

Django REST Framework

# Routing. SPA

---



# На этом уроке

1. Научимся создавать Single Page Application (SPA) с помощью React.
2. Настроим маршрутизацию (Routing) на стороне клиента.
3. Рассмотрим различные варианты маршрутизации.

## Оглавление

### [SPA](#)

[Введение](#)

[Настройка демонстрационного проекта](#)

### [Routing](#)

[Введение](#)

[HashRouter](#)

[Link](#)

[Switch](#)

[Redirect](#)

[Router с параметрами](#)

[BrowserRouter](#)

### [Итоги](#)

### [Глоссарий](#)

### [Дополнительные материалы](#)

### [Используемые источники](#)

### [Практическое задание](#)

[В этой самостоятельной работе тренируем умения](#)

[Зачем?](#)

[Последовательность действий](#)

# SPA

## Введение

SPA (Single Page Application) в отличие от MPA (Multi Page Application), состоят, как понятно из названия, всего из одной страницы. Она загружается с сервера один раз, и далее все действия, происходящие на ней, возможны благодаря выполнению javascript-кода на стороне клиента. SPA имеют следующие достоинства:

1. Высокая скорость.
2. Удобный пользовательский интерфейс.
3. Разделение frontend и back-end.
4. Возможность работать в офлайн-режиме.

Мы рассмотрели, как можно выводить один или несколько React-компонентов на странице. Но часто сразу все компоненты не нужны, особенно если сайт большой. Поэтому удобно добавить переходы по страницам, как с MPA. Но переходами должен заниматься javascript, иначе страницы перезагрузятся как на классических сайтах.

Библиотека [react-router-dom](#) позволяет удобно реализовать маршрутизацию (Routing) на стороне клиента. В этом занятии рассмотрим основные варианты её использования.

## Настройка демонстрационного проекта

Создадим демонстрационный проект для front-end-части.

```
npx create-react-app library
```

In terminal

После создания структуры проекта в папке src сформируем папку components. В ней создадим файл Author.js со следующим кодом:

```
import React from 'react'

const AuthorItem = ({item}) => {
  return (
    <tr>
      <td>{item.id}</td>
      <td>{item.name}</td>
      <td>{item.birthday_year}</td>
    </tr>
  )
}
```

```

const AuthorList = ({items}) => {
  return (
    <table>
      <tr>
        <th>ID</th>
        <th>NAME</th>
        <th>BIRTHDAY_YEAR</th>
      </tr>
      {items.map((item) => <AuthorItem item={item} />)}
    </table>
  )
}

export default AuthorList

```

/src/components/Author.js

Далее создадим файл Books.js с кодом компонентов для книг:

```

import React from 'react'

const BookItem = ({item}) => {
  return (
    <tr>
      <td>{item.id}</td>
      <td>{item.name}</td>
      <td>{item.author.name}</td>
    </tr>
  )
}

const BookList = ({items}) => {
  return (
    <table>
      <tr>
        <th>ID</th>
        <th>NAME</th>
        <th>AUHTOR</th>
      </tr>
      {items.map((item) => <BookItem item={item} />)}
    </table>
  )
}

export default BookList

```

/src/components/Book.js

В файле App.js подключим эти компоненты и напомним следующий код:

```
import React from 'react'
import AuthorList from './components/Author.js'
import BookList from './components/Book.js'

class App extends React.Component {

  constructor(props) {
    super(props)
    const author1 = {id: 1, name: 'Грин', birthday_year: 1880}
    const author2 = {id: 2, name: 'Пушкин', birthday_year: 1799}
    const authors = [author1, author2]
    const book1 = {id: 1, name: 'Алые паруса', author: author1}
    const book2 = {id: 2, name: 'Золотая цепь', author: author1}
    const book3 = {id: 3, name: 'Пиковая дама', author: author2}
    const book4 = {id: 4, name: 'Руслан и Людмила', author: author2}
    const books = [book1, book2, book3, book4]
    this.state = {
      'authors': authors,
      'books': books
    }
  }

  render() {
    return (
      <div className="App">
        <AuthorList items={this.state.authors} />
        <BookList items={this.state.books} />
      </div>
    )
  }
}

export default App;
```

/src/App.js

В конструкторе мы создали тестовые данные для книг и авторов, а в методе render подключили компоненты.

После запуска тестового сервера (npm run start) на одной странице появятся две таблицы. Одну создаст компонент AuthorList, другую — BookList.

Теперь всё готово для добавления Routing и разделения этих компонентов на разные страницы.

# Routing

## Введение

Для добавления маршрутизации используем библиотеку `react-router-dom`. Установим её с помощью `npm`.

