

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Web-программирование

Отчет

Лабораторная работа 3-4

Выполнил:

Дорофеева Арина

Группа к33401

Санкт-Петербург

2021 г.

## Задачи

- Реализовать модель базы данных средствами DjangoORM
  - Реализовать логику работу API средствами Django REST Framework (используя методы сериализации).
  - Подключить регистрацию / авторизацию по токенам / вывод информации о текущем пользователе средствами Djoser.
  - Реализовать документацию, описывающую работу всех используемых endpoint-ов из пункта 3 и 4 средствами Read the Docs или MkDocs.
- 
- Настроить для серверной части CORS.
  - Реализовать интерфейсы авторизации, регистрации и изменения учётных данных и настроить взаимодействие с серверной частью.
  - Реализовать клиентские интерфейсы и настроить взаимодействие с серверной частью.
  - Реализовать документацию, описывающую работу разработанных интерфейсов средствами MkDocs.

## Ход работы

- Реализовать модель базы данных средствами DjangoORM  
models.py

```
models.py x
1 from django.contrib.auth.models import AbstractUser
2 from django.db import models
3
4
5 class User(AbstractUser):
6     phone = models.CharField("phone", max_length=15, blank=True, null=True)
7
8     REQUIRED_FIELDS = ['first_name', 'last_name', 'phone', 'id']
9
10    def __str__(self):
11        return self.username
12
13
14    class VideoOfUser(models.Model):
15        owner = models.ForeignKey(User, on_delete=models.CASCADE)
16        tags = models.TextField()
17        link = models.TextField()
18
19    def __str__(self):
20        return "user: {}, name: {}".format(self.owner, self.owner)
21
22
23    class LinkToVideo(models.Model):
24        tags = models.CharField(max_length=200)
25        link = models.TextField()
26
27    def __str__(self):
28        return "tags: {}, link: {}".format(self.tags, self.link)
29
```

- Реализовать логику работу API средствами Django REST Framework (используя методы сериализации).

api.py

```
api.py x serializers.py x
1 from rest_framework import viewsets, permissions
2 from .serializers import *
3
4
5 class VideoViewSet(viewsets.ModelViewSet):
6
7     def get_queryset(self):
8         queryset = VideoOfUser.objects.all()
9
10        owner = self.request.query_params.get('owner', None)
11        if owner is not None:
12            queryset = queryset.filter(owner=owner)
13        return queryset
14
15        permissions_classes = [
16            permissions.AllowAny
17        ]
18        serializer_class = VideoSerializer
19
20
21 class LinkToVideoViewSet(viewsets.ModelViewSet):
22     def get_queryset(self):
23
24         queryset = LinkToVideo.objects.all()
25         return queryset
26
27     permissions_classes = [
28         permissions.AllowAny
29     ]
30     serializer_class = LinkToVideoSerializer
31
```

serializers.py

```
urls.py × models.py × serializers.py ×
1  from rest_framework import serializers
2  from .models import *
3
4
5  class VideoSerializer(serializers.ModelSerializer):
6      class Meta:
7          model = VideoOfUser
8          fields = "__all__"
9
10
11 class LinkToVideoSerializer(serializers.ModelSerializer):
12     class Meta:
13         model = LinkToVideo
14         fields = "__all__"
15
```

- Подключить регистрацию / авторизацию по токенам / вывод информации о текущем пользователе средствами Djoser.

urls.py

```

urls.py x
1 """axolotls_back URL Configuration
2
3 The `urlpatterns` list routes URLs to views. For more information please see:
4     https://docs.djangoproject.com/en/3.1/topics/http/urls/
5 Examples:
6 Function views
7     1. Add an import:  from my_app import views
8     2. Add a URL to urlpatterns:  path('', views.home, name='home')
9 Class-based views
10    1. Add an import:  from other_app.views import Home
11    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
12 Including another URLconf
13    1. Import the include() function: from django.urls import include, path
14    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
15 """
16 import ...
17
18
19 urlpatterns = [
20     path('admin/', admin.site.urls),
21     path('', include('axolotls_app.urls')),
22
23
24     path('auth/', include('djoser.urls')),
25     re_path(r'^auth/', include('djoser.urls.auth_token')),
26 ]

```

```

7 class User(AbstractUser):
8     phone = models.CharField("Телефон", max_length=15, blank=True, null=True)
9
10     REQUIRED_FIELDS = ['first_name', 'last_name', 'phone', 'id']
11
12     def __str__(self):
13         return self.username

