

РОУТИНГ В REACT

РОУТЕР

- ♦ В React.js нет собственных сервисов роутинга, однако существует несколько бесплатных библиотек с открытым кодом, который прекрасно выполняют эти задачи.
- ♦ Мы обсудим роутер, разработанный командой React.js:
<https://github.com/reactjs/react-router>
- ♦ React Router обеспечивает синхронизацию UI с URL. Он имеет простой интерфейс API с такими мощными встроенными функциями, как асинхронная загрузка кода, динамическое сопоставление маршрутов и обработка изменения расположения.



React Router

ЗАЧЕМ НУЖЕН РОУТЕР?

<http://localhost:3000/shop#/books>

[BOOKS](#) [GAMES](#) [ALBUMS](#) [APPS](#)

Books:

1. [ReactJS](#)
2. [JavaScript](#)
3. [Redux](#)

<http://localhost:3000/shop#/books/1>

РОУТЫ (МАРШРУТЫ)

Роутинг позволяет показывать различное содержимое в зависимости от выбранного роута. Маршрут (роут) указывается в URL.

`http://localhost:3000/shop#/books`

`http://localhost:3000/shop#/albums`

`http://localhost:3000/shop#/games`


`http://localhost:3000/shop#/apps`

books-> BookComponent

albums->AlbumsComponent

games->GamesComponent

apps->AppsComponent



Роут: загружает соответствующий компонент в index.html

Преимущества:

- улучшенная структура приложения;
- возможность перехода вперед/назад в браузере;
- состояние приложения может быть сообщено как URL (для поисковой системы, социальной сети, избранных сайтов, мессенджеров и т. д.)

УСТАНОВКА И ИСПОЛЬЗОВАНИЕ REACT ROUTER 4

npm install react-router-dom

```
const Main = () => (  
  <main>  
    <Switch>  
      <Route exact path="/" component={Home}/>  
      <Route path="/roster" component={Roster}/>  
      <Route path="/schedule"  
        component={Schedule}/>  
    </Switch>  
  </main>  
)
```

```
import { BrowserRouter }  
  from 'react-router-dom'  
ReactDOM.render((  
  <BrowserRouter>  
    <App />  
  </BrowserRouter>  
)  
,  
  document.getElementById('root'))  
  
const App = () => (  
  <div>  
    <Header />  
    <Main />  
  </div>  
)
```

EXACT PATH И НЕ EXACT PATH

`<Route exact path="/one" component={About} />`

path	location.pathname	exact	matches?
/one	/one/two	true	нет
/one	/one/two	false	да

exact: когда true, будет соответствовать (match), только если **path** в точности такой, как **location.pathname**.

ВИДЫ РОУТЕРОВ

`<Router>`, который использует API **HTML5 history** (`pushState`, `replaceState` и событие `popstate`), чтобы синхронизировать ваш интерфейс с URL-адресом.

`<BrowserRouter`

```
  basename={optionalString}  
  forceRefresh={optionalBool}  
  getUserConfirmation={optionalFunc}  
  keyLength={optionalNumber}  
>  
  <App/>  
</BrowserRouter>
```

basename – начальный URL для всех location. Если ваше приложение работает не из корня сервера, например <http://server.com/> а из поддиректории на вашем сервере, например <http://server.com/app> вы должны указать эту поддиректорию:

```
  basename="/app"
```

forceRefresh – перезагружать страницу с сервера (имеет смысл только для браузеров, не поддерживающих HTML5 history)

getUserConfirmation – можно задать функцию, запрашивающую подтверждение перехода на другой адрес:

```
(message, callback) => callback(true | false)
```

ВИДЫ РОУТЕРОВ

`<Router>`, который использует **хэш-часть** URL-адреса (то есть `window.location.hash`), чтобы синхронизировать ваш интерфейс с URL-адресом:

`<HashRouter`

```
  basename={optionalString}  
  forceRefresh={optionalBool}  
  getUserConfirmation={optionalFunc}  
  hashType={optionalString}
```

```
>
```

```
<App/>
```

```
</HashRouter>
```

hashtype: Тип кодирования для `window.location.hash`.

Возможные значения:

"slash" – создает хэш вида `#/` and `#/sunshine/lollipops`

"noslash" – создает хэш вида `#` и `#sunshine/lollipops`

ВЛОЖЕННЫЕ МАРШРУТЫ

Маршрут к профилю игрока `/roster/:number` не включается в предыдущий `<Switch>`. Вместо этого, он отображается компонентом `<Roster>`, который отображается, когда имя пути начинается с `/roster`:

```
const Roster = () => (
  <div>
    <h2>This is a roster page!</h2>
    <Switch>
      <Route exact path='/roster' component={FullRoster}/>
      <Route path='/roster/:number' component={Player}/>
    </Switch>
  </div>
)
```

`/roster` — Используется, только когда имя пути — точно `/roster`, и поэтому данному элементу маршрута нужно дать точное свойство.

`/roster/:number` — Этот маршрут использует параметр пути, чтобы получить части имени пути, которая идет после `/roster`.

