МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»
(БГТУ им. В.Г. Шухова)

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Расчётно-графическое задание

По дисциплине: Технологии Web-программирования

Тема: «**Web приложение каталог радио**»

Выполнил:
студент группы ПВ-192
Витохин К.А.
Проверил:
Картамышев С.В.

Белгород 2022

Оглавление

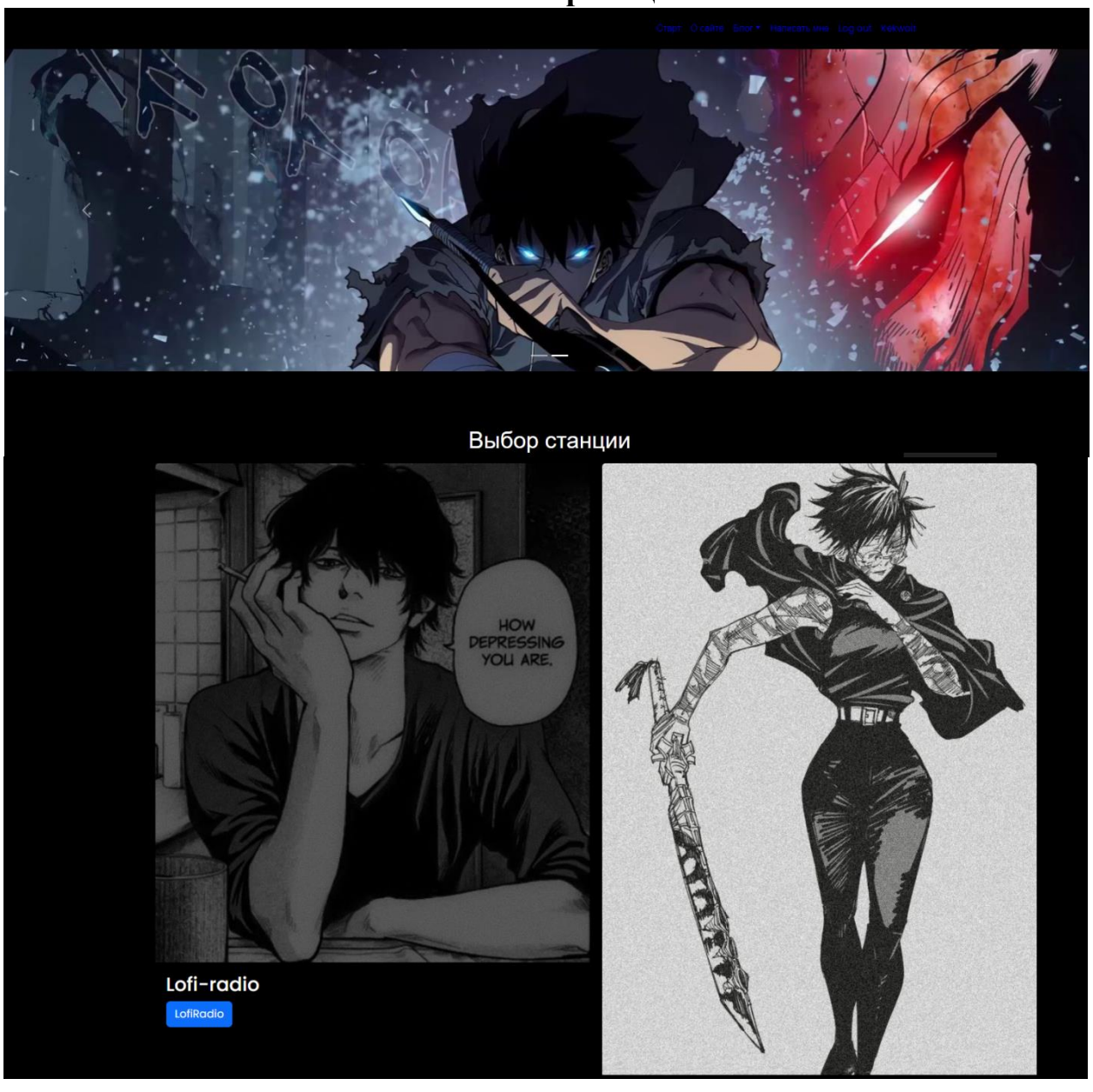
Цель работы: создать web приложение каталог радио.