```

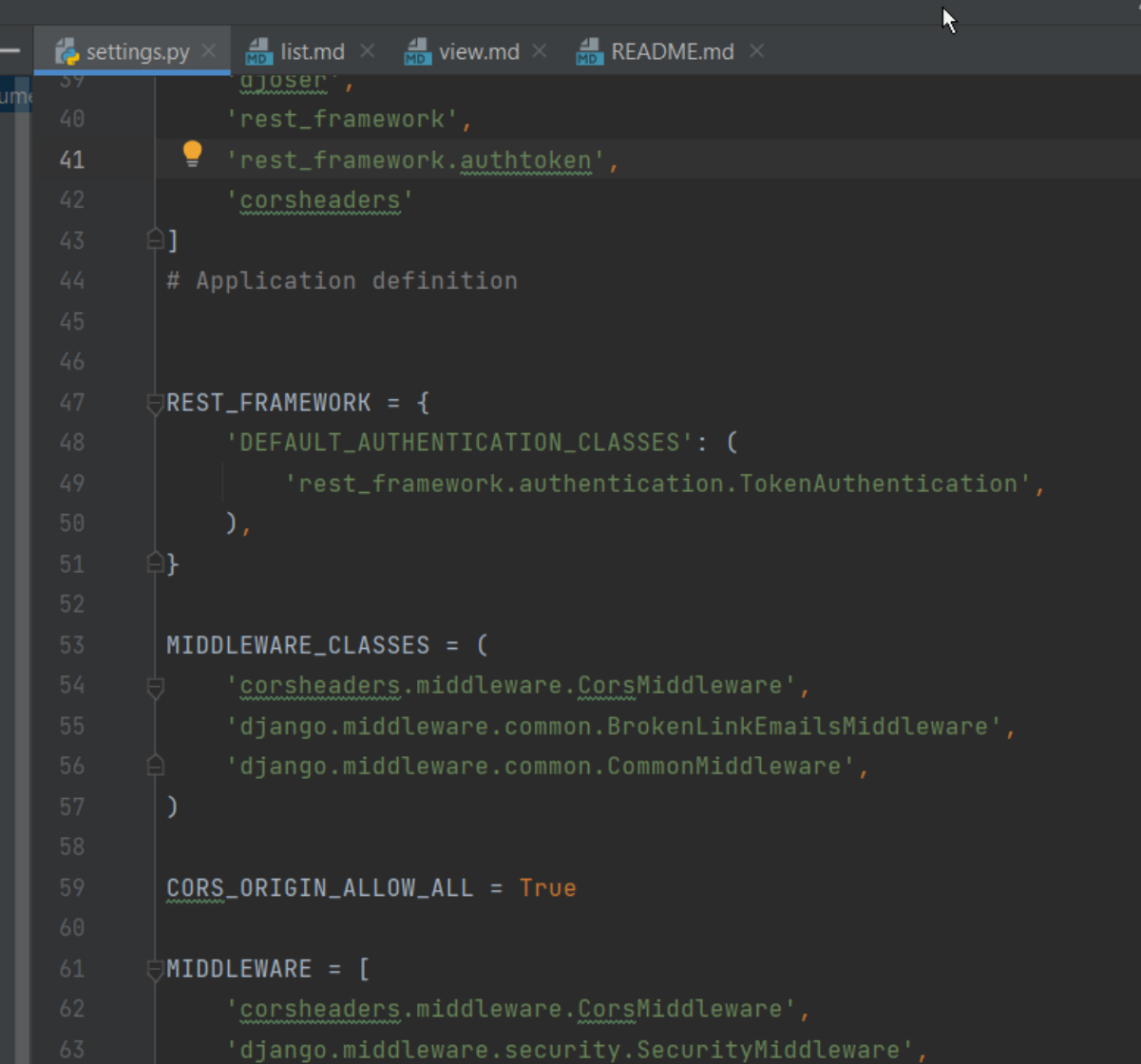
## settings.py

```

urls.py x  models.py x  serializers.py x  settings.py x
31 INSTALLED_APPS = [
32     'django.contrib.admin',
33     'django.contrib.auth',
34     'django.contrib.contenttypes',
35     'django.contrib.sessions',
36     'django.contrib.messages',
37     'django.contrib.staticfiles',
38     'axolotls_app',
39     'djoser',
40     'rest_framework',
41     'rest_framework.auth_token',
42     'corsheaders'
43 ]
44 # Application definition
45
46
47 REST_FRAMEWORK = {
48     'DEFAULT_AUTHENTICATION_CLASSES': (
49
50
51
52
53
54
55
56
57
58
59
60
61 MIDDLEWARE = [
62     'corsheaders.middleware.CorsMiddleware',
63     'django.middleware.security.SecurityMiddleware',
64     'django.contrib.sessions.middleware.SessionMiddleware',
65     'django.middleware.common.CommonMiddleware',
66     'django.middleware.csrf.CsrfViewMiddleware',
67     'django.contrib.auth.middleware.AuthenticationMiddleware',
68     'django.contrib.messages.middleware.MessageMiddleware',
69     'django.middleware.clickjacking.XFrameOptionsMiddleware',
70 ]
71 AUTH_USER_MODEL = "axolotls_app.User"
72 ROOT_URLCONF = "axolotls_back.urls"
73
74 DJOSER = {
75     "USER_ID_FIELD": "username"
76 }

