export default AboutMe;

# Почти, но не совсем

Дело сделано — код написан. На странице «Обо мне» появились ссылки. Правда, они пока декоративные и не работают. Это интерактивный виджет, вы можете с ним взаимодействовать. Попробуйте нажать на любую вкладку или выделить текст на странице.

Например, если перейти по первой ссылке, браузер откроет страницу с адресом localhost:3000/my-story. А по задумке браузер должен перенаправить нас на localhost:3000/about-me/my-story.

Чтобы всё работало правильно, нужно применить хук в компонентах Link и Route.

## Хук useRouteMatch

Хук useRouteMatch позволяет узнать текущее состояние маршрута. Импортируем его из react-router-dom — так же, как другие компоненты, предоставляемые библиотекой. На следующих строках импортируем новые компоненты: MyStory, Hobbies и Contact:

Скопировать кодJSX

import React from "react";

import { Route, Link, useRouteMatch } from "react-router-dom";

import MyStory from "./MyStory";

import Hobbies from "./Hobbies";

import Contact from "./Contact";

import "./AboutMe.css";

function AboutMe() {

const { path, url } = useRouteMatch();

return (

<div>

<ul className="links">

<li>

<Link to="/my-story">Моя история</Link>

</li>

<li>

<Link to="/hobbies">Хобби</Link>

</li>

<li>

<Link to="/contact">Контактная информация</Link>

</li>

</ul>

<Route path="/my-story">

<MyStory />

</Route>

<Route path="/hobbies">

<Hobbies />

</Route>

<Route path="/contact">

<Contact />

</Route>

</div>

);

}

export default AboutMe;

Благодаря useRouteMatch мы можем получить две переменные: url и path. Переменной url мы можем создать относительную ссылку внутри компонента Link. А переменной path — относительный путь для компонентов Route.

Внутри компонента AboutMe, наряду с хуком useRouteMatch, используется синтаксис деструктуризации. Хук useRouteMatch позволяет извлекать значения для url и path, которые позже присваиваются константам с такими же именами:

Скопировать кодJSX

const { path, url } = useRouteMatch();

Изменим код компонентов Link и Route, чтобы они могли использовать относительные значения:

Скопировать кодJSX

// новый JSX внутри AboutMe.js

<ul className="links">

<li>

<Link to={`${url}/my-story`}>Моя история</Link>

</li>

<li>

<Link to={`${url}/hobbies`}>Хобби</Link>

</li>

<li>

<Link to={`${url}/contact`}>Контактная информация</Link>

</li>

</ul>

<Route path={`${path}/my-story`}>

<MyStory />

</Route>

<Route path={`${path}/hobbies`}>

<Hobbies />

</Route>

<Route path={`${path}/contact`}>

<Contact />

</Route>

Теперь навигация работает корректно. С помощью хука useRouteMatch можно легко создавать вложенные маршруты. Например, внутри App.js можно переименовать путь /about-me в /information-zone. Для текущего маршрута переменные url и path динамически извлекаются с помощью useRouteMatch, поэтому все маршруты внутри компонента работают правильно.

Вложенные маршруты помогают делать навигацию логичной и структурированной. Но важно настраивать маршруты так, чтобы можно было получать новые данные от API и баз данных. В следующем уроке вы научитесь это делать.

Пример

import React from 'react';

import './AboutUs.css';

import SiteHistory from './SiteHistory';

import SiteMission from './SiteMission';

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

function AboutUs () {

const { path, url } = useRouteMatch();

return (

<div className="about-us">

<ul>

<li>

<Link to={`${url}/site-history`}>Site History</Link>

</li>

<li>

<Link to={`${url}/site-mission`}>Site Mission</Link>

</li>

</ul>

<Route path={`${path}/site-history`}>

<SiteHistory />

</Route>

<Route path={`${path}/site-mission`}>

<SiteMission />

</Route>

</div>

)

}

export default AboutUs;

# Динамические маршруты

В предыдущих уроках вы познакомились с основами навигации и маршрутизации внутри приложений и попрактиковались в применении инструментов React Router. В этом уроке перейдём на новый уровень и разберём продвинутые возможности этой библиотеки.

