GeekBrains

Django REST Framework

Введение в React. Компонентный подход

[javascript >= ES6]

На этом уроке

1. Научимся использовать django и react совместно.
2. Научимся отправлять запросы на back-end.
3. Узнаем, что такое компонентный подход.
4. Научимся быстро создавать React приложение.
5. Узнаем концепцию one-way data flow.

Оглавление

Введение в React

Выбор библиотеки

На предыдущем занятии мы рассмотрели основы создания REST API и написали небольшой пример на back-end. Так как теперь сервер не возвращает готовые страницы (система становится гибкой), мы будем создавать отдельное приложение на стороне front-end.

Приложение будет заниматься получением данных с back-end и предоставлять пользователю интерфейс для работы с ними. Этот интерфейс может быть веб-приложением, мобильным, десктопным и даже консольным. Такая свобода выбора появилась благодаря разделению back-end и front-end.

Мы будем создавать веб-приложение с использованием javascript стандарта ES6 и выше и библиотеки React.

Сегодня наиболее популярные библиотеки и фреймворки — React, Angular и Vue.

Angular довольно тяжеловесный фреймворк, требующий серьёзной подготовки перед использованием.

React и Vue более лёгкие библиотеки, похожие по своей архитектуре. Из-за нововведений во Vue версии 3.0, неоднозначно принятых сообществом, мы выбрали React. Хотя принципы работы во Vue — компонентный подход, one way data flow — будут похожи.

Мы будем рассматривать использование React на практических примерах с введением некоторых новых понятий по мере необходимости.

У библиотеки есть свой официальный сайт.

Установка

React можно подключить к странице в браузере, например, с CDN или скачав код библиотеки. Такой вариант больше подходит для ознакомления с возможностями библиотеки или для создания небольших скриптов. В нашем проекте, полностью основанном на React, мы будем пользоваться способом установки с помощью node.js.

node.js, npm

node.js позволяет запускать код javascript не только в браузере, но и на сервере. Скачать и установить node можно с официального сайта.

Вместе с node у нас появляется возможность использовать пакетный менеджер npm для установки сторонних библиотек

create-react-app

После установки node всё готово для добавления front-end-части в наш проект. Для демонстрации возьмём library из предыдущего занятия.

Важно! *Структура проектов может быть разная. Она зависит от соглашений, принятых в команде. Поэтому важно следовать рекомендациям по созданию папок и модулей, представленных в этом методическом пособии.*

Переходим в папку с проектом, в которой находится manage.py. Установим модуль npx.

```
node install -g npx
```

Этот модуль позволяет запускать исполняемые файлы, не устанавливая через npm. Ключ -g означает установку в общую папку node.

После успешной установки с помощью create-react-app создадим каркас нашего приложения.