```

- Настроить для серверной части CORS.



The image shows a code editor with a dark theme. The top bar displays four open files: 'settings.py', 'list.md', 'view.md', and 'README.md'. The 'settings.py' file is active and shows Python code for Django settings. Line 39 has a lightbulb icon. Lines 40-42 show a list of settings: 'djoser', 'rest\_framework', 'rest\_framework.authtoken', and 'corsheaders'. Line 43 is a closing bracket. Line 44 is a comment '# Application definition'. Line 47 starts a dictionary 'REST\_FRAMEWORK = {' with a collapse icon. Line 48 is 'DEFAULT\_AUTHENTICATION\_CLASSES: ('. Line 49 is 'rest\_framework.authentication.TokenAuthentication,'. Line 50 is a closing parenthesis. Line 51 is a closing bracket for the dictionary. Line 53 starts 'MIDDLEWARE\_CLASSES = ('. Line 54 is 'corsheaders.middleware.CorsMiddleware,'. Line 55 is 'django.middleware.common.BrokenLinkEmailsMiddleware,'. Line 56 is 'django.middleware.common.CommonMiddleware,'. Line 57 is a closing parenthesis. Line 59 is 'CORS\_ORIGIN\_ALLOW\_ALL = True'. Line 61 starts 'MIDDLEWARE = [' with a collapse icon. Line 62 is 'corsheaders.middleware.CorsMiddleware,'. Line 63 is 'django.middleware.security.SecurityMiddleware,'.

```
39  'djoser',
40  'rest_framework',
41  'rest_framework.authtoken',
42  'corsheaders'
43 ]
44 # Application definition
45
46
47 REST_FRAMEWORK = {
48     'DEFAULT_AUTHENTICATION_CLASSES': (
49         'rest_framework.authentication.TokenAuthentication',
50     ),
51 }
52
53 MIDDLEWARE_CLASSES = (
54     'corsheaders.middleware.CorsMiddleware',
55     'django.middleware.common.BrokenLinkEmailsMiddleware',
56     'django.middleware.common.CommonMiddleware',
57 )
58
59 CORS_ORIGIN_ALLOW_ALL = True
60
61 MIDDLEWARE = [
62     'corsheaders.middleware.CorsMiddleware',
63     'django.middleware.security.SecurityMiddleware',
```

```

10 https://docs.djangoproject.com/en/3.1/ref/settings/
11 """
12
13 from pathlib import Path
14
15 # Build paths inside the project like this: BASE_DIR / 'subdir'.
16 BASE_DIR = Path(__file__).resolve().parent.parent
17
18
19 # Quick-start development settings - unsuitable for production
20 # See https://docs.djangoproject.com/en/3.1/howto/deployment/checklist/
21
22 # SECURITY WARNING: keep the secret key used in production secret!
23 SECRET_KEY = '4d%0s+k8y1-&xc%z@wex3gla$vu1_37^q36tpd66!^th+turw^'
24
25 # SECURITY WARNING: don't run with debug turned on in production!
26 DEBUG = True
27
28 ALLOWED_HOSTS = ['*']
29
30

```

- Реализовать интерфейсы авторизации, регистрации и изменения учётных данных и настроить взаимодействие с серверной частью.
- Реализовать клиентские интерфейсы и настроить взаимодействие с серверной частью.

```

9  const tokenConfig = (token) =>{
10      const config = {
11          headers: {
12              "Content-Type": "application/json"
13          }
14      }
15      if (token) {
16          config.headers["Authorization"] = `Token ${token}`
17      }
18      return config
19  }

```



```

21 export default createStore( options: {
22   state: {
23     videos: [],
24     videosTmp: [],
25     token: localStorage.getItem( key: 'token'),
26     userData: null,
27     videosOfUser: [],
28     registrationError: null,
29     axolotls: [],
30     loginError: null,
31     repeatPasswordError: "",
32   },
33   mutations: {