Ход работы

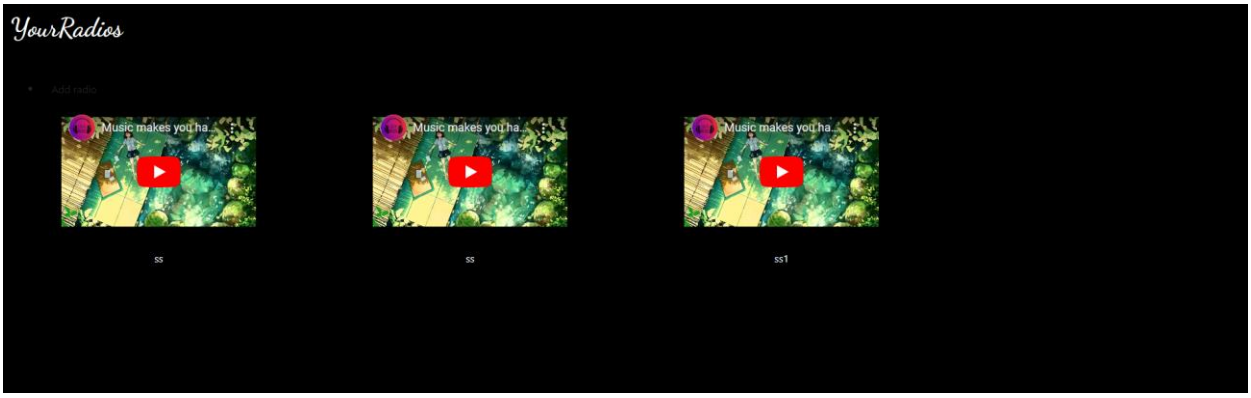
HTML. Разработка макетов и верстка шаблонов web-приложения с помощью языков HTML и CSS

1. Выбрать шаблон или разработать свой для web-приложения. Шаблон должен включать минимум страницы (главная, контентная страница, страница авторизации/регистрации).
2. Сделать макеты страниц с помощью языка разметки HTML.
3. Стилизовать страницы с помощью языка CSS.
4. Продемонстрировать внешний вид разработанных страниц.

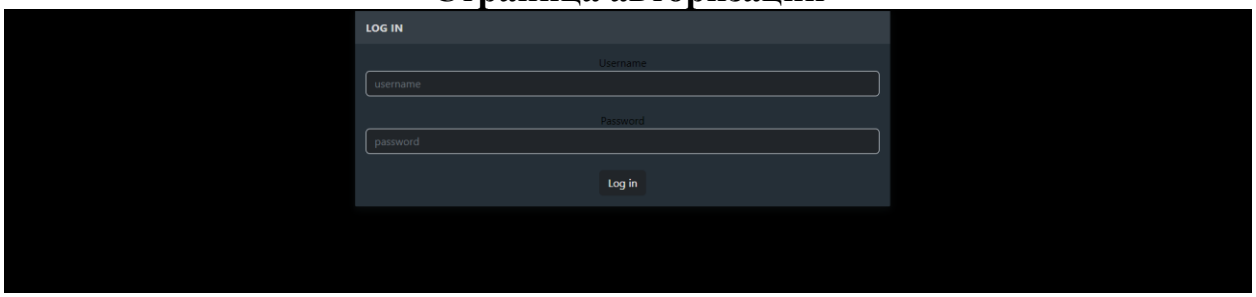
Главная страница



Контентная страница



Страница авторизации



1. Изучить основы разработки на языке JavaScript.
2. Изучить основы разработки frontend-приложения.
3. Развернуть базовое приложение на фреймворке.
4. Добавить необходимые компоненты и перенести в них вёрстку, сделанную в прошлой лабораторной работе.
5. Продемонстрировать работу web-приложения.