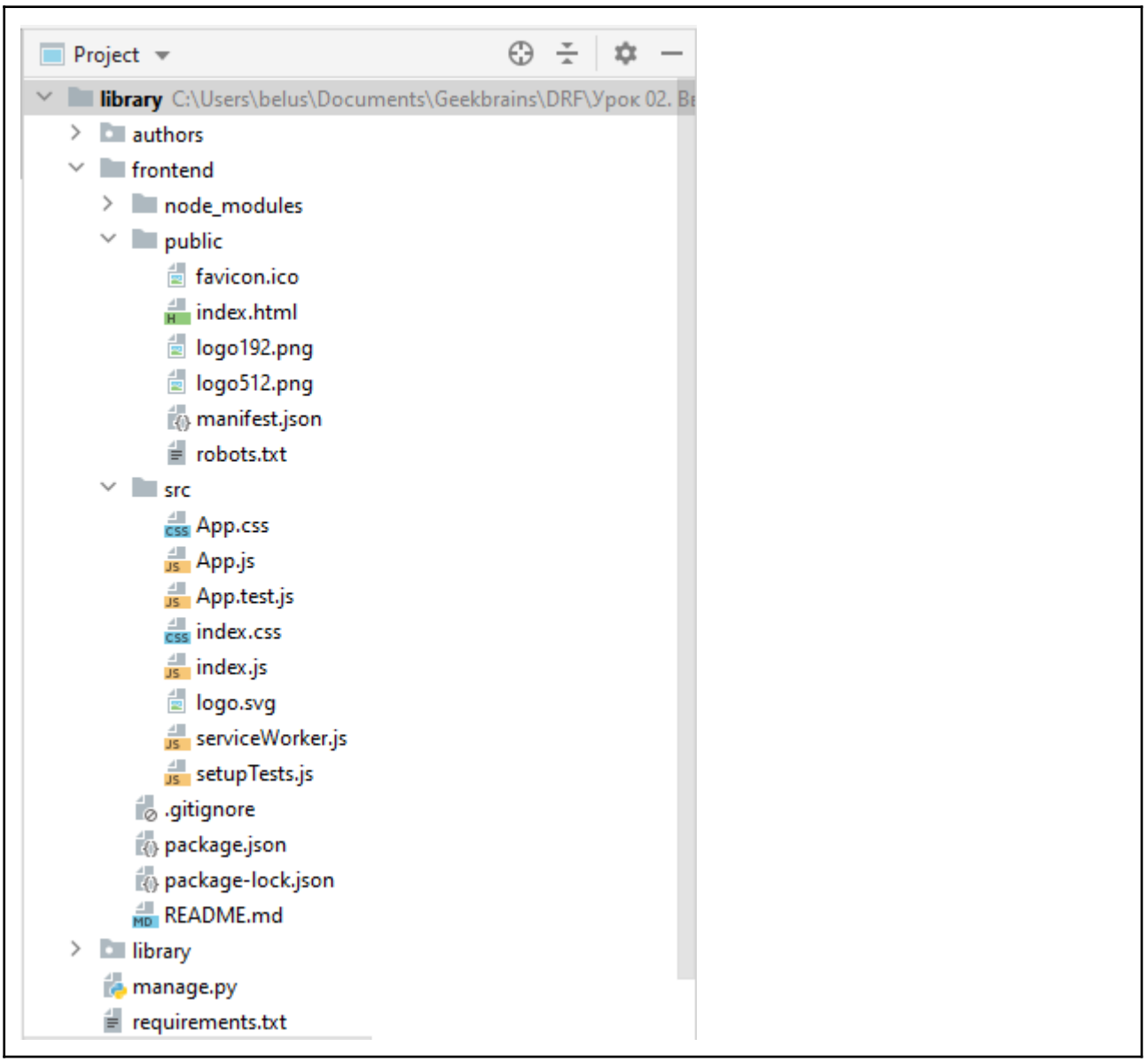
```
npx create-react-app frontend
```

В этом случае front-end — название нашего приложения.

Утилита create-react-app — очень удобный инструмент для создания и управления приложением на реакт. Она позволяет:

- создать удобную чёткую структуру приложения;
- управлять приложением в тестовом режиме;
- собирать приложение для production.

После выполнения команды у нас получится следующая структура проекта:



Папка front-end содержит в себе front-end-часть нашего приложения. В папке node_modules хранятся подключённые библиотеки. В public — различные файлы для запуска проекта. Сам код проекта находиться в src, там мы и будем работать.

После того как структура проекта создана, у нас есть возможность:

- запустить фронтенд в тестовом режиме;
- собрать проект для production.

Сборку в production мы рассмотрим в отдельном занятии, а разработку удобно вести в тестовом режиме.

Чтобы запустить тестовый сервер, переходим в папку front-end.

```
cd frontend
```

Важно! *Для работы с back-end-частью приложения выполняем команды в корне проекта. В ней находиться manage.py. В работе с frontend-частью приложения выполняем команды в папке front-end. Удобно сразу открыть 2 терминала, либо пользоваться двумя различными IDE.*

Для запуска тестового сервера выполняем:

```
npm run start
```

По умолчанию тестовый сервер запустится на http://localhost:3000, и откроется браузер.

One Way Data Flow. Теория и общие правила

Приложение на React состоит из компонентов. Каждый может включать в себя несколько других. Компоненты можно разделить на два основных типа:

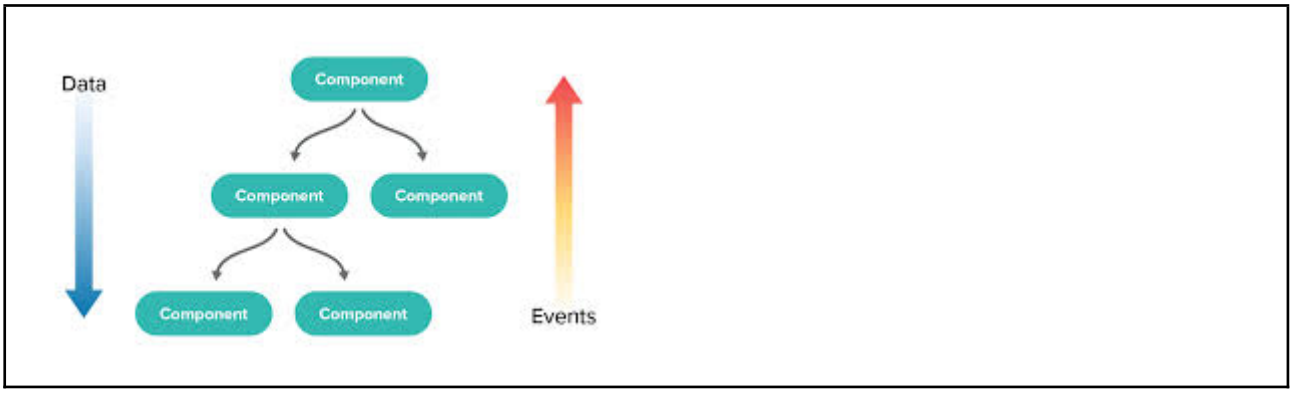
- у которых есть состояние;
- нет состояния. Они проще в использовании.

Компоненты без состояния мы будем создавать на основе функций, а те, у которых есть состояние — на основе классов.

При разработке приложений часто возникает вопрос, какие компоненты должны иметь состояние, а какие — нет. Есть несколько типовых решений этой задачи. Кратко рассмотрим каждый из подходов.

One Way Data Flow

Состояние есть только у компонента самого верхнего уровня, все остальные его не имеют. Таким образом, данные передаются в одном направлении от верхнего компонента к нижним. Поэтому этот способ имеет соответствующее название. Функции по изменению состояния тоже находятся в одном компоненте верхнего уровня. Они вместе с данными передаются сверху вниз к дочерним компонентам.



Этот подход хорошо работает для небольших и средних приложений. Так как он достаточно прост в освоении, будем пользоваться именно им.

Unidirectional Data Flow

В этом подходе несколько компонентов могут иметь состояние. Таким образом, система состоит из сложных компонентов, взаимодействующих между собой. Функции по изменению состояния находятся вместе с состоянием в нескольких компонентах. Вызывать их могут другие компоненты.

Поэтому довольно трудно отслеживать изменение состояния и поддерживать приложение в целом.

Flux. Redux

Паттерн Flux отделяет данные от компонентов и вводит понятия. Для знакомства с ним предлагаем самостоятельно изучить дополнительные материалы. Наиболее популярная реализация Flux для React — это Redux. Основная идея в том, что все данные и методы работы с ними находятся в специальном хранилище, а сами компоненты React не имеют состояния.

Этот подход хорошо работает в средних и крупных проектах и требует дополнительного изучения архитектуры Flux и библиотеки Redux.

Итоги

Так как большая часть внимания уделяется back-end на python, будем использовать One Way Data Flow из-за его надёжности и простоты и небольшой сложности проекта.

Компонентный подход

Главное приложение

Приступаем к использованию React.

В модуле App.js находится наш компонент верхнего уровня App. Только у него (компонента) будет состояние. Перепишем код модуля App.js. Он будет иметь следующий вид:

```
import React from 'react';
import logo from './logo.svg';
import './App.css';

class App extends React.Component {
   constructor(props) {
       super(props)
       this.state = {
           'authors': []
       }
```

```
    }

    render () {
        return (
            <div>
                Main App
            </div>
        )
    }
}

export default App;
```

/frontend/src/App.js

Разберём код по частям:

```
import React from 'react';
import logo from './logo.svg';
import './App.css';
```

Импортируем React из библиотеки react, svg-файл и файл со стилями. Таким образом, мы можем работать как с библиотеками js, так и с другими типами файлов.

```
class App extends React.Component
```

Создадим класс App, наследуем его от React.Component. Компонент App — это класс, имеющий состояние. Все остальные компоненты в нашей программе будут простыми функциями.

```
constructor(props) {
        super(props)
        this.state = {
            'authors': []
        }
    }
```

В конструктор класса передаётся объект props, super(props) — вызывает родительский конструктор. this.state — это объект состояния нашего компонента. Он будет хранить массив авторов, которые мы будем получать с back-end.

```
render () {
        return (
            <div>
                Main App
```

```
            </div>
        )
    }
```

Метод render отвечает за отрисовку нашего компонента. Пока он будет отрисовывать один тег div.

```
export default App;
```

Экспортируем наш компонент для использования в других модулях. Если открыть файл index.js, можно увидеть, что в нём используется компонент App.