Представьте, что вам нужно разработать приложение, в котором пользователи могут создавать профили и общаться друг с другом, — такую небольшую социальную сеть.

Вы создали социальную сеть для орнитологов, и у вас уже зарегистрировались несколько пользователей. Один из них выбрал логин parrot\_rescuer42 и, похоже, уже нашёл пару друзей.

Но новые друзья — явление непостоянное, так и список друзей пользователя parrot\_rescuer42 может меняться. Поэтому при создании маршрутов внутри приложения разработчик должен помнить о динамических данных: именах пользователей, фотографиях, идентификаторах друзей и постов.

# Данные приложения

У вас есть доступ к данным о списке друзей parrot\_rescuer42 в [формате JSON](https://praktikum.yandex.ru/trainer/web/lesson/ca5c3656-788c-4dc4-9f07-2223e36650d7/task/d5a9672e-b082-49af-8077-6391e09cb74a/):

Скопировать кодJSON

"friends":[

{

"name":"Саша",

"id":"0",

"profilePicDark":"/profile-images/bill-dark.png",

"profilePicLight":"/profile-images/bill-light.png",

"location":"Санкт-Петербург, Россия",

"favBirdQuote":"Красна птица пением, а человек – умением.",

"parrotsOwned":[

{

"name":"Кеша",

"favoriteToys":[

"бубенчик",

"бумажный бантик"

]

},

{

"name":"Чарли",

"favoriteToys":[

"маленькая чашечка",

"медвежонок"

]

}

]

},

{

"name":"Женя",

"id":"1",

"profilePicDark":"/profile-images/mary-dark.png",

"profilePicLight":"/profile-images/mary-light.png",

"location":"Владивосток, Россия",

"favBirdQuote":"Лучше синица в руках, чем журавль в небе.",

"parrotsOwned":[

{

"name":"Абрикос",

"favoriteToys":[

"карандаш",

"пищалка"

]

}

]

}

]

Обычно подобные данные получают с сервера, но сейчас мы сохраним эти данные как JavaScript-объект и импортируем его в файл App.js.

Затем внутри App.js настроим новые маршруты:

Скопировать кодJSX

<Route path='/friends/0'>

<Friend serverData={serverData} />

</Route>

<Route path='/friends/1'>

<Friend serverData={serverData} />

</Route>

У пользователя parrot\_rescuer42 только два друга, поэтому работать с данными несложно. Но если parrot\_rescuer42 решит обзавестись ещё друзьями, могут возникнуть сложности. Приложение совсем не масштабируемо, поскольку описанные в нём маршруты не умеют работать с динамическими данными. Поправим это с помощью React Router:

Скопировать кодJSX

*// файл App.js*

import React from "react";

import "./App.css";

import { BrowserRouter, NavLink, Route, Switch } from "react-router-dom";

*// импортируем нужные компоненты*

import Friends from "./Friends";

import Friend from "./Friend";

import Dashboard from "./Dashboard";

import serverData from "../serverData";

function App() {

return (

<BrowserRouter>

<div className="App">

<header className="header">

<NavLink to="/" className="header\_\_logo">

Parrot Friendship Society

</NavLink>

<nav className="menu">

<ul className="menu\_\_list">

<li className="menu\_\_list-item">

<NavLink className="menu\_\_link" to="/friends">

Друзья

</NavLink>

</li>

</ul>

</nav>

</header>

<Switch>

<Route exact path="/">

<Dashboard />

</Route>

<Route exact path="/friends">

<Friends serverData={serverData} />

</Route>

<Route path="/friends/:id">

<Friend serverData={serverData} />

</Route>

</Switch>

</div>

</BrowserRouter>

);

}

export default App;

Новый компонент Friends отображает набор ссылок на друзей, добавленных пользователем. А компонент Friend отвечает за отображение данных о конкретном друге. Мы передали serverData обоим компонентам как пропс.

Внутри пути компонента Route стоит двоеточие : перед id:

Скопировать кодJSX

<Route path='/friends/:id'>

<Friend serverData={serverData} />

</Route>

Разберём, зачем это нужно.