Клиентское программирование

Главная страница

```
import 'bootstrap/dist/css/bootstrap.min.css';
import '../../css/home.css';
import React from 'react';
import { Link } from "react-router-dom";

const Body = () => {return (<main id="main" className="py-4 bg-black text-white">
                <div className="container pb-3 mb-5 py-4 bg-black text-white">
                        <h1 className="card-title my-5 font-poppins">Выбор
станции </h1>
                        <div className="row row-cols-lg-2 row-cols-md-3 row-cols-
sm-2  font-poppins g-3">
```

```
                            <div className="col">
                                <div className="card text-white bg-black mb-3
text-start">
                                    <img src={require("../images/1.png")}
alt="hero" className="img-fluid card-img-top"/>
                                    <div className="card-body">
                                        <h3 className="card-title">Lofi-
radio</h3>
                                        <Link className="btn btn-primary"
to={'/lofi'}>LofiRadio</Link>
                                    </div>
                                </div>
                            </div>
                            <div className="col">
                                <div className="card text-white bg-black mb-3">
                                    <img src={require("../images/2.png")}
alt="hero" className="img-fluid card-img-top"/>
                                    <div className="card-body text-start">
                                        <h3 className="card-title">Phonk-
radio</h3>
                                        <Link className="btn btn-primary"
to={'/phonk'}>PhonkRadio</Link>
                                    </div>
                                </div>
                            </div>
                        </div>
                    </div>
                    </main>)
}
export default Body
```

Контентная страница

```
import 'bootstrap/dist/css/bootstrap.min.css';
import '../../css/lofi.css';
import 'bootstrap/js/dist/carousel'
import React from 'react';
import RadioData from '../../models/RadioData';
import { getRadiosUser } from '../../api/radioapi';
import RadioCard from './Radklcard';

const Body = () => {

    const [radios, setRadios] = React.useState<RadioData[]>([])
    React.useEffect(()=>{
      getRadiosUser(setRadios)
    }, [])
    console.log();

    return (<div className="radio">
```

```
      { radios.map(radio => <RadioCard {...radio} key={radio.id} />) }
     </div>
  )
}
export default Body
```

Страница авторизации

```
import {Field, Form, Formik} from 'formik';
import React from 'react';
import Widget from '../components/ContentWidget';
import 'bootstrap/dist/css/bootstrap.min.css';
import loginValidate from '../validations/loginValidate';
import { postLogin } from '../api/loginApi';
import { useNavigate } from 'react-router-dom';
import authContext from '../components/AuthContext';


export interface LoginValues {
    username: string,
    password: string
}

const Login = () => {
    const navigate = useNavigate();
    const auth_context = React.useContext(authContext);
    const onSubmit = (values: LoginValues) =>{
        postLogin(values, auth_context)
        navigate('/')
    }

    return (
        <Widget title='Log in'>
            <Formik
                initialValues={{
                username: '',
                password: '',
                }}
                onSubmit={onSubmit}
                validate={loginValidate}
            >{({ errors, touched }) => (
                <Form>
                    <label htmlFor="username">Username</label>
                    <Field className='form-control bg-dark text-white'
id="username" name="username" placeholder="username" />
                    {errors.username && touched.username &&
                        <div className='text-danger'>
                            {errors.username}
                        </div>}
                    <br/>
```

```
                    <label htmlFor="password">Password</label>
                    <Field type='password' className='form-control bg-dark text-
white' id="password" name="password" placeholder="password" />
                    {errors.password && touched.password &&
                        <div className='text-danger'>
                            {errors.password}
                        </div>}
                    <br/>
                    <div className='text-center'>
                        <button className='btn bg-dark text-white'
type="submit">Log in</button>
                    </div>
                    </Form>
            )}
            </Formik>
        </Widget>
    )
}

export default Login;
```