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

In terminal

Применим следующие основные компоненты библиотеки:

1. `HashRouter`.
2. `Link`.
3. `Switch`.
4. `Redirect`.
5. `BrowserRouter`.

Рассмотрим назначение каждого из них на практическом примере.

## HashRouter

Такой вид роутера использует возможность браузеров обрабатывать `#` в адресе и переходить на соответствующую часть страницы. На практике он используется редко. В будущем заменим его на `BrowserRouter`.

Изменим код в `App.js` следующим образом:

```
import React from 'react'
import AuthorList from './components/Author.js'
import BookList from './components/Book.js'
import {HashRouter, Route} from 'react-router-dom'

class App extends React.Component {

  constructor(props) {
    super(props)
    const author1 = {id: 1, name: 'Грин', birthday_year: 1880}
    const author2 = {id: 2, name: 'Пушкин', birthday_year: 1799}
    const authors = [author1, author2]
    const book1 = {id: 1, name: 'Алые паруса', author: author1}
    const book2 = {id: 2, name: 'Золотая цепь', author: author1}
    const book3 = {id: 3, name: 'Пиковая дама', author: author2}
    const book4 = {id: 4, name: 'Руслан и Людмила', author: author2}
    const books = [book1, book2, book3, book4]
    this.state = {
```

```

    'authors': authors,
    'books': books
  }
}

render() {
  return (
    <div className="App">
      <HashRouter>
        <Route exact path="/" component={() => <AuthorList
items={this.state.authors} />} />
        <Route exact path="/books" component={() => <BookList
items={this.state.books} />} />
      </HashRouter>
    </div>
  )
}
}

export default App;

```

/src/App.js

Рассмотрим его по частям:

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

Сначала мы импортировали компоненты HashRouter и Router для их дальнейшего использования.

```

<div className="App">
  <HashRouter>
    <Route exact path="/" component={() => <AuthorList
items={this.state.authors} />} />
    <Route exact path="/books" component={() => <BookList
items={this.state.books} />} />
  </HashRouter>
</div>

```

Далее ту часть страницы, на которой компоненты будут меняться в зависимости от адреса, помещаем в компонент HashRouter. Router позволяет указать адрес с помощью path. Атрибут component служит для указания компонента, который отразится по этому адресу. Если требуется передать данные в component (как в нашем случае), передаётся не сам компонент, а функция замыкания. Она вернёт компонент с нужными данными.

Теперь при переходе по адресу `/` появится таблица авторов. При переходе на адрес `/#/books` увидим таблицу с книгами. Наш роутинг работает.

## Link

Теперь создадим меню для нашего приложения. В нём будут ссылки для перехода с одной страницы на другую. Для этого изменим код в `App.js` следующим образом:

```
...
import {HashRouter, Route, Link} from 'react-router-dom'

class App extends React.Component {

  ...

  render() {
    return (
      <div className="App">
        <HashRouter>
          <nav>
            <ul>
              <li>
                <Link to="/">Authors</Link>
              </li>
              <li>
                <Link to="/books">Books</Link>
              </li>
            </ul>
          </nav>

          <Route exact path="/" component={() => <AuthorList
items={this.state.authors} />} />
          <Route exact path="/books" component={() => <BookList
items={this.state.books} />} />
        </HashRouter>
      </div>

    )
  }
}

export default App;
```

`/src/App.js`

Выделим наиболее важные моменты.

```
<HashRouter>
  <nav>
    <ul>
      <li>
```



```
        <Link to="/">Authors</Link>
      </li>
      <li>
        <Link to="/books">Books</Link>
      </li>
    </ul>
  </nav>
  ...

```

Для создания ссылок используется специальный компонент `Link` (вместо стандартного тега `<a>`). В `Link` помещается атрибут `to` — это адрес, на который осуществляется переход (по аналогии с `href` у тега `<a>`). Компоненты `Link` должны находиться внутри компонента `HashRouter`, а не за ним.

Весь остальной код остался без изменений.

Таким образом, с помощью компонента `Link` можно создать ссылки для переходов по ним.

Если заменить `Link` на тег `<a>`, тогда при переходе по ссылке обновится окно браузера и загрузится страница. При использовании `Link` все переходы происходят на стороне клиента, и страница с сервера не загружается. Это делает работу сайта быстрой.