# Параметры URL

Значение пропса path компонента Route может содержать переменные элемента, которые начинаются с двоеточия :. Так мы можем получить доступ к любому значению, которое находится на этом же месте в URL. Затем к этому значению можно обратиться как к переменной внутри дочернего компонента Route.

Вы уже умеете передавать пропсы из родительского компонента в дочерний. Похожим образом можно передать значение из URL в компонент.

Скопировать кодJSX

<Route path='/friends/:id'>

<Friend serverData={serverData} />

</Route>

Предположим, пользователь перешёл на URL localhost:3000/friends/123.

Поскольку в маршруте стоит :, можно получить доступ к значению 123 внутри компонента Friend. Например, чтобы вывести его в JSX:

Скопировать кодJSX

<h2>{id}</h2>

Параметру URL можно дать любое имя, не только :id. Назовём его favoriteParrot:

Скопировать кодJSX

<Route path='/my-birds/:favoriteParrot'>

<Friend serverData={serverData} />

</Route>

Затем можно получить доступ к значению параметра внутри компонента:

Скопировать кодJSX

<p>{favoriteParrot}</p>

Чтобы это работало, нужно использовать «Реакт-хук» useParams:

Скопировать кодJSX

*// компонент Friend*

import React from 'react';

import { useParams } from 'react-router-dom'; *// импортируем хук*

function Friend() {

let { id } = useParams(); *// получаем значение id*

return (

<div>

<h2>{id}</h2>

</div>

);

}

export default Friend;

Мы импортировали хук useParams из react-router-dom и с помощью деструктуризации извлекли переменную id. Эта переменная будет соответствовать любому значению, которое находится на месте :id в URL.

Когда страница localhost:3000/friends/i3r7d1w2w открыта, переменная id равна значению i3r7d1w2w. Это значение можно получить внутри компонента Friend.

Так мы сможем использовать значение как ключ и искать дополнительную информацию о пользователе.

# Параметры URL на практике

В приложении есть экран со списком друзей. Чтобы попасть на него, нужно перейти по маршруту /friends. Внутри этого компонента мы собираем данные о друзьях parrot\_rescuer42. Применим метод map, чтобы генерировать компоненты Link для каждого друга. При этом будем использовать идентификатор друга в пропсе to:

Скопировать кодJSX

*// компонент Friends*

import React from 'react';

import { useRouteMatch, Link } from 'react-router-dom';

import './Friends.css';

function Friends(props) {

let { path } = useRouteMatch();

let { friends } = props.serverData; *// достаём данные*

return (

<div className="friends">

<ul className="friends\_\_list">

{

friends.map((friend) => {

return (

<li className="friend-preview" key={friend.id} >

<Link to={`${path}/${friend.id}`}>

<img className="friend-preview\_\_image" src={friend.profilePicDark} alt=""/>

<span className="friend-preview\_\_name">{friend.name}</span>

</Link>

</li>

)

})

}

</ul>

</div>

);

}

export default Friends;

Разберём эту логику на примере. Для друга с идентификатором 555 метод map создаст ссылку /friends/555. То же будет проделано для всех друзей, указанных в объекте friends.

Посмотрим ещё раз на маршруты:

Скопировать кодJSX

<Switch>

<Route exact path='/friends'>

<Friends serverData={serverData} />

</Route>

<Route path='/friends/:id'>

<Friend serverData={serverData} />

</Route>

</Switch>

Когда пользователь нажимает на ссылку с путём "/friends/:id", отрисовывается компонент Friend:

Скопировать кодJSX

*// компонент Friend*

import React from 'react';

import { useParams } from 'react-router-dom'; *// импортируем хук*

import './Friend.css';

function Friend(props) {

let { id } = useParams(); *// получаем доступ к параметру URL*

let { friends } = props.serverData; *// достаём данные, используя деструктуризацию*

const friend = friends.find(f => f.id === id);

return (

<div className="friend">

<div className="friend\_\_card">

<img className="friend\_\_userpic" src={friend.profilePicLight} alt={friend.name}/>