Серверное программирование

1. Развернуть базовое приложение.
2. Настроить конфигурацию работы приложения с docker.
3. Добавить модуль для работы с API.
4. Добавить несколько контроллеров со статическими данными.
5. Продемонстрировать работу API в Postman.

docker-compose.yaml

```
version: "3.7"

services:
  db:
    image: postgres
    volumes:
      - postgres_data:/var/lib/postgresql/data
    env_file:
      - .env.template


  backend:
    build:
      dockerfile: Dockerfile
      context: ../backend
```

```
    volumes:
      - ../backend:/var/www
    env_file:
      - .env.template
    depends_on:
      - db
    restart: always
    tty: true

  frontend:
    build:
      dockerfile: Dockerfile
      context: ../frontend
    ports:
      - "81:80"
    depends_on:
      - backend
volumes:
  postgres_data:
```

Dockerfile программы сервера

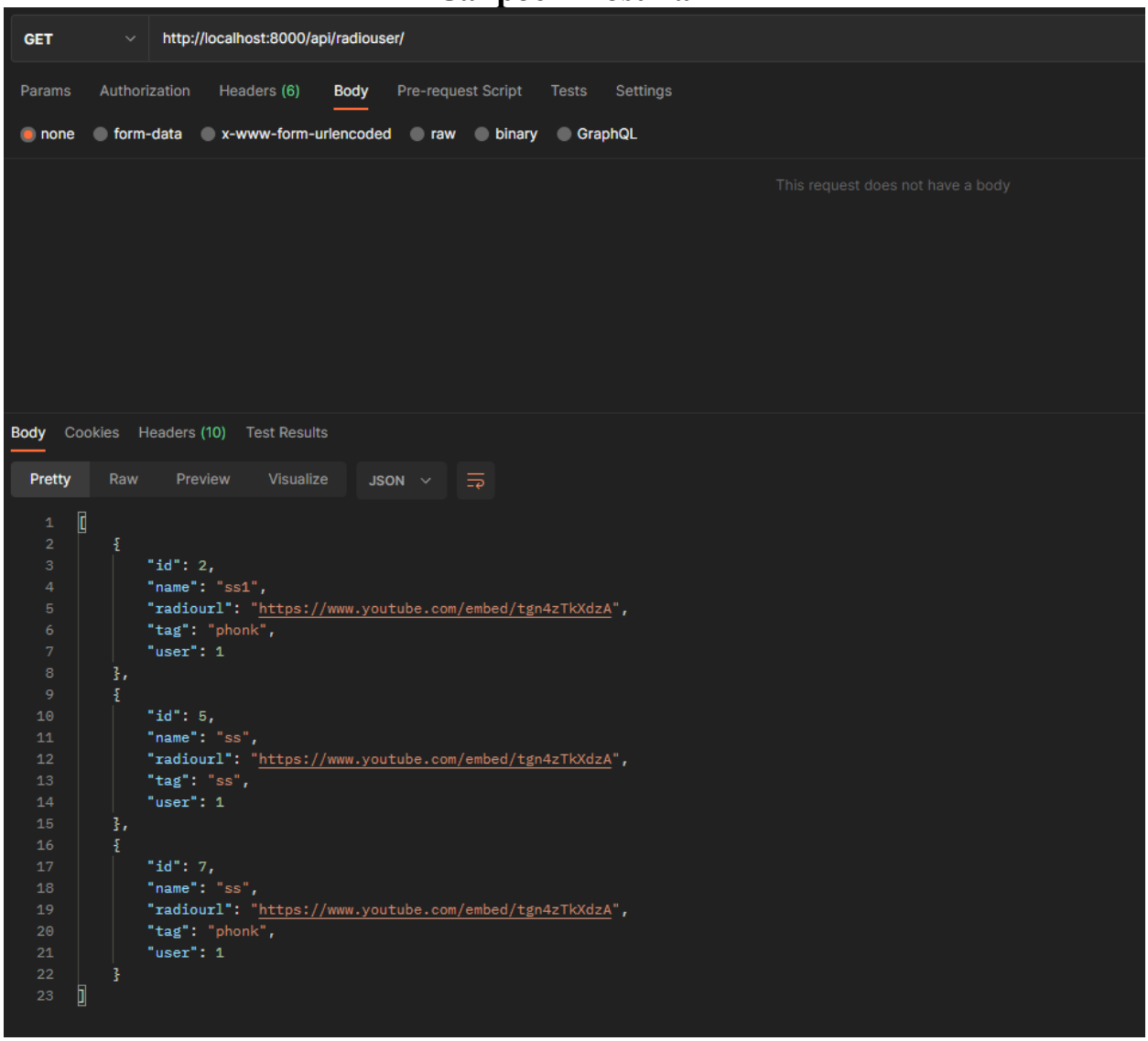
```
FROM python:latest

WORKDIR /var/www/
COPY ./requirements.txt .
RUN pip install -r requirements.txt
COPY ./ .
ENTRYPOINT ./run.sh
```