Тестовый сервер можно не останавливать. Если в коде будут ошибки, мы увидим их в браузере и терминале. Сервер автоматически обновит страницу браузера, если ошибок нет.

Компоненты для одного автора и списка авторов

Теперь будем использовать компонентный подход. Мы создадим следующие компоненты:

- один автор;
- список авторов.

Компонент для отображения одного автора

В папке src создадим файл components. В нём будут находиться другие компоненты нашего приложения. В папке components создадим файл Author.js. Добавим в него следующий код.

```
import React from 'react'


const AuthorItem = ({author}) => {
    return (
        <tr>
            <td>
                {author.first_name}
            </td>
            <td>
                {author.last_name}
            </td>
            <td>
                {author.birthday_year}
            </td>
        </tr>
    )
}
```

/frontend/src/components/Author.js

Рассмотрим код компонента подробнее.

```
const AuthorItem = ({author}) =>
```

Это простой компонент без состояния. Такие компоненты удобнее создавать как функции. В нашем случае применяется стрелочная функция, но можно использовать и обычную function. В параметры каждого компонента на react приходит объект props. Он содержит в себе все переданные в компонент данные. Параметр {author} означает, что из всех props мы ожидаем получить author — объект автора, и далее работать только с ним.

```
return (
       <tr>
           <td>
               {author.first_name}
           </td>
           <td>
               {author.last_name}
           </td>
           <td>
               {author.birthday_year}
           </td>
       </tr>
   )
```

Функция возвращает разметку компонента. `{author.first_name}.` В неё мы помещаем данные из объекта author.

Компонент для списка авторов

В Author.js добавим ещё один компонент, и код примет следующий вид:

```
import React from 'react'


const AuthorItem = ({author}) => {
   return (
       <tr>
           <td>
               {author.first_name}
           </td>
           <td>
               {author.last_name}
           </td>
           <td>
```

```
                {author.birthday_year}
            </td>
        </tr>
    )
}


const AuthorList = ({authors}) => {
    return (
        <table>
            <th>
                First name
            </th>
            <th>
                Last Name
            </th>
            <th>
                Birthday year
            </th>
            {authors.map((author) => <AuthorItem author={author} />)}
        </table>
    )
}


export default AuthorList
```

/frontend/src/components/Author.js

Рассмотрим компонент AuthorList подробнее:

```
{authors.map((author) => <AuthorItem author={author} />)}
```

authors — это массив данных об авторах, который мы передадим в компонент. Используем функцию map, чтобы превратить каждого автора из массива в соответствующий компонент AuthorItem.

Обратите внимание, что компоненты в React могут быть вложены в друг друга и использоваться как новые теги с соответствующим именем.

```
export default AuthorList
```

Экспортируем компонент для дальнейшего использования в других модулях.

Изменения в компоненте App

Внесём следующие изменения в App.js:

- создадим заглушки для авторов;
- подключим и используем компонент AuthorList.

После этого код в App.js примет следующий вид:

```
import React from 'react';
import logo from './logo.svg';
import './App.css';
import AuthorList from './components/Author.js'


class App extends React.Component {

   constructor(props) {
       super(props)
       this.state = {
           'authors': []
       }
   }

   componentDidMount() {
       const authors = [
           {
               'first_name': 'Фёдор',
               'last_name': 'Достоевский',
               'birthday_year': 1821
           },
           {
               'first_name': 'Александр',
               'last_name': 'Грин',
               'birthday_year': 1880
           },
       ]
       this.setState(
           {
               'authors': authors
           }
       )
   }

   render () {
       return (
           <div>
               <AuthorList authors={this.state.authors} />
           </div>
       )
   }
}
```

```
export default App;
```

/frontend/src/App.js

Мы добавили метод componentDidMount.

```
componentDidMount() {
        const authors = [
            {
                'first_name': 'Фёдор',
                'last_name': 'Достоевский',
                'birthday_year': 1821
            },
            {
                'first_name': 'Александр',
                'last_name': 'Грин',
                'birthday_year': 1880
            },
        ]
        this.setState(
            {
                'authors': authors
            }
        )
    }
```

Он будет вызываться при монтировании компонента на страницу. В нём создали массив из объектов авторов для проверки работы приложения. Далее эти данные мы будем получать с back-end.