<div className="friend\_\_details">

<h3 className="friend\_\_name">{friend.name}</h3>

<p className="friend\_\_location">Местоположение: {friend.location}</p>

<p className="friend\_\_quantity">Количество домашних попугаев: {friend.parrotsOwned.length}</p>

<p className="friend\_\_fav-quote">Любимое высказывание о птицах: "{friend.favBirdQuote}"</p>

</div>

</div>

</div>

);

}

export default Friend;

Если у parrot\_rescuer42 есть друг по имени Пётр с идентификатором 55, на странице friends/ автоматически сгенерируется компонент Link со ссылкой на localhost:3000/friends/55. Если пользователь перейдёт по этой ссылке внутри приложения или введёт её вручную в браузере, он увидит страницу Петра с соответствующими динамическими данными: name, location, parrotsOwned.length и favBirdQuote.

В этом примере мы использовали данные из локального файла JavaScript, но процесс почти идентичен, когда данные приходят из API.

Вы справились с самым трудным уроком темы. Что-то может казаться ещё не до конца понятным, но с практикой всё встанет на свои места.

Примеры

import React from 'react';

import './App.css';

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

function Doubler() {

const { number } = useParams();

if (!isNaN(number)){

// если обнаружено число, будет запущен этот код

return (

<p>Обнаружен числовой URL-адрес. Результат удвоения: ${{number} \* 2}</p>

);

} else {

// если значение не числовое, будет запущен этот код

return(

<p>Укажите число, которое хотите удвоить</p>

);

}

}

export default Doubler;

Завершим работу над Emoji Critic. Мы уже извлекли и отформатировали данные из API обзоров с помощью fetch. Сейчас эти данные передаются как пропс в компонент Reviews, а ещё выводится список заголовков для каждого обзора.

В Reviews.js внутри функции map вложите новый компонент Link в <li>. Внутрь Link переместите {review.title}. Стили и значение key в списке менять не нужно.

Путь Link должен состоять из /reviews/ и идентификатора обзора. К идентификатору можно обратиться так: review.id.

Всё нужное уже импортировано.

import React from 'react';

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

import './Reviews.css';

function Reviews (props) {

return (

// необходимо изменить код внутри метода map

<>

<ul className="reviews">

{props.reviews && props.reviews.map((review)=>{

return <li key={review.id} className="reviews\_\_item">

<Link to={`/reviews/${review.id}`}>{review.title}</Link>

</li>

})}

</ul>

</>

)

}

export default Reviews;

Когда пользователь нажимает на обзор, должен отрисовываться компонент Review с соответствующим идентификатором, который совпадает с URL-параметром. В файле Review.js требуется заполнить JSX значениями title, text и rating обзоров. Заголовок уже добавлен. Ваша задача — добавить текст и рейтинг по аналогии.

import React from 'react';

import './Review.css';

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

function Review (props) {

let { reviews } = props;

let { id } = useParams();

// the object keys start with 0, but the IDs in the API begin at 1

id = id - 1;

return (

<div className="review">

{

reviews &&

<div className="review\_\_item">

<h3>{reviews[id].title}</h3>

<p>{reviews[id].text}</p>

<p className="review\_\_rating">Рейтинг: {reviews[id].rating}/5</p>

</div>

}

</div>

);

}

export default Review;

# Программная навигация

Когда пользователь перемещается по экранам в приложении, за кулисами создаётся история навигации. Когда пользователь посещает новые страницы, в историю помещаются новые элементы. Через историю навигации пользователь может переходить назад и вперёд по страницам. Именно эту функцию и выполняют кнопки «Назад» и «Вперёд» в веб-браузере. В этом уроке вы научитесь реализовывать такую функциональность и в приложении. Кроме того, сможете полностью заменить текущую запись в стеке другой записью и перенаправить пользователя на другую страницу.

## Объект history

Добавим кнопку «Назад», чтобы пользователи могли легко возвращаться на страницу со списком друзей. Для этого воспользуемся хуком useHistory. Он предоставляет несколько методов, которые помогают дополнительно контролировать навигацию пользователя.