Dockerfile для nginx

```
FROM node:lts-alpine as deps
WORKDIR /var/www/frontend
COPY package.json .
RUN npm install
FROM node:lts-alpine as build
WORKDIR /var/www/frontend
COPY --from=deps /var/www/frontend/node_modules ./node_modules
COPY . .
RUN npm run build
FROM nginx:stable-alpine
COPY --from=build /var/www/frontend/build /usr/share/nginx/html
COPY --from=build /var/www/frontend/nginx.conf /etc/nginx/conf.d/default.conf
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Запрос в Postman



Разработка и проектирование базы данных

1. Выбрать подходящую СУБД.
2. Изучить методы взаимодействия web-приложения с базой данных.
3. Разработать структуру базы данных.
4. Разработать соответствующие модели в приложении.
5. В отчёт приложить схему базы данных, а также код одной из моделей (на своё усмотрение).

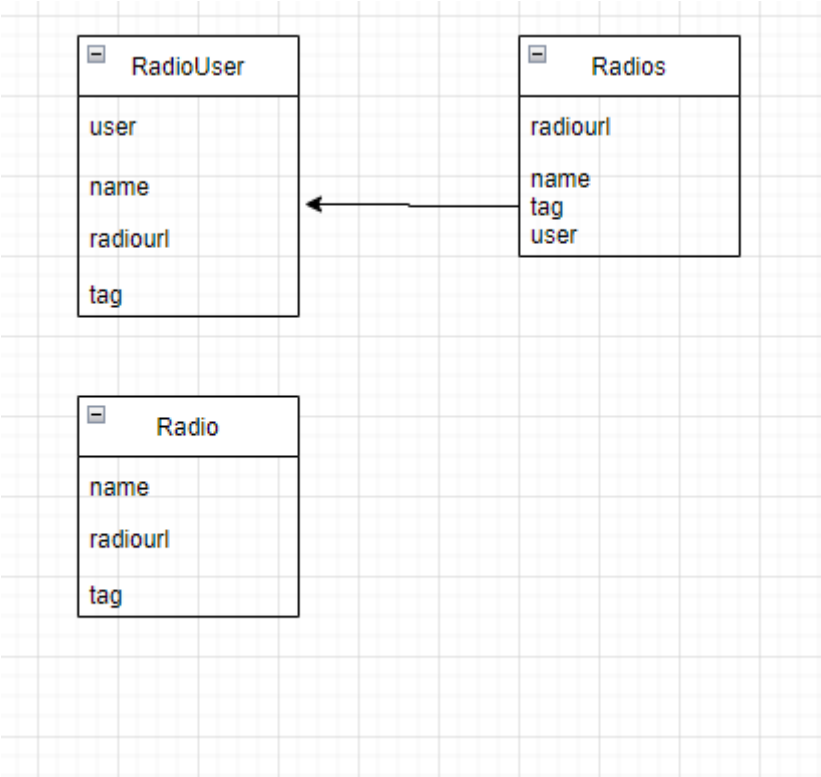
СУБД - PostrgeSQL

Схема базы данных

Код модели Game