После этого с помощью метода this.setState меняем состояние компонента App и передаём данные об авторах.

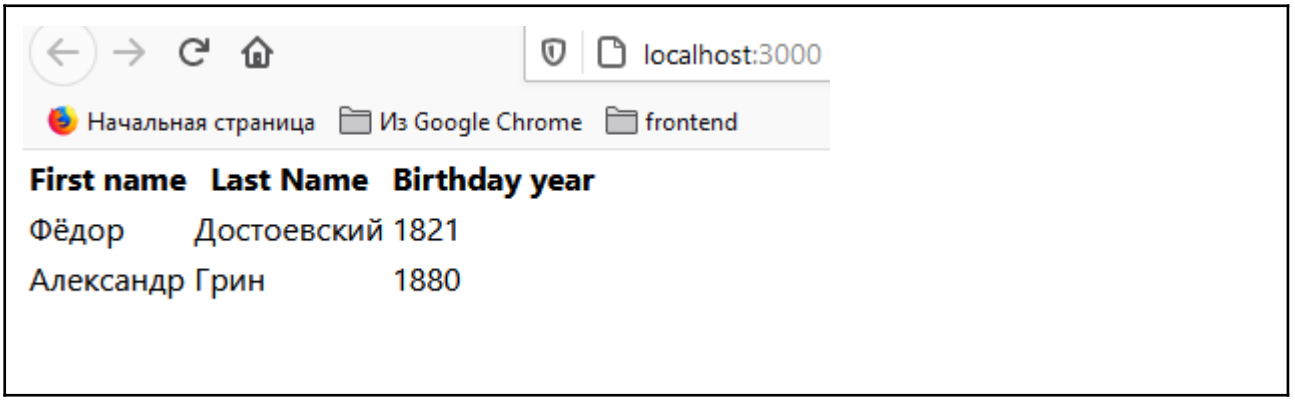
```
render () {
        return (
            <div>
                <AuthorList authors={this.state.authors} />
            </div>
        )
    }
```

Отрисовываем компонент App, который включает в себя компонент AuthorList, и передаём в AuthorList данные об авторах {this.state.authors}.

Не забываем предварительно импортировать компонент AuthorList.

```
import AuthorList from './components/Author.js'
```

Если в коде нет ошибок, в браузере покажется следующая страница (скриншот из браузера):



Резюме по frontend-части

1. Приложение на React состоит из компонентов, каждый из которых может включать в себя другие компоненты.
2. Данные в компонент можно передать через props.
3. Компоненты могут иметь и не иметь состояния.
4. По принципу One Way Data Flow храним данные в компоненте верхнего уровня и передаем в них сверху компоненты нижнего.

Взаимодействие front-end и back-end

После разработки API на back-end и приложения на front-end осталось сделать последние несколько действий:

- настроить политику CORS на back-end, чтобы получать запросы с тестового front-end-сервера;
- отправить запрос с front-end на back-end;
- получить и обработать ответ.

Настройка политики CORS

При попытке отправить запрос на сервер мы столкнёмся с ошибкой, так как сервер django по умолчанию не будет принимать запросы от тестового сервера. Он находится на другом порту. Предлагаем самостоятельно получить эту ошибку, пропустив этот пункт и отправив запрос на сервер.

Для настройки политики CORS удобно использовать стороннюю библиотеку django-cors-headers.

Устанавливаем библиотеку.

```
pip install django-cors-headers
```

Добавляем приложение в INSTALLED_APPS.

```
INSTALLED_APPS = [
    ...
    'corsheaders',
    ...
]
```

/library/settings.py

Подключаем 'corsheaders.middleware.CorsMiddleware в MIDDLEWARE.

```
MIDDLEWARE = [
    ...
    'corsheaders.middleware.CorsMiddleware',
    'django.middleware.common.CommonMiddleware',
    ...
]
```

/library/settings.py

И в settings.py проекта добавляем адреса, с которых будет возможен запрос.

```
CORS_ALLOWED_ORIGINS = [
   "http://localhost:3000",
]
```

/library/settings.py

Теперь сервер будет отвечать на запросы с http://localhost:3000, на котором располагается тестовый front-end сервер.