ПАРАМЕТРЫ МАРШРУТИЗАТОРА

Часть `:number` пути `/roster/:number` означает, что часть имени пути, которая идет после `/roster/`, будет получена и сохранена как **`match.params.number`**. Например, имя пути `/roster/6` создаст объект **`params : { number: '6' }`**

```
const Player = (props) => {  
  const player = PlayerAPI.get(  
    parseInt(props.match.params.number, 10)  
  )  
  if (!player) {  
    return <div>Sorry, but the player was not found</div>  
  }  
  return (  
    <div>  
      <h1>{player.name} ({player.number})</h1>  
      <h2>{player.position}</h2>  
    </div>  
  )  
}
```

ССЫЛКИ

```
const Header = () => (
  <header>
    <nav>
      <ul>
        <li><Link to='/'>Home</Link></li>
        <li><Link to='/roster'>Roster</Link></li>
        <li><Link to='/schedule'>Schedule</Link></li>
      </ul>
    </nav>
  </header>
)
```

Параметр **to** может быть строкой или **положением объекта** (содержащим комбинацию имени пути, поискового запроса, хэша и свойств состояния). Если он является строкой, он преобразуется в положение объекта.

```
<Link to={{ pathname: '/roster/7' }}>Player #7</Link>
```

ССЫЛКИ LINK

<Link

to={{

pathname: **"/courses"**,

search: **"?sort=name"**,

hash: **"#the-hash"**,

state: { fromDashboard: **true** }

}}

/>

pathname: строка, описывающая куда ведет ссылка

search: строковое представление параметров query

hash: хэш для помещения в URL, например #a-hash.

state: состояние, которое надо передать в location

<Link to="/courses" replace />

Когда true, нажатие ссылки заменит текущую запись в истории вместо добавления новой записи.

ССЫЛКИ NAVLINK

```
<NavLink to="/faq" activeClassName="selected">  
  FAQs  
</NavLink>
```

специальная версия Link, которая добавляет класс
activeClassName к элементу ссылки, когда ссылка активна

```
<NavLink to="/faq"  
  activeStyle={{ fontWeight: "bold", color: "red" }} >  
  FAQs  
</NavLink>
```

здесь к ссылке применяются стили, определенные в activeStyle

ОГРАНИЧЕНИЕ ПЕРЕХОДА ПО ССЫЛКЕ

Используется для запроса подтверждения у пользователя перед тем, как перейти на другую страницу. Когда ваше приложение находится в состоянии, которое должно препятствовать навигации (например, форма заполнена наполовину), рендерите `<Prompt>`:

```
import { Prompt } from 'react-router'
<Prompt when={formIsHalfFilledOut} message="Are you sure you want to leave?" />
```

Также в `Prompt` можно передавать функцию:

```
<Prompt message={location =>
  location.pathname.startsWith("/app") ? true
  : `Are you sure you want to go to ${location.pathname}?` }
/>
```

РАЗДЕЛЕНИЕ КОДА

Одна из замечательных особенностей Интернета заключается в том, что нам не нужно заставлять посетителей сайта загружать все приложение, прежде чем его использовать.

Вы можете думать о разделении кода как о поэтапной загрузке приложения. Для этого мы будем использовать **webpack**, **@babel/plugin-syntax-dynamic-import** и **react-loadable**.

webpack имеет встроенную поддержку динамического импорта; однако, если вы используете Babel (например, для компиляции JSX на JavaScript), вам нужно будет использовать плагин **@babel/plugin-syntax-dynamic-import**.

.babelrc должен выглядеть примерно так:

```
{  
  "presets": ["@babel/react"],  
  "plugins": ["@babel/plugin-syntax-dynamic-import"]  
}
```

РАЗДЕЛЕНИЕ КОДА

react-loadable - это компонент высшего порядка для загрузки компонентов с динамическим импортом. Он делает разделение кода простым!

Вот пример использования **react-loadable**:

```
import Loadable from "react-loadable";  
import Loading from "./Loading";
```

```
const LoadableComponent = Loadable({  
  loader: () => import("./Dashboard"),  
  loading: Loading  
});
```

```
export default class LoadableDashboard extends Component {  
  render() {  
    return <LoadableComponent />;  
  }  
}
```

Loadable – это компонент высшего порядка: он принимает исходный компонент как параметр, и возвращает новый компонент, обогащенный дополнительной функциональностью.

Компонент загружается из отдельного файла `./Dashboard`, и будет загружен тогда, когда компонент `LoadableDashboard` нужно будет отрисовать

РАЗДЕЛЕНИЕ КОДА

Мы можем сконфигурировать процесс загрузки:

```
import Loadable from "react-loadable";
import Loading from "./Loading";
```

```
const LoadableComponent = Loadable({
  loader: () => import("./Dashboard"),
  loading: Loading ←
});
```

```
export default class LoadableDashboard
  extends Component {
  render() {
    return <LoadableComponent />;
  }
}
```

```
const Loading = ({isLoading,timedOut,pastDelay,error}) {
  if (isLoading) {
    if (timedOut) {
      return <div>Loader timed out!</div>;
    } else if (pastDelay) {
      return <div>Loading...</div>;
    } else {
      return null;
    }
  } else if (error) {
    return <div>Error! Component failed to load</div>;
  } else {
    return null;
  }
}
```

ПРАКТИКА

Блок 2.

Задание 2.

Роутинг в React.