```
class Radio(models.Model):
    name = models.CharField(max_length=255)
    radiourl = models.URLField(max_length=500)
    tag = models.CharField(max_length=100)

class RadioUser(models.Model):
    user = models.ForeignKey(django.contrib.auth.models.User, on_delete=models.CASCADE)
    name = models.CharField(max_length=255)
    radiourl = models.URLField(max_length=500)
    tag = models.CharField(max_length=200)
```



REST API

1. Изучить структуру формата представления данных JSON.
2. Изучить типы запросов к API: HEAD, GET, POST, PUT, DELETE.
3. Спроектировать и реализовать собственное REST API (Получение, создание, изменение и удаление каких-либо объектов).
4. В отчёт необходимо предоставить документацию к использованию методов. (Либо словесным описание, либо через Swagger)

Схема API (Swagger)

GET /radio/{id}/

Parameters

Name	Description
id * required integer *(path)*	A unique integer value identifying this radio. id

Responses

Code	Description
200	Example Value | Model

```
Radio ⌄ {
  id          integer
              title: ID
              readOnly: true
  name*       string
              title: Name
              maxLength: 255
              minLength: 1
  radiourl*   string($uri)
              title: Radiourl
              maxLength: 500
              minLength: 1
  tag*        string
              title: Tag
              maxLength: 100
              minLength: 1
}
```

PATCH /radio/{id}/

Parameters

Name	Description
data * required object *(body)*	Example Value | Model

```
Radio ⌄ {
  name*       string
              title: Name
              maxLength: 255
              minLength: 1
  radiourl*   string($uri)
              title: Radiourl
              maxLength: 500
              minLength: 1
  tag*        string
              title: Tag
              maxLength: 100
              minLength: 1
}
```

id * required integer *(path)*	A unique integer value identifying this radio. id

Responses

Code	Description
200	Example Value | Model

```
Radio ⌄ {
  id          integer
              title: ID
              readOnly: true
  name*       string
              title: Name
              maxLength: 255
              minLength: 1
  radiourl*   string($uri)
              title: Radiourl
              maxLength: 500
              minLength: 1
  tag*        string
              title: Tag
              maxLength: 100
              minLength: 1
}
```

POST /radio/

Parameters

Name	Description
data * required object *(body)*	Example Value | **Model** **Radio** { name* string *title: Name* *maxLength: 255* *minLength: 1* radiourl* string($uri) *title: Radiourl* *maxLength: 500* *minLength: 1* tag* string *title: Tag* *maxLength: 100* *minLength: 1* }

Responses

Code	Description
201	Example Value | **Model** **Radio** { id integer *title: ID* *readOnly: true* name* string *title: Name* *maxLength: 255* *minLength: 1* radiourl* string($uri) *title: Radiourl* *maxLength: 500* *minLength: 1* tag* string *title: Tag* *maxLength: 100* *minLength: 1* }

DELETE /radiouser/{id}/

Parameters

Name	Description
id * required integer (path)	A unique integer value identifying this radio user. id

Responses

Code	Description
204	

GET /radiouser/

Parameters

No parameters

Responses

Code	Description
200	Example Value | Model

```
[RadioUser {
  id*          integer
               title: ID
               readOnly: true
  name*        string
               title: Name
               maxLength: 255
               minLength: 1
  radiourl*    string($uri)
               title: Radiourl
               maxLength: 500
               minLength: 1
  tag*         string
               title: Tag
               maxLength: 200
               minLength: 1
  user*        integer
               title: User
}]
```

POST /radiouser/

Parameters

Name	Description
data * required object (body)	Example Value | Model

