**Министерство науки и высшего образования РФ**

**Государственное образовательное учреждение**

**высшего профессионального образования**

**Белгородский Государственный Технологический Университет им В. Г. Шухова**

Кафедра программного обеспечения вычислительной техники и

автоматизированных систем.

**Расчётно-графическое задание по дисциплине**

**«Технологии Web-программирования»**

Выполнил:

студент группы ВТ-41

Ковалёв И. Д.

Проверил: Картамышев С. В.

Белгород 2020 г.

**Оглавление**

[Цель работы 3](#_Toc60071316)

[Выполнение 3](#_Toc60071317)

[Глава 1. HTML-страницы 4](#_Toc60071318)

[Глава 3. Серверная часть. Приложение Django 10](#_Toc60071319)

[Глава 4. Серверная часть. Разработка и проектирование базы данных 13](#_Toc60071320)

[Глава 5. Серверная часть. REST API 15](#_Toc60071321)

[Глава 6. Работа с HTTP-запросами 17](#_Toc60071322)

[Вывод 21](#_Toc60071323)

# **Цель работы**

Целью расчетно-графического задания является изучение технологий web-программирования, а именно:

1. Изучение работы протокола передачи гипертекста HTTP и языка CSS,
2. Изучение основ разработки frontend-приложений, в частности, с использованием языка Javascript и библиотеки React,
3. Ознакомление с основами работы docker,
4. Изучение основ разработки REST API, в частности, с использованием Django, Django-rest-framework,
5. Изучение основ взаимодействия frontend-приложения с backend-приложением.

# **Выполнение**

В качестве предметной области был выбран сервис для поиска музыки.

1. Данный сайт должен предоставлять такие возможности:
2. Просмотр исполнителей, их альбомов
3. Поиск по исполнителям и альбомам
4. Возможность добавление альбомов и исполнителей в избранное

# **Глава 1. HTML-страницы**

В первую очередь были сверстаны html-страницы. Для упрощения верстки был использован bootstrap.

В результате, были созданы такие макеты:

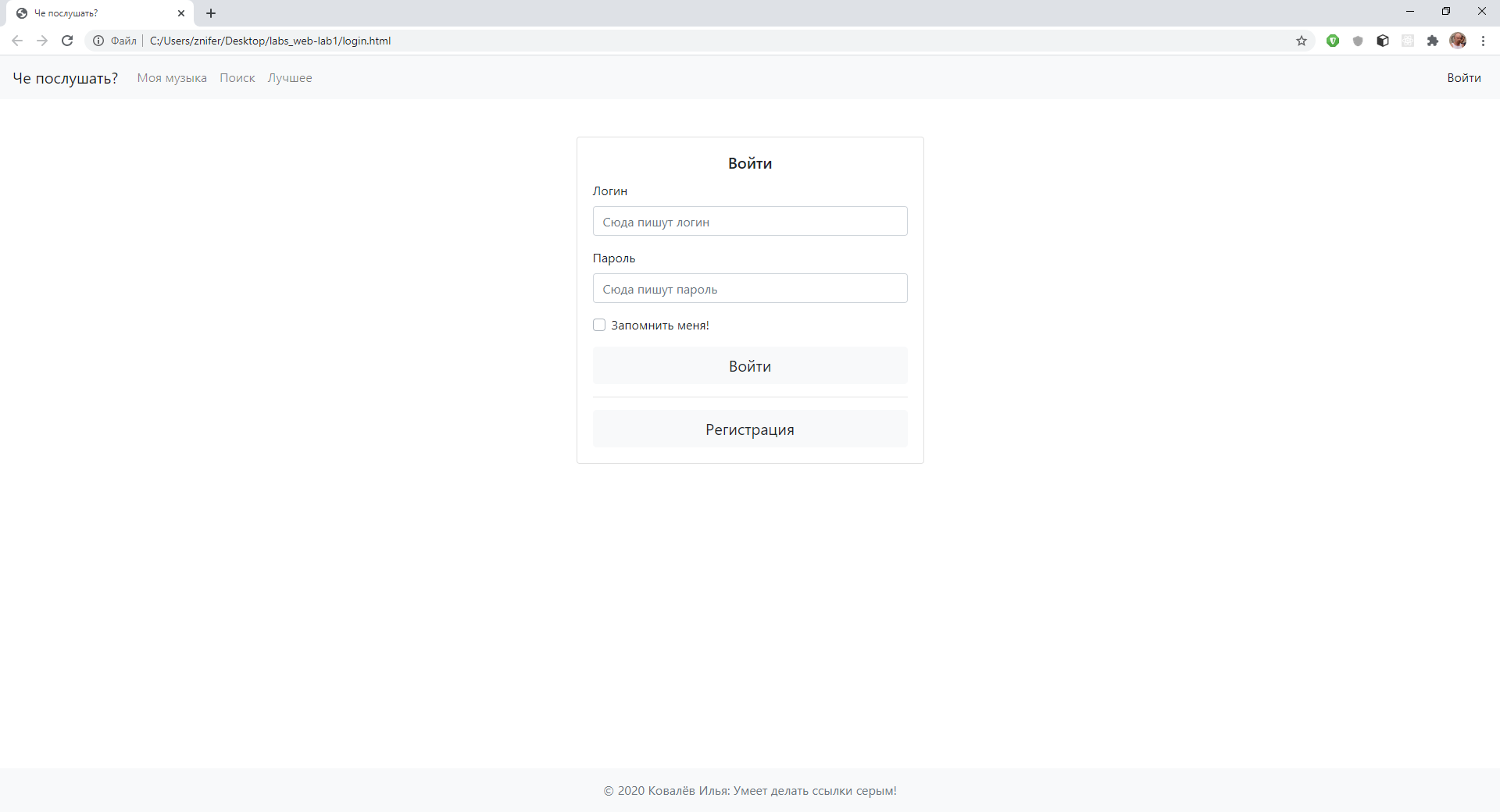


Рис 1.1 Страница авторизации

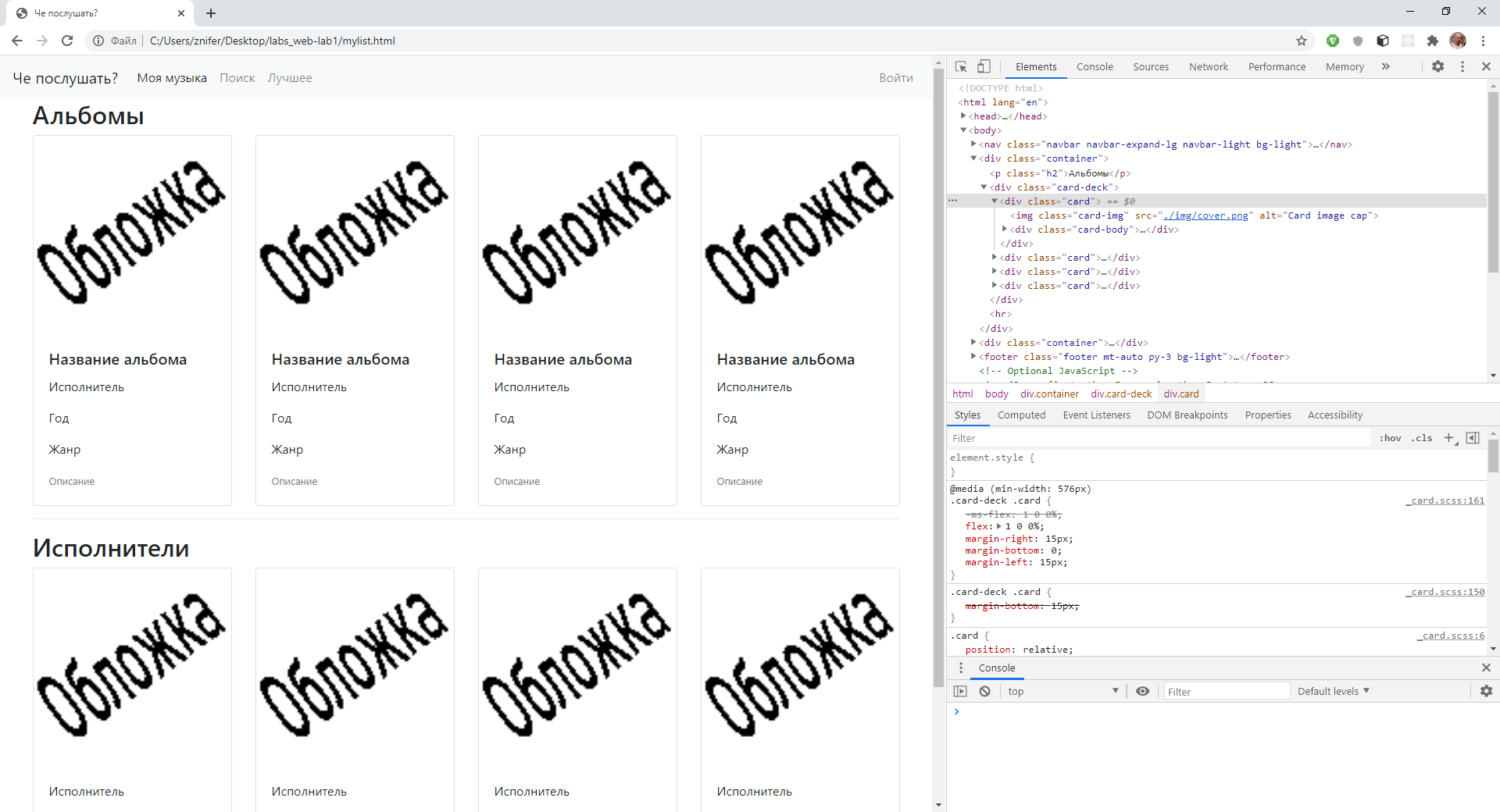


Рис 1.2 Контентная страница

**Содержимое файла Mylist.html:**

<!doctype html>

<html lang="en">

    <head>

<title>Че послушать?</title>

    </head>

    <body>

        <nav class="navbar navbar-expand-lg navbar-light bg-light">

            <a class="navbar-brand" href="#">Че послушать?</a>

            <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarText" aria-controls="navbarText" aria-expanded="false" aria-label="Toggle navigation">

                <span class="navbar-toggler-icon"></span>

            </button>

            <div class="collapse navbar-collapse" id="navbarText">

                <ul class="navbar-nav mr-auto w-100">

                    <li class="nav-item">

                        <a class="nav-link active" href="mylist.html">Моя музыка</a>

                    </li>

                    <li class="nav-item">

                        <a class="nav-link" href="search.html">Поиск</a>

                    </li>

                    <li class="nav-item">

                        <a class="nav-link" href="top.html">Лучшее</a>

                    </li>

                </ul>

                <ul class="navbar-nav mr-auto justify-content-end">

                    <li id="login-link">

                        <a class="nav-link" href="login.html">Войти</a>

                    </li>

                </ul>

            </div>

        </nav>

        <div class="container">

            <p class="h2">Альбомы</p>

            <div class="card-deck">

                <div class="card">

                    <img class="card-img" src="./img/cover.png" alt="Card image cap">

                    <div class="card-body">

                        <h5 class="card-title">Название альбома</h5>

                        <p class="card-text">Исполнитель</p>

                        <p class="card-text">Год</p>

                        <p class="card-text">Жанр</p>

                        <p class="card-text"><small class="text-muted">Описание</small></p>

                    </div>

                </div>

                <div class="card">

                    <img class="card-img" src="./img/cover.png" alt="Card image cap">

                    <div class="card-body">

                        <h5 class="card-title">Название альбома</h5>

                        <p class="card-text">Исполнитель</p>

                        <p class="card-text">Год</p>

                        <p class="card-text">Жанр</p>

                        <p class="card-text"><small class="text-muted">Описание</small></p>

                    </div>

                </div>

                <div class="card">

                    <img class="card-img" src="./img/cover.png" alt="Card image cap">

                    <div class="card-body">

                        <h5 class="card-title">Название альбома</h5>

                        <p class="card-text">Исполнитель</p>

                        <p class="card-text">Год</p>

                        <p class="card-text">Жанр</p>

                        <p class="card-text"><small class="text-muted">Описание</small></p>

                    </div>

                </div>

                <div class="card">

                    <img class="card-img" src="./img/cover.png" alt="Card image cap">

                    <div class="card-body">

                        <h5 class="card-title">Название альбома</h5>

                        <p class="card-text">Исполнитель</p>

                        <p class="card-text">Год</p>

                        <p class="card-text">Жанр</p>

                        <p class="card-text"><small class="text-muted">Описание</small></p>

                    </div>

                </div>

            </div>

        <hr>

        </div>

        <div class="container">

            <p class="h2">Исполнители</p>

            <div class="card-deck">

                <div class="card">

                    <img class="card-img" src="./img/cover.png" alt="Card image cap">

                    <div class="card-body">

                        <p class="card-text">Исполнитель</p>

                        <p class="card-text">Жанр</p>

                        <p class="card-text"><small class="text-muted">Описание</small></p>

                    </div>

                </div>

                <div class="card">

                    <img class="card-img" src="./img/cover.png" alt="Card image cap">

                    <div class="card-body">

                        <p class="card-text">Исполнитель</p>

                        <p class="card-text">Жанр</p>

                        <p class="card-text"><small class="text-muted">Описание</small></p>

                    </div>

                </div>

                <div class="card">

                    <img class="card-img" src="./img/cover.png" alt="Card image cap">

                    <div class="card-body">

                        <p class="card-text">Исполнитель</p>

                        <p class="card-text">Жанр</p>

                        <p class="card-text"><small class="text-muted">Описание</small></p>

                    </div>

                </div>

                <div class="card">

                    <img class="card-img" src="./img/cover.png" alt="Card image cap">

                    <div class="card-body">

                        <p class="card-text">Исполнитель</p>

                        <p class="card-text">Жанр</p>

                        <p class="card-text"><small class="text-muted">Описание</small></p>

                    </div>

                </div>

            </div>

        <hr>

        </div>

        <footer class="footer mt-auto py-3 bg-light">

            <div class="container text-center text-muted">

                © 2020 Ковалёв Илья:

                <a href="https://github.com/znifer/" class="text-muted">Умеет делать ссылки серым!</a>

            </span>

        </footer>

</body>

</html>

Глава 2. React-приложение

Полученные макеты были перенесены на JavaScript, была добавлена логика переходов по страницам.