Отправка запроса с front-end на back-end

Есть много способов отправить запрос на back-end. Мы будем использовать библиотеку axios, это довольно удобный и распространённый способ.

Устанавливаем axios.

```
npm install axios
```

Помним, что с npm и front-end-частью мы работаем из папки front-end.

Отправка запроса с помощью axios имеет следующий вид:

```
axios.get(<url>)
       .then(response => {
              <действия с response>
       }).catch(error => <действия с error>)
```

Отправляем запрос на некоторый адрес. При успехе попадаем в блок then и работаем с объектом response. Если действие оказалось неудачным, попадаем в блок catch и работаем с ошибкой (объектом error).

Получение и обработка ответа с back-end

В файле App.js импортируем axios.

```
import axios from 'axios'
```

Изменим метод ComponentDidMount в классе App следующим образом:

```
componentDidMount() {
   axios.get('http://127.0.0.1:8000/api/authors')
       .then(response => {
           const authors = response.data
               this.setState(
               {
                   'authors': authors
               }
           )
       }).catch(error => console.log(error))
}
```

Мы получаем объект response и его данные response.data. Это и есть список авторов из API на back-end. Далее меняем состояние объекта App и передаём полученные данные вместо заглушек.

Запускаем тестовый сервер django на http://127.0.0.1:8000 и тестовый front-end сервер на http://localhost:3000.

Если ошибок нет, приложение работает с реальными данными и отрисовывает их на стороне клиента.

Итоги

На этом занятии мы научились:

1. Создавать структуру проекта на React с помощью create-react-app.

2. Запускать тестовый front-end-сервер.
3. Создавать приложение на React, состоящее из нескольких компонентов.
4. Разделять компоненты с состоянием и без него.
5. Получать данные с back-end.

Рекомендуем при создании других компонентов придерживаться принципов One Way Data Flow и соглашений о структуре файлов и папок в проекте.

Таким образом, теперь у нас есть понимание, как back-end взаимодействует с front-end. Мы знаем, как строится front-end-часть проекта. К её разработке вернёмся через несколько занятий. Рассмотрим, как создать SPA (Single Page Application) и как реагировать на действия пользователей, например, нажатие кнопки.

Глоссарий

1. **axios** — пакет javascript для удобной отправки http-запросов.
2. **CORS** (Cross-Origin Resource Sharing) — механизм, использующий дополнительные HTTP-заголовки. Это даёт возможность агенту пользователя получать разрешения на доступ к выбранным ресурсам с сервера на источнике (домене), отличном от того, что сайт использует в данный момент.
3. **create-react-app** — пакет для удобной работы с приложением на React.
4. **npx** — пакет для запуска пакетов без установки в систему.
5. **One Way Data Flow** — принцип разработки приложения, при котором данные хранятся в компоненте верхнего уровня и передаются дочерним сверху.
6. **React** — библиотека для создания пользовательских интерфейсов на javascript.

Дополнительные материалы

1. React CDN.
2. Официальный сайт node.js.
3. Мануал create-react-app.
4. npx.
5. Redux.
6. Flux Wiki.
7. django-cors-headers.

Используемые источники

1. Официальная документация React.
2. axios.
3. Thinking in react.
4. Статья One Way Data Flow.
5. Алекс Бэнкс, Ева Порселло «React и Redux функциональная веб-разработка».

Практическое задание

У нас есть API для работы с пользователями. Теперь необходимо создать для него интерфейс на React.

В этой самостоятельной работе мы тренируем умения

- создавать приложение на React;
- использовать компонентный подход;
- настраивать взаимодействие back-end и front-end;
- применять One Way Data Flow.

Зачем?

Для создания front-end-части приложений на React.

Последовательность действий

1. С помощью create-react-app создать приложение для front-end-части проекта.
2. На React создать страницу для отображения списка пользователей из нескольких компонентов. *Пока эта страница будет доступна всем, после разграничения прав и только для администратора.*
3. Добавить на страницу компоненты Menu и Footer.
4. В главном приложении получить данные обо всех пользователях и вывести их на странице.