Как и с предыдущими хуками React Router, импортируем его в файл Friend.js:

Скопировать кодJSX

*// импортируем ещё один хук во Friend.js*

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

Теперь вызовем хук внутри функционального компонента. Для этого присвоим его константе history:

Скопировать кодJSX

*// функциональный компонент Friend*

const history = useHistory();

Внутри JSX добавим кнопку <button>. Нажатие на неё вызовет метод history.goBack, который отправит пользователя на предыдущую страницу:

Скопировать кодJSX

// JSX-код компонента Friend

<div className="friend\_\_card">

<img className="friend\_\_userpic" src={friend.profilePicLight} alt={friend.name}/>

<div className="friend\_\_details">

<h3 className="friend\_\_name">{friend.name}</h3>

<p className="friend\_\_location">Местоположение: {friend.location}</p>

<p className="friend\_\_quantity">Количество домашних попугаев: {friend.parrotsOwned.length}</p>

<p className="friend\_\_fav-quote">Любимое высказывание о птицах: "{friend.favBirdQuote}"</p>

</div>

</div>

<button className="button button\_type\_back" onClick={() => history.goBack()}></button>

Мы использовали метод history.goBack, чтобы переместить пользователя по истории навигации на предыдущую страницу. Ещё у объекта history бывают такие методы:

* history.goForward — перемещает пользователя «вперёд» по навигации истории;
* history.push — создаёт запись в навигации истории.

Так же как метод history.goBack похож на нажатие кнопки «Назад» в браузере, а history.goForward соответствует нажатию кнопки «Вперёд», history.push аналогичен открытию новой страницы.

# Переадресация

Компонент Redirect заменяет текущую запись в навигации истории и перенаправляет пользователя на другую страницу. Его часто используют в веб-приложениях после авторизации: например, чтобы авторизованный пользователь не мог вернуться на страницу входа.

Посмотрим, как это работает на практике: используем тернарный оператор внутри маршрута, чтобы проверить значение loggedIn и определить, вошёл ли пользователь в систему. Если да, пользователь увидит компонент своего профиля UserProfile. В противном случае Redirect отправит пользователя на экран авторизации:

Скопировать кодJSX

<Route path="/my-profile">

{!loggedIn ? <Redirect to="/log-in" /> : <UserProfile />}

</Route>

Теперь вам под силу такое приложение, в котором пользователь сможет перемещаться вперёд и назад, сколько ему вздумается. А ещё вы сумеете настроить переадресацию в случаях, когда она нужна.

В следующем уроке вы научитесь делать страницу 404. Она пригодится, когда пользователь переходит по несуществующему адресу.

# Создание страницы 404

Если пользователь перейдёт по ссылке, которой нет в приложении, он всё также будет видеть панель навигации, но ничего больше не отрисуется. Хороший тон –– проинформировать пользователя, что он перешёл на неопределённый маршрут.

Для этого с помощью React Router сделаем маршрут «Страница не найдена».

Создадим новый компонент PageNotFound и поместим его в директорию components/:

Скопировать кодJSX

import React from 'react';

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

import './PageNotFound.css';

import Bye from '../images/404.svg';

function PageNotFound () {

return (

<div className="not-found">

<h3 className="not-found\_\_title">

<span>404</span> - Страница не найдена

</h3>

<img className="not-found\_\_image" src={Bye} alt=""/>

<p className="not-found\_\_text">

Ой, здесь ничего нет

</p>

<Link className="button button\_type\_to-main" to="/">Назад</Link>

</div>

)

}

export default PageNotFound;

Импортируем новый компонент в корневой компонент App приложения Emoji Critic.

Внутри App уже создано несколько компонентов Route в Switch:

Скопировать кодJSX

// маршруты внутри файла App.js

<Switch>

<Route exact path="/">

<Dashboard />

</Route>

<Route path="/reviews">

<Reviews />

</Route>

<Route path="/about-me">

<AboutMe />

</Route>

<Route path="/about-us">

<AboutUs />

</Route>

</Switch>

Важен порядок компонентов Route внутри Switch: только первый совпадающий маршрут будет отрисован. Когда Switch обнаружит, что URL браузера не совпадает с /, /reviews или /about-me, то отрисует компонент PageNotFound. Для этого в конце Switch добавим ещё один компонент Route со значением path="\*". Символ \* обозначает «любой».