В результате, было получено приложение, реализующее базовый функционал проекта:

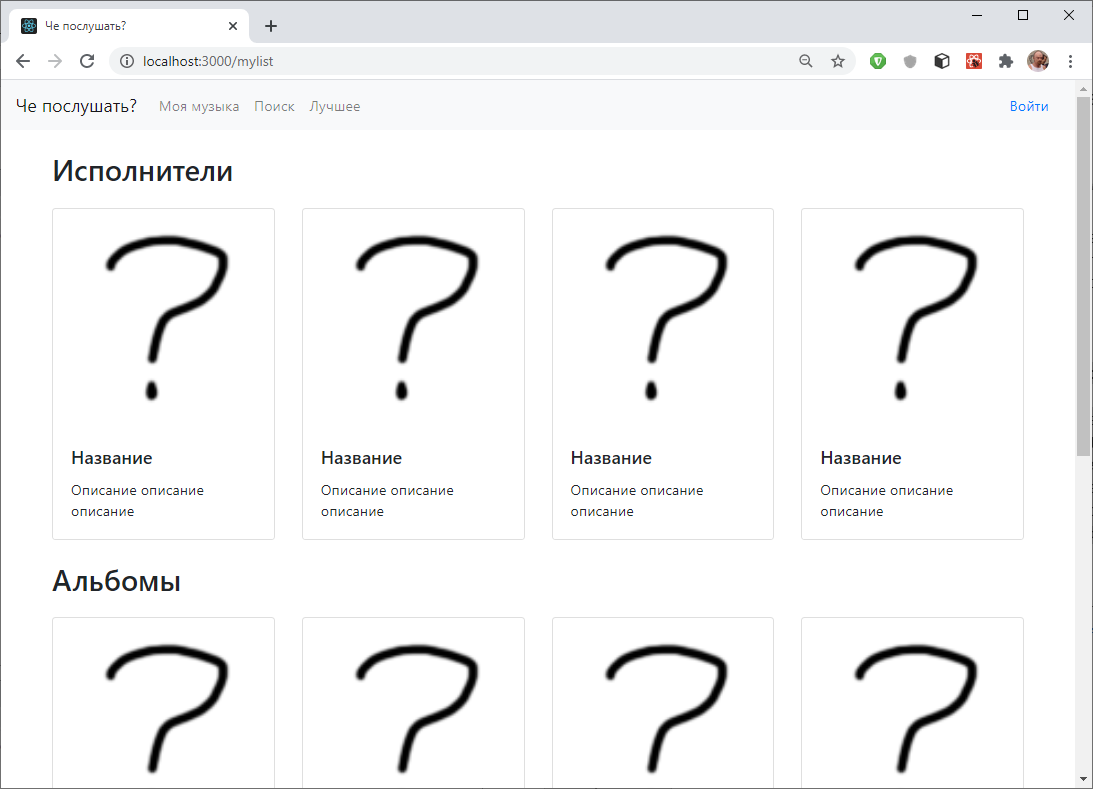


Рис. 2.1 Контентная страница

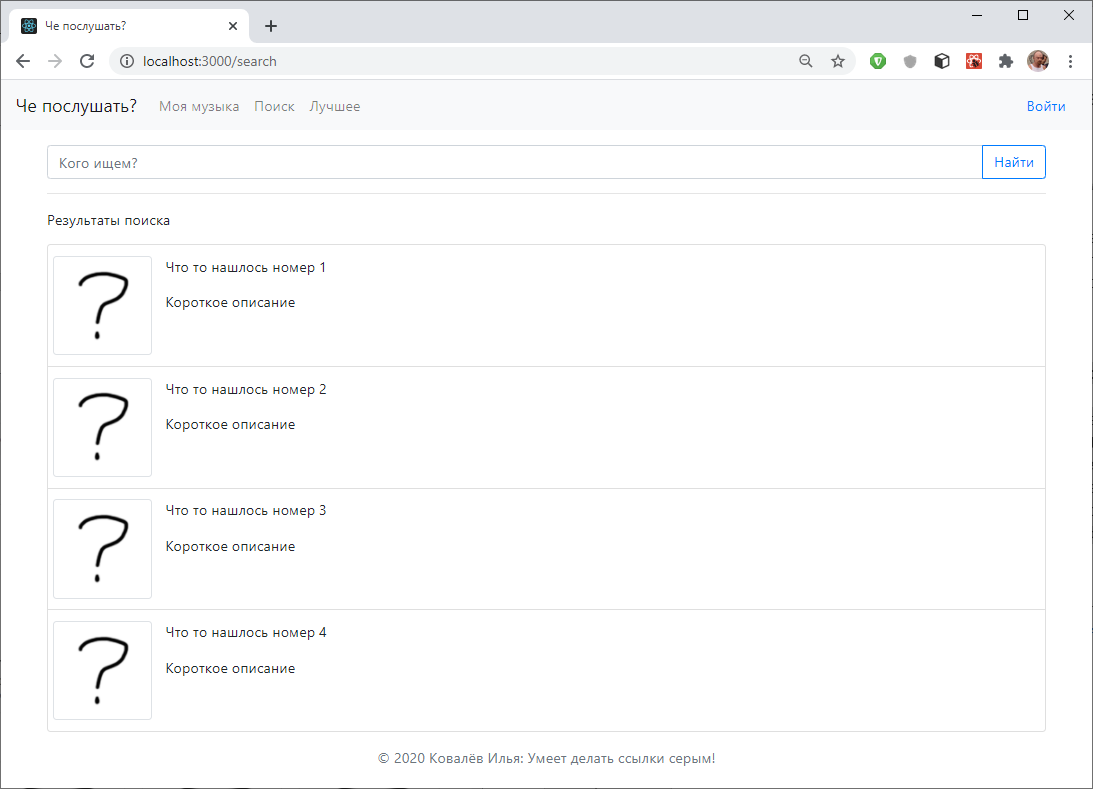


Рис. 2.2 Страница поиска

**Листинг разработанного приложения:**

**App.js:**

import 'bootstrap/dist/css/bootstrap.min.css'

import './index.css'

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

import Header from "./Header"

import Footer from "./Footer"

import LoginForm from "./Loginform"

import RegistrationForm from "./Registrationform"

import Search from "./Search"

import MyList from "./MyList"

import Top from "./Top"

const App = () => {

    return (

        <BrowserRouter>

            <Header></Header>

            <Switch>

                <Route exact path="/login" component={LoginForm} />

                <Route exact path="/registration" component={RegistrationForm} />

                <Route exact path="/search" component={Search} />

                <Route exact path="/mylist" component={MyList} />

                <Route exact path="/top" component={Top} />

            </Switch>

            <Footer></Footer>

        </BrowserRouter>

    )

}

export default App;

**Footer.js:**

import { Container } from 'react-bootstrap'

const Footer = () => {

    return (

        <Container className="mt-auto py-3 bottom">

            <Container className="text-center text-muted">

                <span>© 2020 Ковалёв Илья: </span>

                <a href="https://github.com/znifer/" class="text-muted">Умеет делать ссылки серым!</a>

            </Container>

        </Container>

    )

}

export default Footer

**Header.js:**

import {Navbar, Nav, Button} from 'react-bootstrap'

const Header = () => {

    return (

        <Navbar bg="light" expand="lg">

            <Navbar.Brand>Че послушать?</Navbar.Brand>

            <Nav className="mr-auto">

                <Nav.Link href="mylist">Моя музыка</Nav.Link>

                <Nav.Link href="search">Поиск</Nav.Link>

                <Nav.Link href="top">Лучшее</Nav.Link>

            </Nav>

            <Button variant="link" href="login">Войти</Button>

        </Navbar>

    )

}

export default Header;

# **Глава 3. Серверная часть. Приложение Django**

Для реализации серверной части приложения был использован Python и фреймворки Django и Django-rest-framework.