```

```

67   login (state, authData) {
68
69     const config = {
70       headers: {
71         "Content-Type": 'application/json'
72       }
73     }
74     const {username, password} = authData;
75     const body = JSON.stringify( value: {username, password});
76
77
78     axios.post( url: URL + '/auth/token/login/', body, config).then(res => {
79       localStorage.setItem("token", res.data.auth_token);
80       console.log(localStorage.getItem( key: "token"));
81
82       state.token = res.data.auth_token
83       router.push({
84         path: '/cabinet',
85       });
86     }).catch(err => {
87       state.loginError = err.response.data
88     })
89   },
90   setRepeatPasswordError(state, msg) {
91     state.repeatPasswordError = msg;
92   },

```

```

93     logout (state) {
94         const token = state.token || localStorage.getItem(key: 'token');
95         const config = tokenConfig(token)
96
97         localStorage.removeItem(key: "token")
98         axios.post(url: URL + '/auth/token/logout/', data: null, config).then(res => {
99             router.push({
100                 path: '/',
101             });
102         }).catch(err => {
103             console.log(err)
104         })
105         state.userData = null
106         state.token = ""
107         state.videos = []
108         state.videosTmp = []
109         state.videosOfUser = []
110         state.registrationError = null
111         state.loginError = null
112         state.repeatPasswordError = ""
113         localStorage.removeItem(key: 'token');
114     },
115     loadUser(state, res) {
116         console.log(res)
117         state.userData = res.data
118     },

```

```

119     changeUser(state, res) {
120
121         const token = localStorage.getItem(key: 'token');
122         const config = tokenConfig(token)
123
124         console.log(res)
125         console.log(token)
126         axios.put(url: URL + '/auth/users/me/', res, config).then(res => {
127             console.log(res.data)
128         }).catch(err => {
129
130         })
131     },

```

```

registerUser(state, res) {

  const config = {
    headers: {
      "Content-Type": 'application/json'
    }
  }

  axios.post(URL + '/auth/users/', res, config).then(res => {
    router.push({
      path: "/sign-in"
    })
  }).catch(err => {
    state.registrationError = err.response.data
  })
}
}

```

```

132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161

addVideo(state, res){
  const token = state.token || localStorage.getItem(key: 'token');
  const config = tokenConfig(token)
  const data = {
    ...res,
    owner: state.userData.id
  }
  axios.post(URL + '/api/videos-of-user/', data, config).then(res => {
    console.log(res)
  }).catch(err => {
    console.log(err)
  })
},

removeVideo(state, data){
  const token = state.token || localStorage.getItem(key: 'token');
  const config = tokenConfig(token)
  axios.delete(URL + '/api/videos-of-user/${data.id}', config).then(res => {
    state.videosOfUser = state.videosOfUser.filter(video => video.id !== data.id)
  }).catch(err => {
    console.log(err)
  })
},

getVideosOfUser(state, res) {
  const token = localStorage.getItem(key: 'token');
  const config = tokenConfig(token)
  axios.get(URL + '/api/videos-of-user/?owner=${state.userData.id}', config).then(res => {
    state.videosOfUser = res.data;
  }).catch(err => {
    console.log(err)
  })
}

```

```
182 actions: {
183   registerUser(ctx) {
184     ctx.registerUser()
185   },
186   logoutUser(ctx) {
187     ctx.logout();
188   },
189   loadVideos(ctx) {
190     axios.get( url: "http://127.0.0.1:8000/api/link-videos/")
191       .then(data => {
192         ctx.commit("setVideos", data.data)
193       }).catch((error) => {
194         console.log("ERROR: " + error);
195       });
196   },
197   getUserData (ctx) {
198     const token =localStorage.getItem( key: 'token');
199     const config = tokenConfig(token)
200
201     axios.get( url: URL + '/auth/users/me/', config).then(res => {
202       console.log(res)
203       console.log(res.data)
204       ctx.commit("loadUser", {data: res.data})
205
206     }).catch(err => {
207       router.push({
208         path: '/sign-in',
209       });
210     })
211   },
212 }
```

```
220     getters: {
221         allVideos(state) {
222             return state.videosTmp
223         },
224         userData(state) {
225             return state.userData
226         },
227         getToken(state) {
228             return state.token
229         },
230         getAllVideosOfUser(state){
231             return state.videosOfUser;
232         },
233         getRegistrationError(state) {
234             return state.registrationError;
235         },
236         getLoginError(state) {
237             return state.loginError;
238         },
239         getRepeatPasswordError(state) {
240             return state.repeatPasswordError;
241         },
242         getAxolotls(state) {
243             return state.axolotls;
244         }
245     }
```