```
RadioUser {
  name*        string
               title: Name
               maxLength: 255
               minLength: 1
  radiourl*    string($uri)
               title: Radiourl
               maxLength: 500
               minLength: 1
  tag*         string
               title: Tag
               maxLength: 200
               minLength: 1
  user*        integer
               title: User
}
```

Responses

Code	Description
201	Example Value | Model

```
RadioUser {
  id           integer
               title: ID
               readOnly: true
  name*        string
               title: Name
               maxLength: 255
               minLength: 1
  radiourl*    string($uri)
               title: Radiourl
               maxLength: 500
               minLength: 1
  tag*         string
               title: Tag
               maxLength: 200
               minLength: 1
  user*        integer
               title: User
}
```

GET /radio/

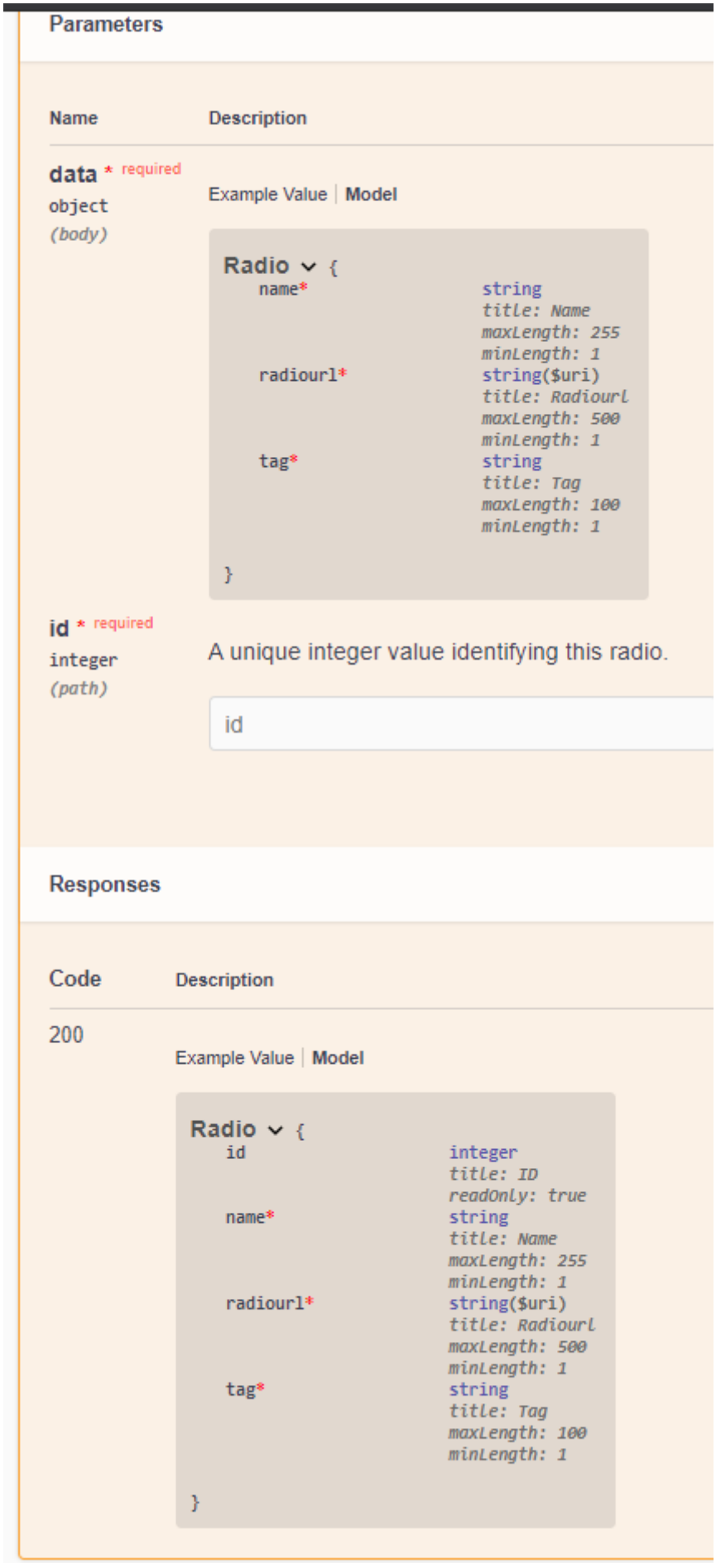
Parameters

No parameters

Responses

Code	Description
200	Example Value | **Model**