Было развернуто базовое приложение, а также настроены переменные окружения для работы с docker-compose и реализовано API.

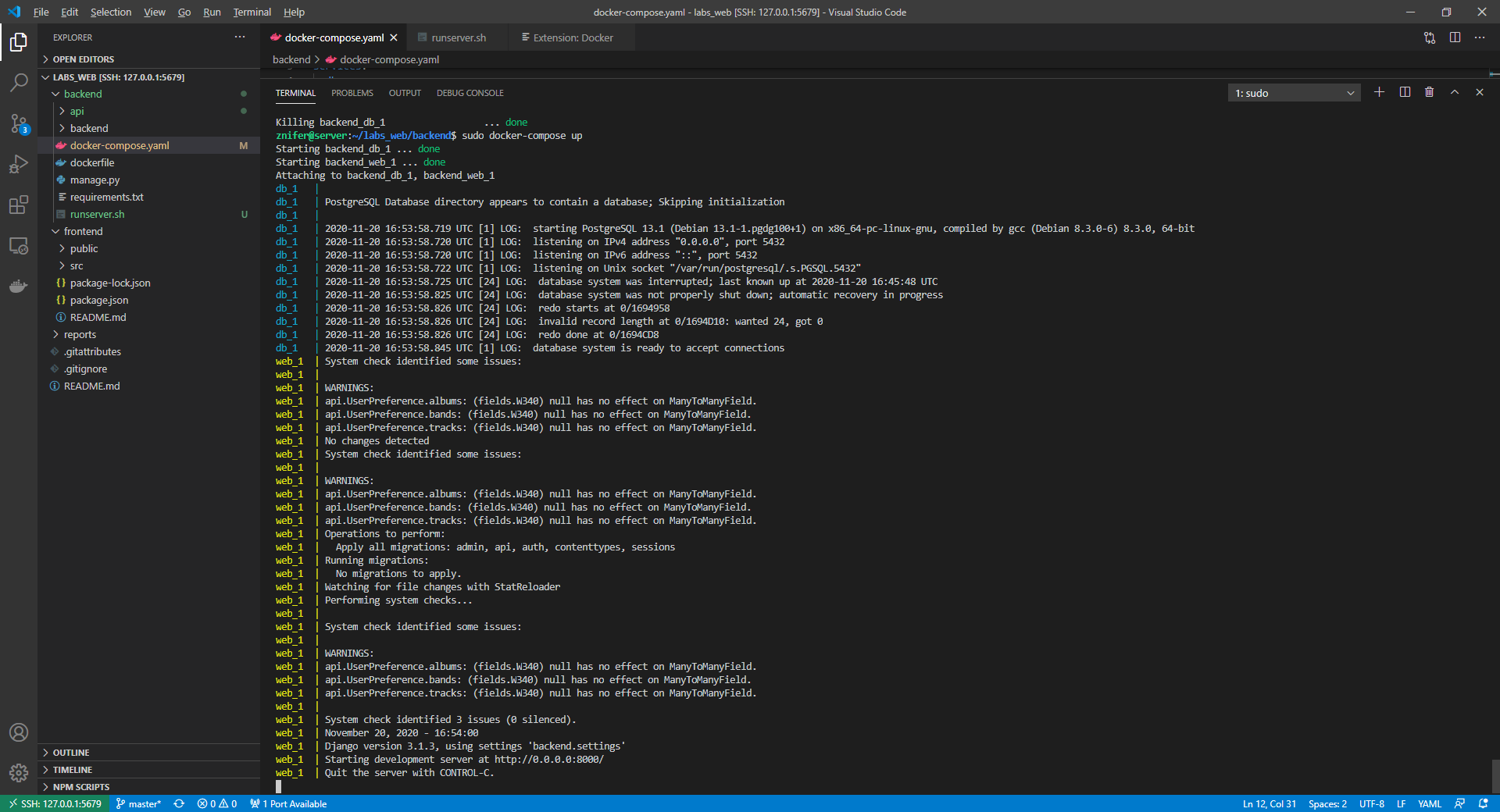


Рис. 3.1 Результат выполнения docker-compose

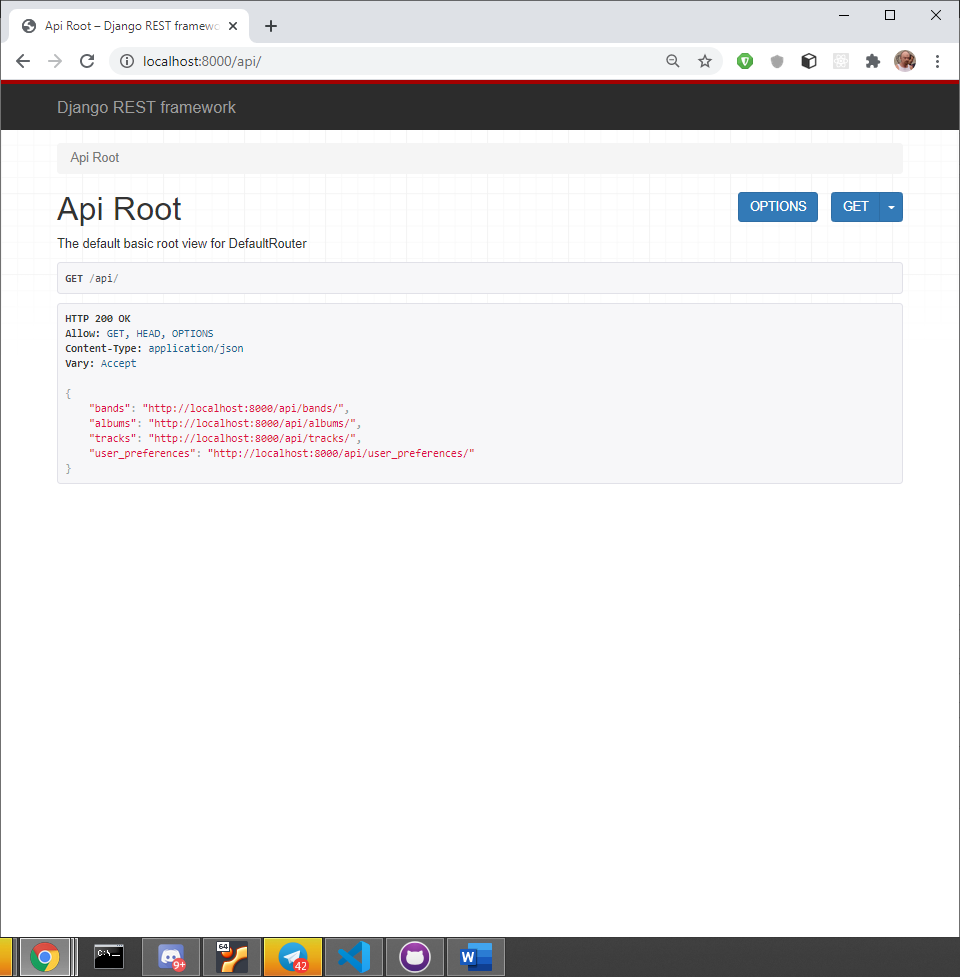


Рис. 3.2 Развернутое API

Содержимое dockerfile:

FROM python:3

ENV PYTHONUNBUFFERED=1

WORKDIR /code

COPY requirements.txt /code/

RUN pip install -r requirements.txt

COPY . /code/

Содержимое docker-compose.yaml:

version: "3.8"

services:

  db:

    image: postgres

    environment:

      - POSTGRES\_DB=postgres

      - POSTGRES\_USER=postgres

      - POSTGRES\_PASSWORD=postgres

  web:

    build: .

    command: sh ./runserver.sh

    volumes:

      - .:/code

    ports:

      - "8000:8000"

    depends\_on:

      - db

Скрипт runserver.sh, выполняющий миграции на сервере и запускающий его:

python manage.py makemigrations

python manage.py migrate

python manage.py runserver 0.0.0.0:8000

Содержимое файла settings.py, конфигурирующий работу серверного приложения:

import os

from datetime import timedelta

BASE\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))

SECRET\_KEY = 'r091bbiu##n6%b0n)a3o3rbwtti^0\*h7uqw7!@2nom4dk%f(&e'

DEBUG = True

ALLOWED\_HOSTS = []

# Application definition

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'rest\_framework',

    'corsheaders',

    'api.apps.ApiConfig',

    'django\_extensions',

    'django\_filters',

]

MIDDLEWARE = [

    'django.middleware.security.SecurityMiddleware',

    'django.contrib.sessions.middleware.SessionMiddleware',

    'django.middleware.common.CommonMiddleware',

    'django.middleware.csrf.CsrfViewMiddleware',

    'django.contrib.auth.middleware.AuthenticationMiddleware',

    'django.contrib.messages.middleware.MessageMiddleware',

    'django.middleware.clickjacking.XFrameOptionsMiddleware',

]

ROOT\_URLCONF = 'backend.urls'

TEMPLATES = [

    {

        'BACKEND': 'django.template.backends.django.DjangoTemplates',

        'DIRS': [],

        'APP\_DIRS': True,

        'OPTIONS': {

            'context\_processors': [

                'django.template.context\_processors.debug',

                'django.template.context\_processors.request',

                'django.contrib.auth.context\_processors.auth',

                'django.contrib.messages.context\_processors.messages',

            ],

        },

    },

]

WSGI\_APPLICATION = 'backend.wsgi.application'

SIMPLE\_JWT = {

    'ACCESS\_TOKEN\_LIFETIME': timedelta(minutes=5),

    'REFRESH\_TOKEN\_LIFETIME': timedelta(days=1),

    'ROTATE\_REFRESH\_TOKENS': False,

    'BLACKLIST\_AFTER\_ROTATION': True,

    'ALGORITHM': 'HS256',

    'SIGNING\_KEY': SECRET\_KEY,

    'VERIFYING\_KEY': None,

    'AUDIENCE': None,

    'ISSUER': None,

    'AUTH\_HEADER\_TYPES': ('Bearer',),

    'USER\_ID\_FIELD': 'id',

    'USER\_ID\_CLAIM': 'user\_id',

    'AUTH\_TOKEN\_CLASSES': ('rest\_framework\_simplejwt.tokens.AccessToken',),

    'TOKEN\_TYPE\_CLAIM': 'token\_type',

    'JTI\_CLAIM': 'jti',

    'SLIDING\_TOKEN\_REFRESH\_EXP\_CLAIM': 'refresh\_exp',

    'SLIDING\_TOKEN\_LIFETIME': timedelta(minutes=60),

    'SLIDING\_TOKEN\_REFRESH\_LIFETIME': timedelta(days=1),

}

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.postgresql',

        'NAME': 'postgres',

        'USER': 'postgres',

        'PASSWORD': 'postgres',

        'HOST': 'db',

        'PORT': 5432,

    }

}

LANGUAGE\_CODE = 'en-us'

TIME\_ZONE = 'UTC'

USE\_I18N = True

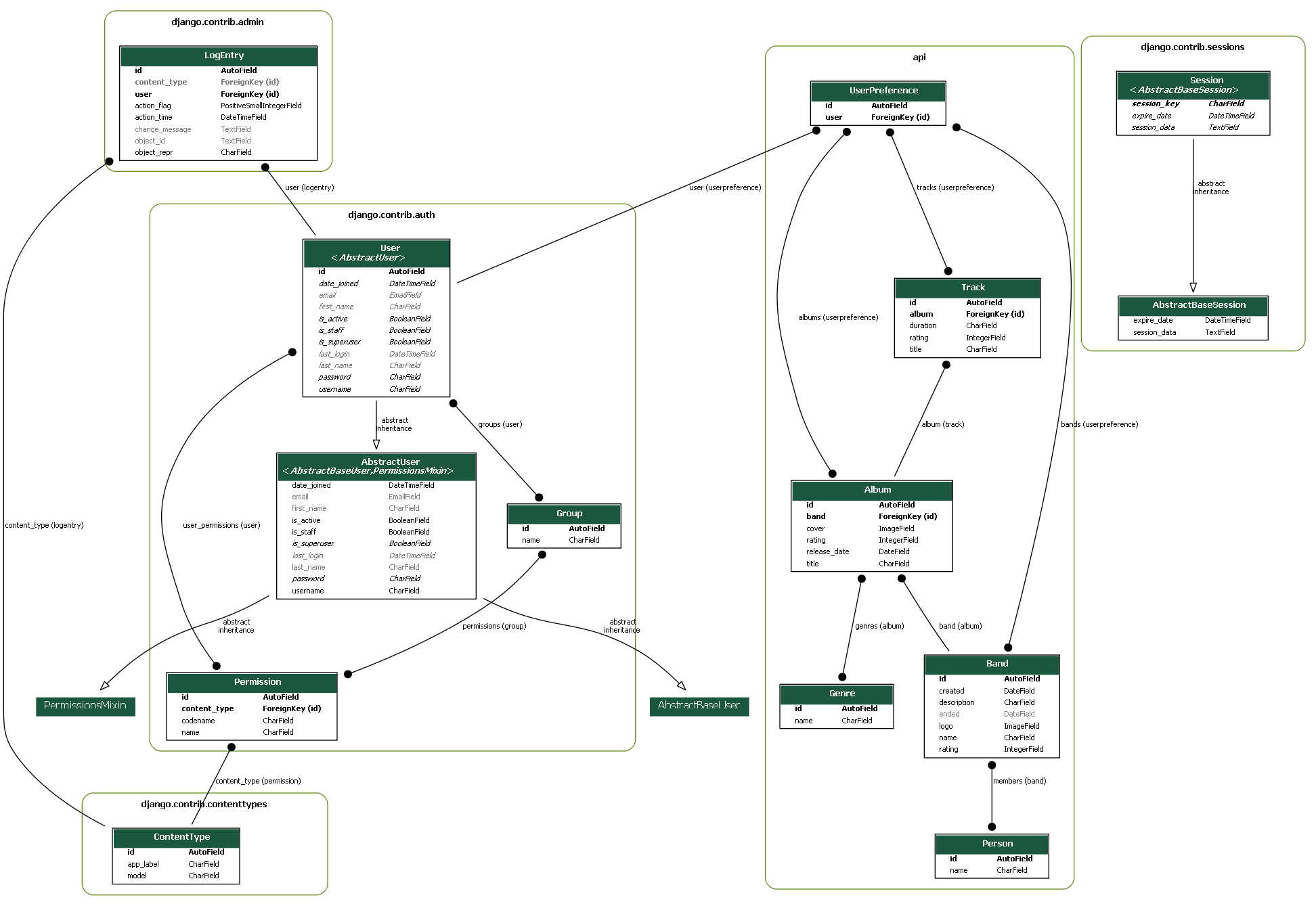
USE\_L10N = True

USE\_TZ = True

STATIC\_URL = '/static/'