## Switch

Если хотим добавить возможность перехода на несуществующую страницу в нашем приложении, понадобится компонент `Switch`. Он позволяет указать компонент, который надо отрисовать, если не совпал ни один из указанных выше адресов. Внесём изменения в файл `App.js`:

```
import React from 'react'
import AuthorList from './components/Author.js'
import BookList from './components/Book.js'
import {HashRouter, Route, Link, Switch} from 'react-router-dom'

const NotFound404 = ({ location }) => {
  return (
    <div>
      <h1>Страница по адресу '{location.pathname}' не найдена</h1>
    </div>
  )
}

class App extends React.Component {
  ...

  render() {
    return (

```

```

    <div className="App">
      <HashRouter>
        <nav>
          <ul>
            <li>
              <Link to="/">Authors</Link>
            </li>
            <li>
              <Link to="/books">Books</Link>
            </li>
          </ul>
        </nav>
        <Switch>
          <Route exact path="/" component={() => <AuthorList
items={this.state.authors} />} />
          <Route exact path="/books" component={() => <BookList
items={this.state.books} />} />
          <Route component={NotFound404} />
        </Switch>
      </HashRouter>
    </div>
  )
}
}

export default App;

```

/src/App.js

Рассмотрим код по частям:

```

const NotFound404 = ({ location }) => {
  return (
    <div>
      <h1>Страница по адресу '{location.pathname}' не найдена</h1>
    </div>
  )
}

```

Сначала создадим простой компонент для отображения несуществующей страницы.

```

<Switch>
  <Route exact path="/" component={() => <AuthorList items={this.state.authors} />}
/>
  <Route exact path="/books" component={() => <BookList items={this.state.books} />}
/>
  <Route component={NotFound404} />
</Switch>

```

Объявление маршрутов мы дополнительно поместили в компонент Switch. В самом конце добавили компонент NotFound404. Таким образом, если ни один адрес не совпадает с указанным в браузере, будет выбран компонент NotFound404. Он покажет, что такой маршрут отсутствует.

## Redirect

Иногда требуется добавить перенаправление с одного адреса на другой. Это позволяет сделать компонент Redirect. Добавим в App.js перенаправление с адреса /authors/ на главную страницу.

```
...
<Switch>
<Route exact path="/" component={() => <AuthorList items={this.state.authors} />} />
<Route exact path="/books" component={() => <BookList items={this.state.books} />} />
<Redirect from="/authors" to="/" />
<Route component={NotFound404} />
</Switch>
...
```

/src/App.js

Компонент редирект принимает from — откуда и to — куда надо совершить перенаправление. Теперь при переходе по адресу /#/authors Router переправит нас на адрес /#/.

## Router с параметрами

Мы рассмотрели, как можно создать переходы по статическим адресам. Немного более сложная задача — переход по динамическому адресу с параметрами. Рассмотрим пример, когда при нажатии на автора нужно вывести только его книги.

Для начала внесём небольшие изменения в компонент AuthorItem в файле Author.js:

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

const AuthorItem = ({item}) => {
  return (
    <tr>
      <td><Link to={`author/${item.id}`}>{item.id}</Link></td>
      <td>{item.name}</td>
      <td>{item.birthday_year}</td>
    </tr>
  )
}
...
```

/src/components/Author.js

```
<td><Link to={`author/${item.id}`}>{item.id}</Link></td>
```

В этой строчке добавили ссылку Link. По ней осуществим переход на динамический адрес, зависящий от id автора. По этому адресу выведем книги только этого автора.

Далее в папке components создадим новый файл AuthorBook.js со следующим кодом:

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

const BookItem = ({item}) => {
  return (
    <tr>
      <td>{item.id}</td>
      <td>{item.name}</td>
      <td>{item.author.name}</td>
    </tr>
  )
}

const BookList = ({items}) => {
  let { id } = useParams();
  let filtered_items = items.filter((item) => item.author.id == id)
  return (
    <table>
      <tr>
        <th>ID</th>
        <th>NAME</th>
        <th>AUTHOR</th>
      </tr>
      {filtered_items.map((item) => <BookItem item={item} />)}
    </table>
  )
}

export default BookList
```

```
/src/components/AuthorBook.js
```

Мы создали компоненты для отображения книг одного автора. Так выглядит новый код:

```
let { id } = useParams();
let filtered_items = items.filter((item) => item.author.id == id)
```

useParams позволит получить параметры, переданные в адрес роутера. В этом случае нужен только один параметр id. После того как мы его получили, отфильтруем книги по идентификатору автора.