```
[Radio {
  id                integer
                    title: ID
                    readOnly: true
  name*             string
                    title: Name
                    maxLength: 255
                    minLength: 1
  radiourl*         string($uri)
                    title: Radiourl
                    maxLength: 500
                    minLength: 1
  tag*              string
                    title: Tag
                    maxLength: 100
                    minLength: 1
}]
```



Работа с HTTP запросами

1. Изучить возможности js для отправки http запросов.
2. Выбрать подходящую библиотеку для работы с запросами.
3. Реализовать взаимодействие фронтенда с REST API, спроектированном в прошлой лабораторной работе.
4. Продемонстрировать работу взаимодействия фронтенд приложения с REST API.

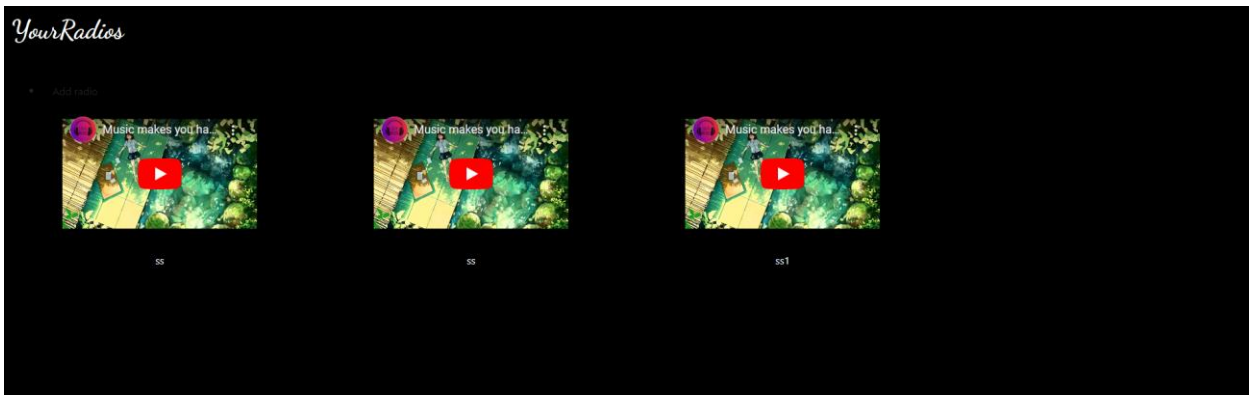
Код запроса

```
const radioPath = 'http://127.0.0.1:8000/api/radio/'
const userradioPath = 'http://127.0.0.1:8000/api/radiouser/'

export async function getRadios(resultHandler: (data: any)=>void, tag: string){
    Axios.get(radioPath,
        {params:{tag: tag},responseType: "json" }
    ).then
    (result => {
        const data: RadioData[] = (result as AxiosResponse<RadioData[]>).data;
        resultHandler(data);
    })
    .catch((error: AxiosError) => {
        alert(error.message);
    });
}

export async function getRadiosUser(resultHandler: (data: any)=>void){
    Axios.get(userradioPath,
        {params:{user:
parseJwt(localStorage.getItem('access')).user_id},responseType: "json" }
    ).then
    (result => {
        const data: RadioData[] = (result as AxiosResponse<RadioData[]>).data;
        resultHandler(data);
    })
    .catch((error: AxiosError) => {
        alert(error.message);
    });
}
```

Результат работы запроса


YourRadios
Add radio
Music makes you ha...
ss
Music makes you ha...
ss
Music makes you ha...
ss1