# **Глава 4. Серверная часть. Разработка и проектирование базы данных**

Была разработана структура БД приложения, которая имеет вид:



Содержимое файла models.py:

from django.db import models

from django.contrib.auth.models import User

from django.conf import settings

class Person(models.Model):

    name = models.CharField(max\_length=255)

class Band(models.Model):

    name = models.CharField(max\_length=255)

    description = models.CharField(max\_length=10000)

    created = models.DateField()

    ended = models.DateField(blank=True, null=True)

    logo = models.ImageField(upload\_to=settings.MEDIA\_ROOT)

    members = models.ManyToManyField(

        'Person'

    )

    rating = models.IntegerField()

    def \_\_str\_\_(self):

        return "Исполнитель: {}".format(self.name)

class Album(models.Model):

    title = models.CharField(max\_length=255)

    release\_date = models.DateField()

    cover = models.ImageField(upload\_to=settings.MEDIA\_ROOT)

    rating = models.IntegerField()

    band = models.ForeignKey(

        'Band',

        on\_delete=models.CASCADE

    )

    genres = models.ManyToManyField(

        'Genre'

    )

    def \_\_str\_\_(self):

        return "Альбом: {}".format(self.title)

class Genre(models.Model):

    name = models.CharField(max\_length=255)

class Track(models.Model):

    title = models.CharField(max\_length=255)

    duration = models.CharField(max\_length=255)

    rating = models.IntegerField()

    album = models.ForeignKey(

        'Album',

        on\_delete=models.CASCADE

    )

    def \_\_str\_\_(self):

        return "{} - {}".format(self.title, self.duration)

class UserPreference(models.Model):

    user = models.ForeignKey(

        User,

        on\_delete=models.CASCADE

    )

    bands = models.ManyToManyField(

        'Band',

        blank=True,

        null=True

    )

    albums = models.ManyToManyField(

        Album,

        blank=True,

        null=True

    )

    tracks = models.ManyToManyField(

        Track,

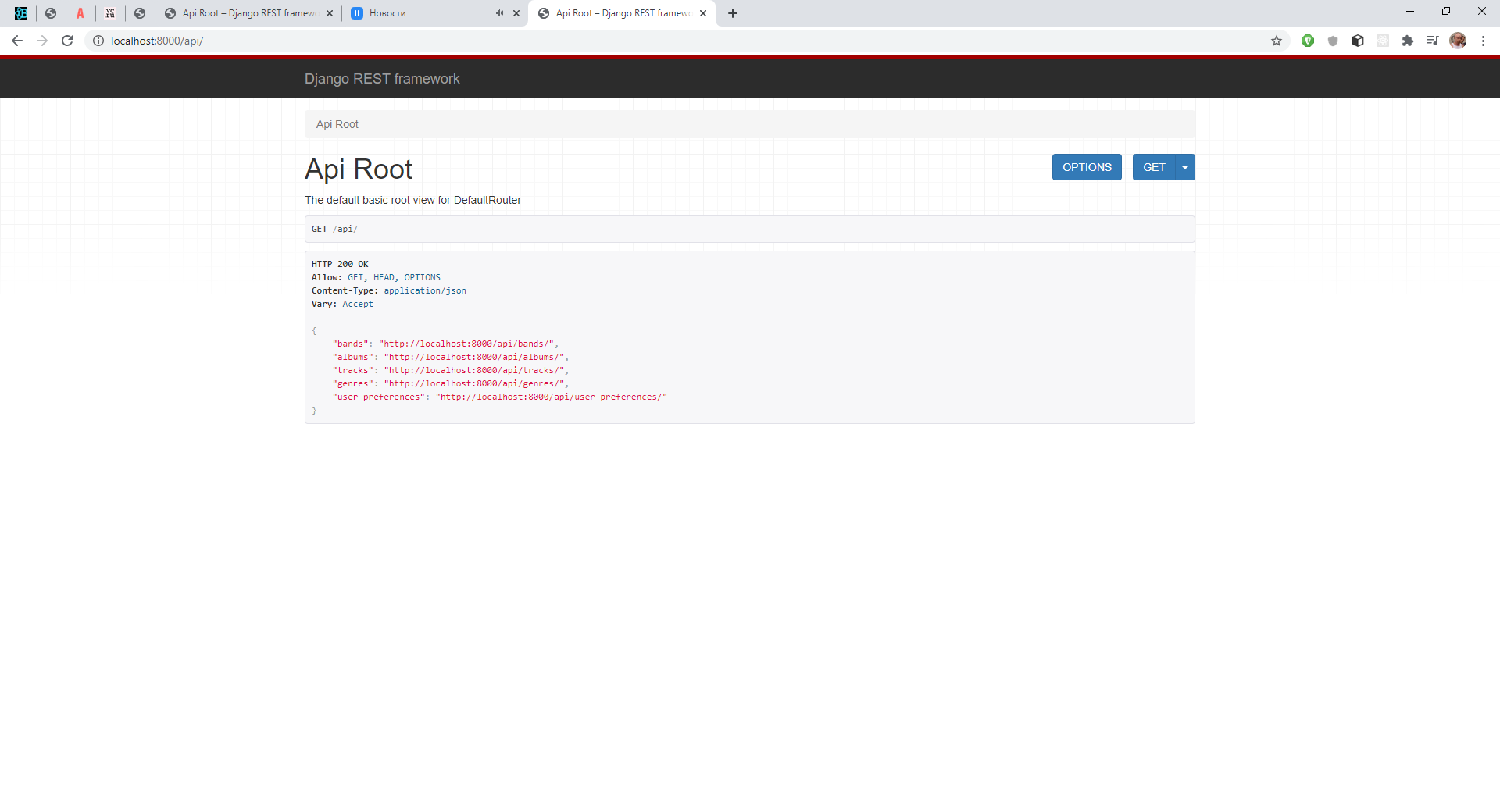
        blank=True,

        null=True

    )

# **Глава 5. Серверная часть. REST API**

Были созданы такие URL:



Была создана документация по API с помощью drf-yasg, которая доступна по <http://localhost:8000/api/swagger/> или http://localhost:8000/api/redoc/

**Содержимое файла serializers.py:**

from rest\_framework import serializers

from .models import \*

class BandSerializer(serializers.ModelSerializer):

    class Meta:

        model = Band

        fields = '\_\_all\_\_'

class TrackSerializer(serializers.ModelSerializer):

    class Meta:

        model = Track

        fields = '\_\_all\_\_'

class AlbumSerializer(serializers.ModelSerializer):

    class Meta:

        model = Album

        fields = '\_\_all\_\_'

class UserPreferenceSerializer(serializers.ModelSerializer):

    # Отдавать данные пользователя поприкольнее?

    class Meta:

        model = UserPreference

        fields = [

            'user',

            'artists',

            'albums',

            'tracks'

        ]

class GenreSerializer(serializers.ModelSerializer):

    class Meta:

        model = Genre

        fields = '\_\_all\_\_'

class PersonSerializer(serializers.ModelSerializer):

    class Meta:

        model = Person

        fields = '\_\_all\_\_'