Скопировать кодJSX

// маршруты внутри файла App.js

<Switch>

<Route exact path="/">

<Dashboard />

</Route>

<Route path="/reviews">

<Reviews />

</Route>

<Route path="/about-me">

<AboutMe />

</Route>

<Route path="/about-us">

<AboutUs />

</Route>

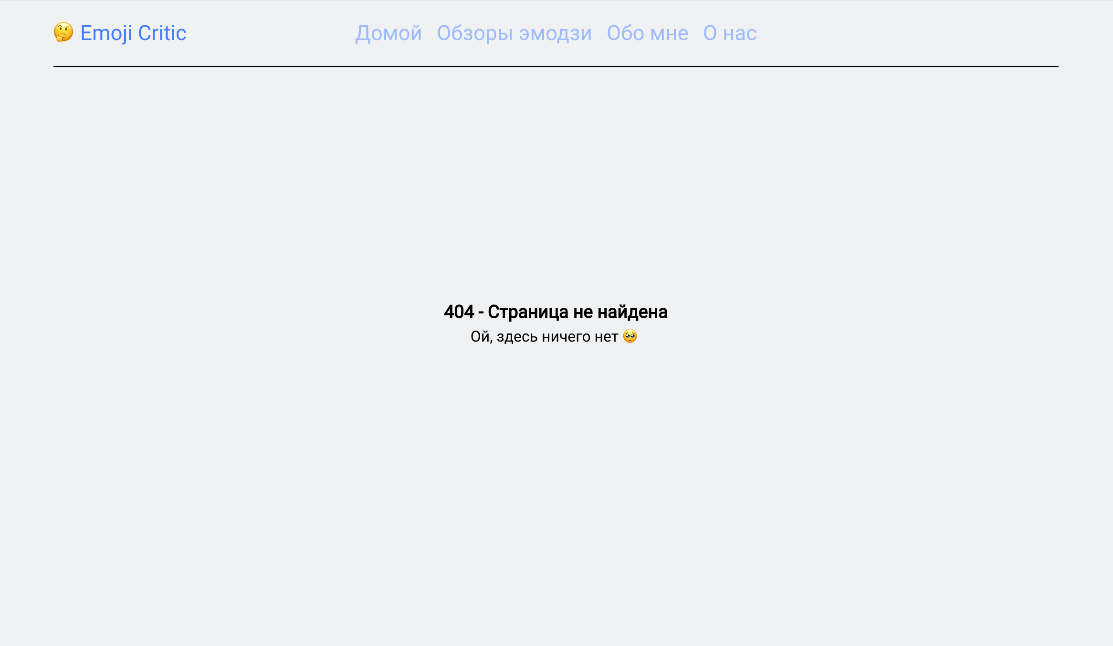
<Route path="\*">

<PageNotFound />

</Route>

</Switch>

Теперь, если запустить проект на локальном сервере и перейти на несуществующий маршрут, например, localhost:3000/zdesnichegonet, приложение отрисует страницу 404.



В этой теме вы узнали много нового о библиотеке React Router и её возможностях. Ещё немного практики и можно двигаться дальше!

# Заключение

Вы овладели искусством маршрутизации. В этой теме вы познакомились с React Router. Научились работать с несколькими маршрутами и настраивать навигацию по ссылке. Узнали, как создавать вложенные маршруты и настраивать маршруты для обработки динамически генерируемых данных.

Роутинг обеспечивает синхронизацию URL-пути с пользовательским интерфейсом. Грамотно построенная маршрутизация сохраняет чистоту кода и поддерживает интуитивно понятный интерфейс.

Теперь вы знаете, почему React Router — популярный инструмент среди разработчиков. С помощью него можно легко настроить маршрутизацию и использовать все преимущества компонентов и хуков.

Впереди ещё много интересного, так что продолжим изучение «Реакта» в следующей теме.