Остаётся добавить в App.js новый маршрут с параметром:

```
import React from 'react'
import AuthorList from './components/Author.js'
import BookList from './components/Book.js'
import AuthorBookList from './components/AuthorBook.js'
import {HashRouter, Route, Link, Switch, Redirect} from 'react-router-dom'

const NotFound404 = ({ location }) => {
  return (
    <div>
      <h1>Страница по адресу '{location.pathname}' не найдена</h1>
    </div>
  )
}

class App extends React.Component {

  constructor(props) {
    super(props)
    const author1 = {id: 1, name: 'Грин', birthday_year: 1880}
    const author2 = {id: 2, name: 'Пушкин', birthday_year: 1799}
    const authors = [author1, author2]
    const book1 = {id: 1, name: 'Алые паруса', author: author1}
    const book2 = {id: 2, name: 'Золотая цепь', author: author1}
    const book3 = {id: 3, name: 'Пиковая дама', author: author2}
    const book4 = {id: 4, name: 'Руслан и Людмила', author: author2}
    const books = [book1, book2, book3, book4]
    this.state = {
      'authors': authors,
      'books': books
    }
  }

  render() {
    return (
      <div className="App">
        <HashRouter>
          <nav>
            <ul>
              <li>
                <Link to="/">Authors</Link>
              </li>
              <li>
                <Link to="/books">Books</Link>
              </li>
            </ul>
          </nav>
        </HashRouter>
      </div>
    )
  }
}
```

```

        </nav>
        <Switch>
            <Route exact path="/" component={() => <AuthorList
items={this.state.authors} />} /> />
            <Route exact path="/books" component={() => <BookList
items={this.state.books} />} /> />
            <Route path="/author/:id">
                <AuthorBookList items={this.state.books} />
            </Route>
            <Redirect from="/authors" to="/" />
            <Route component={NotFound404} />
        </Switch>
    </HashRouter>
</div>
)
}
}

export default App;

```

/src/App.js

```

<Route path="/author/:id">
    <AuthorBookList items={this.state.books} />
</Route>

```

В параметре path у Route можно указывать параметры. Они идут после двоеточия. Далее внутри указываем соответствующий компонент. Это ещё один способ связи маршрута с компонентом. Можно использовать атрибут component в Route, как мы делали раньше.

Теперь при нажатии на id автора перейдём на страницу только с его книгами.

## BrowserRouter

HashRouter работает не во всех случаях и часто не рекомендуется к использованию. Он удобен в тех случаях, когда маршрутизация на сервере может пересекаться с маршрутизацией на клиенте. В таких случаях HashRouter гарантирует, что адреса не совпадут.

Для использования всех возможностей браузера заменим HashRouter на BrowserRouter. Это актуальное решение для большинства задач. BrowserRouter позволяет осуществлять переходы, использовать клавишу «назад» в браузере и сохранять историю. То есть полноценно работать с маршрутизацией на стороне клиента.

Достаточно просто заменить HashRouter на BrowserRouter при импорте или использовании. Всё остальное останется без изменения:

```

...
import {BrowserRouter, Route, Link, Switch, Redirect} from 'react-router-dom'

...

class App extends React.Component {

  ...

  render() {
    return (
      <div className="App">
        <BrowserRouter>
          ...
        </BrowserRouter>
      </div>
    )
  }
}

export default App;

```

/src/App.js

Теперь при переходах по ссылкам будем получать адреса без знака #, например, /books/ вместо /#/books/). Приложение будет работать так же быстро, без перезагрузки страниц.

## Итоги

В этом занятии мы рассмотрели большинство задач, связанных с маршрутизацией, и их решение с помощью библиотеки react-router-dom. Теперь мы знаем, как создавать современные SPA-приложения.

## Глоссарий

1. **BrowserRouter** — маршрутизатор на клиенте, использующий все возможности браузера.
2. **HashRouter** — маршрутизатор, использующий знак # в адресе.
3. **MPA** — Multi Page Application. Многостраничное приложение.
4. **Router** — маршрутизатор на стороне клиента.
5. **SPA** — Single Page Application. Одностраничное приложение.

## Дополнительные материалы

1. [Официальный сайт react-router-dom](#).

2. [Статья про виды веб-приложений](#).
3. [Официальный сайт React](#).

## Используемые источники

1. [Официальный сайт react-router-dom](#).
2. [Статья про виды веб-приложений](#).
3. [Официальный сайт React](#).
4. Алекс Бэнкс, Ева Порселло «React и Redux функциональная веб-разработка».

## Практическое задание

Добавьте маршрутизацию и новые страницы к приложению.

### В этой самостоятельной работе тренируем умения

1. Использовать компоненты react-router-dom.
2. Создавать переходы.
3. Создавать новые страницы и компоненты.

### Зачем?

Чтобы использовать Routing в SPA-приложениях.

### Последовательность действий

1. Сделать переходы между тремя страницами: список пользователей, список проектов, список ToDo.
2. Добавить компоненты для новых страниц (список проектов и список ToDo) и загрузку данных с back-end.
3. При необходимости перенастроить сериализацию на стороне back-end.
4. По желанию можно добавить любые другие страницы.
5. (Задание со \*) Реализовать страницу с информацией для одного проекта. Переход на неё осуществляется по нажатию на проект из списка.