**Содержимое файла views.py:**

from django\_filters.rest\_framework import DjangoFilterBackend

from rest\_framework import viewsets

from rest\_framework.permissions import IsAuthenticated

from .serializers import \*

from .models import \*

class BandViewSet(viewsets.ModelViewSet):

    queryset = Band.objects.all()

    serializer\_class = BandSerializer

class TrackViewSet(viewsets.ModelViewSet):

    queryset = Track.objects.all()

    serializer\_class = TrackSerializer

class AlbumViewSet(viewsets.ModelViewSet):

    queryset = Album.objects.all()

    serializer\_class = AlbumSerializer

class UserPreferenceViewSet(viewsets.ModelViewSet):

    #queryset = UserPreference.objects.all()

    serializer\_class = UserPreferenceSerializer

    # Отдаем только аутентифицированному пользователю

    permission\_classes = [IsAuthenticated]

    # Все, что его интересует

    def get\_queryset(self):

        return UserPreference.objects.filter(user=self.request.user)

class GenreViewSet(viewsets.ModelViewSet):

    queryset = Genre.objects.all()

    serializer\_class = GenreSerializer

class PersonViewSet(viewsets.ModelViewSet):

    queryset = Person.objects.all()

    serializer\_class = PersonSerializer

# **Глава 6. Работа с HTTP-запросами**

Для обращения к API будем использовать axios.

Был создан класс ApiService AuthService для работы с API.

Листинг ApiService.js:

export default class ApiService {

    \_apiBase = 'http://localhost:8000/api'

    async getResource(url) {

        try{

            const response = await axios({

                method: 'get',

                url: `${this.\_apiBase}${url}`,

            });

            return response.data;

        }

        catch (e){

            return null

        }

    }

    async getAllAlbums() {

        const res = await this.getResource('/albums/')

        return res

    }

    async getBestAlbums() {

        const res = await this.getResource('/top\_albums/')

        return res

    }

    async getAlbum(id) {

        const res = await this.getResource(`/albums/${id}/`)

        return res

    }

    async getAllbands() {

        const res = await this.getResource('/bands/')

        return res

    }

    async getBand(id) {

        const res = await this.getResource(`/bands/${id}/`)

        return res

    }

    async searchBand(searchRequest) {

        const res = await this.getResource(`/bands/?search=${searchRequest}`)

        return res

    }

    async searchAlbum(searchRequest) {

        const res = await this.getResource(`/albums/?search=${searchRequest}`)

        return res

    }

    async getPreferences(){

        const accessToken = localStorage.getItem('access')

            const response = await axios({

                method: 'get',

                url: `${this.\_apiBase}/user\_preferences/`,

                headers: {

                    "Authorization" : `Bearer ${accessToken}`

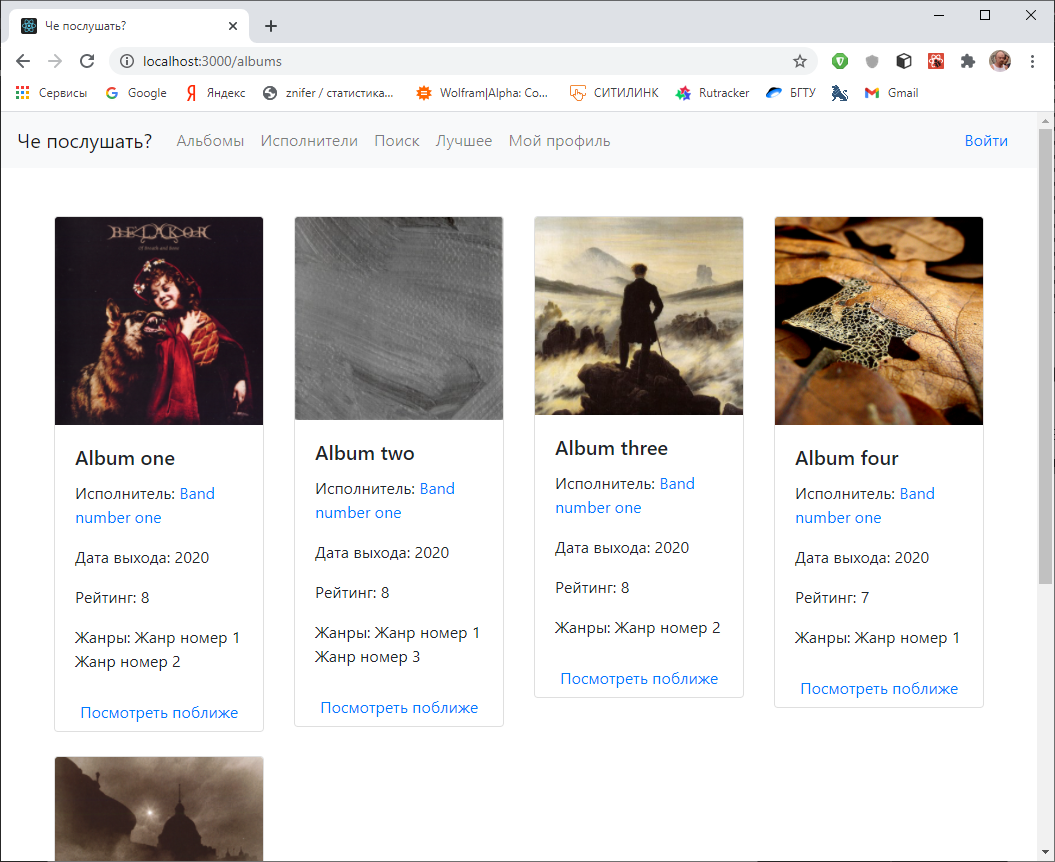
                }

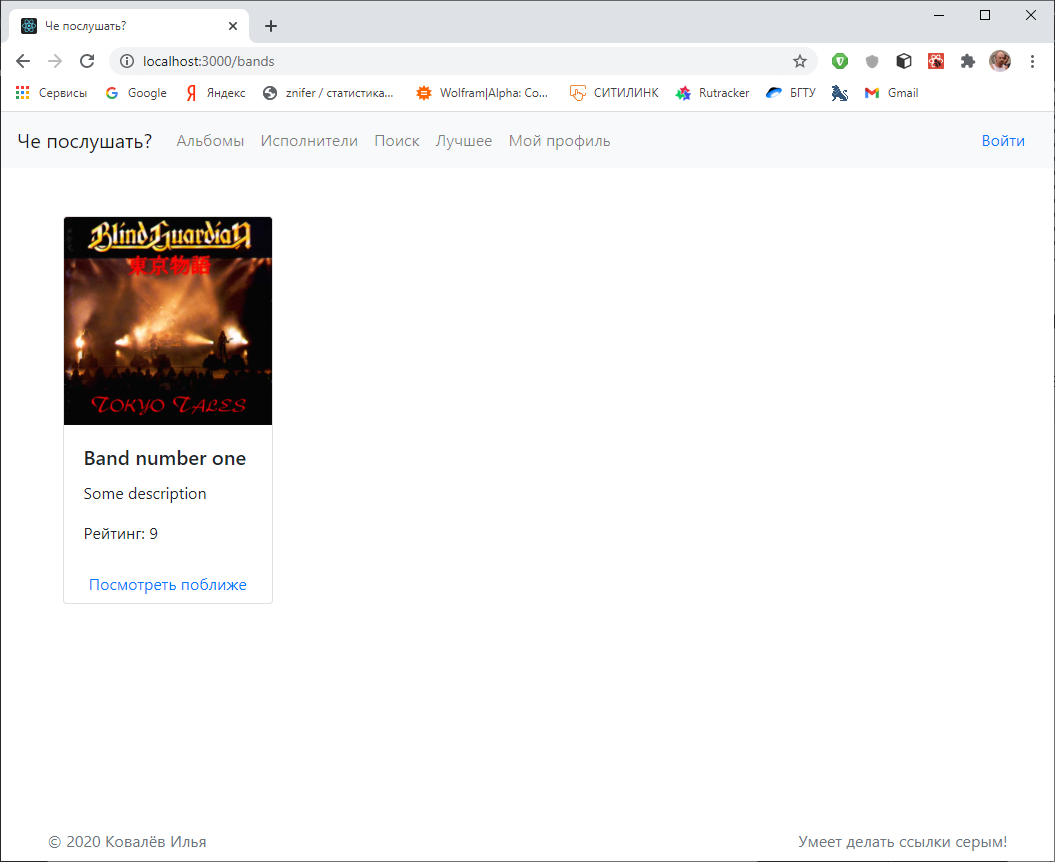
            });