- [Документация](#)

Axolotl

Search

Axolotl

Home

videos-of-user

link-of-videos

view

list

Create video

URL: /Video/create/

Methods: PUT, POST, DELETE

Data constraints: {}

Generics type: CreateAPIView

Success Responses

Code: 200 OK

Content: {}

```
class LinkToVideo(models.Model):
    tags = models.CharField(max_length=200)
    link = models.TextField()

    def __str__(self):
        return "tags: {}, link: {}".format(self.tags, self.link)

class LinkToVideoSerializer(serializers.ModelSerializer):
    class Meta:
        model = LinkToVideo
        fields = "__all__"
```

Table of contents

Success Responses

Axolotl

Search

Axolotl

Home

videos-of-user

view

list

link-of-videos

List of user videos

URL: /video-of-user/list/

Method: GET

Data constraints: {}

Success Responses

Code: 200 OK

Content: {}

```
[
  {
    "id": 3,
    "owner": 1,
    "tags": "colored axolotls yellow black pink",
    "link": "https://www.youtube.com/embed/uhdTmsKvtwU"
  }
]
```

Table of contents

Success Responses

Previous view

Next view

Axolotl

Search

Axolotl

Home

videos-of-user

view

list

link-of-videos

Add video for user

URL: /video-of-user/create/

Methods: PUT, POST, DELETE

Data constraints: {}

Generics type: CreateAPIView

Code: 200 OK

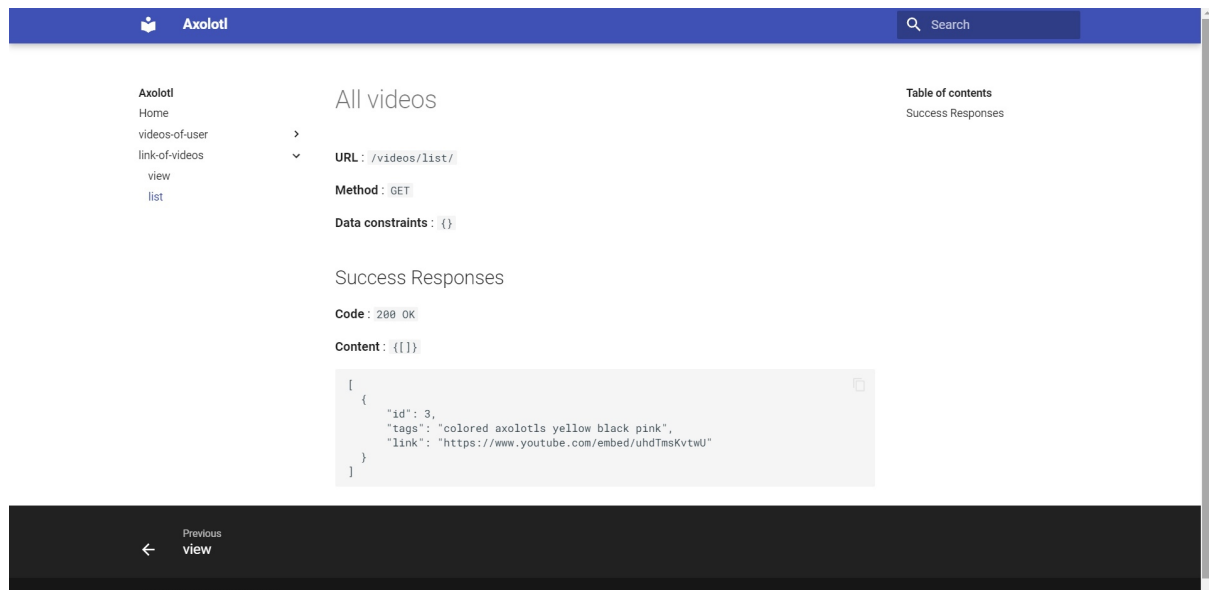
Content: {}

```
class VideoOfUser(models.Model):
    owner = models.ForeignKey(User, on_delete=models.CASCADE)
    tags = models.TextField()
    link = models.TextField()

    def __str__(self):
        return "пользователь: {}, Название: {}".format(self.owner, self.owner)

class VideoOfUserSerializer(serializers.ModelSerializer):
    class Meta:
        model = VideoOfUser
        fields = "__all__"

class VideoViewSet(viewsets.ModelViewSet):
```



## Вывод

Реализована серверная часть на django rest.

Реализована клиентская часть средствами Vue.JS.