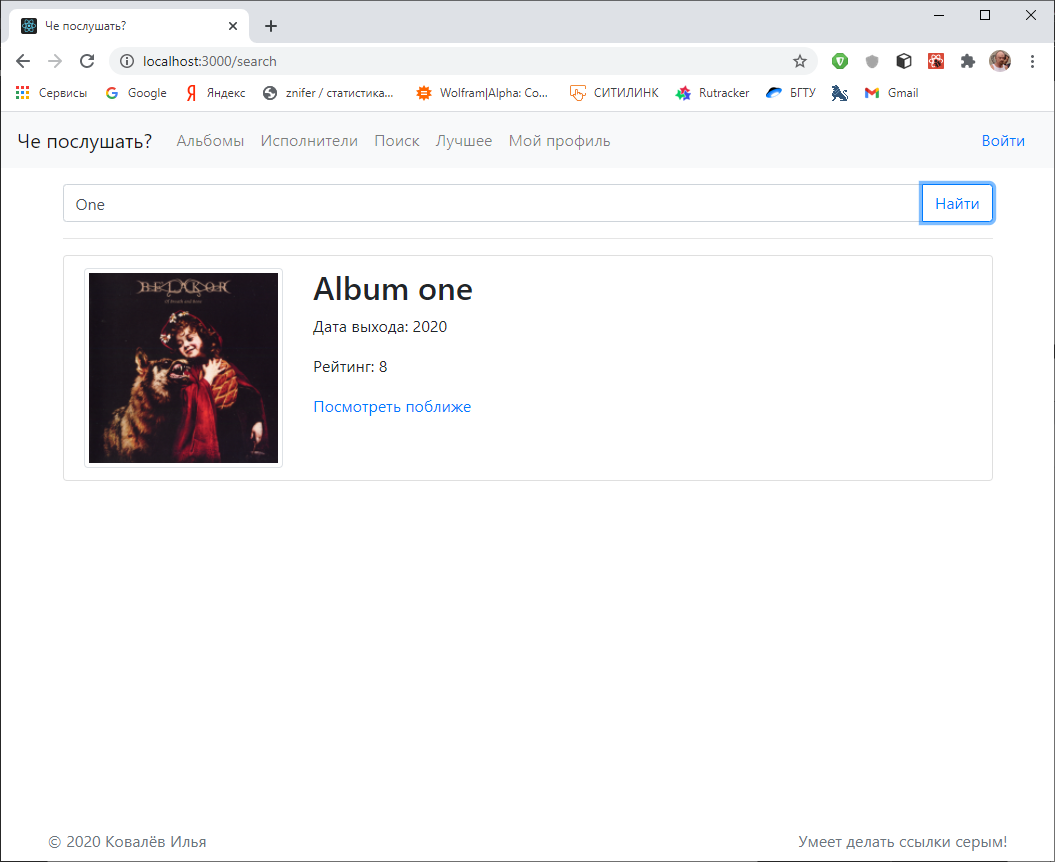
            return response.data;

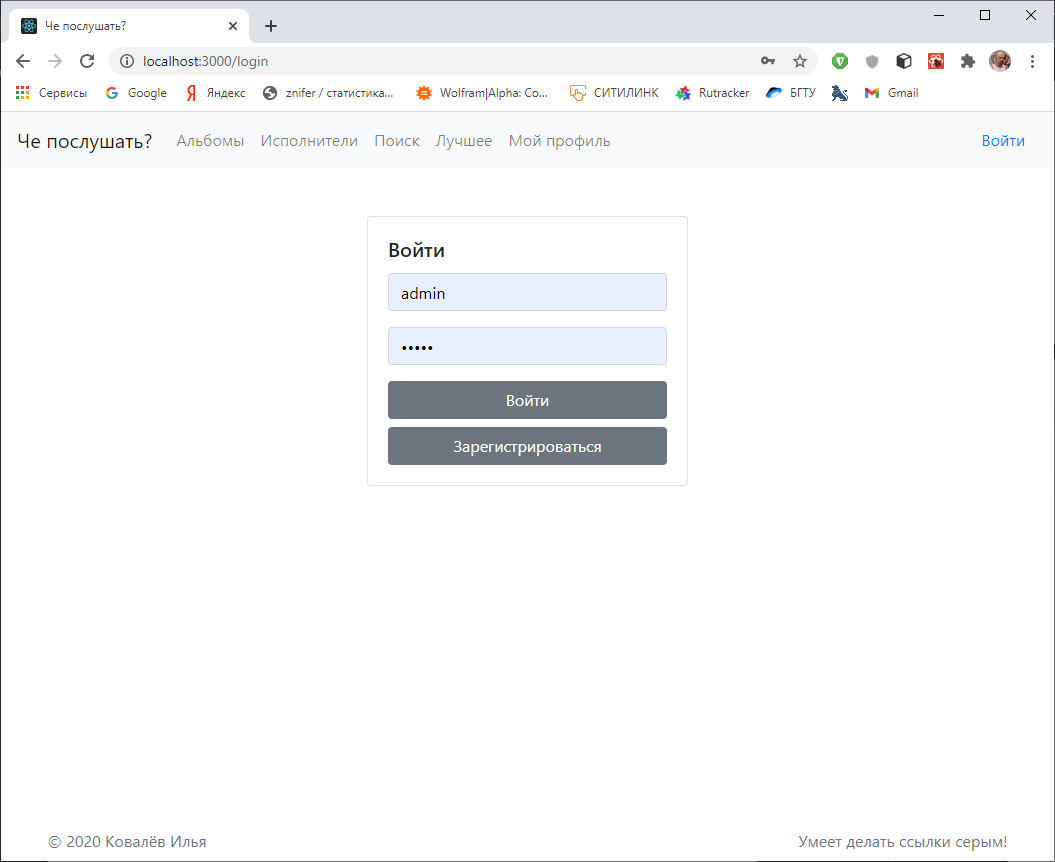
}

В результате было получено web-приложение, имеющее такой вид:









# **Вывод**

В результате выполнения расчетно-графического задания было разработано web-приложение, а также изучен принцип работы протокола передачи гипертекста HTTP и языка CSS, изучены основы разработки frontend-приложений, в частности, с использованием языка Javascript и библиотеки React, изучены основы работы docker, основы разработки REST API, в частности, с использованием Django, Django-rest-framework и основы взаимодействия frontend-приложения с backend-приложением.