МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ имени академика С.П.КОРОЛЕВА»

ИНСТИТУТ ИНФОРМАТИКИ И КИБЕРНЕТИКИ

КАФЕДРА ТЕХНИЧЕСКОЙ КИБЕРНЕТИКИ

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

Тема: «Авторизация»

Дисциплина: «Технологии сетевого программирования»

Выполнил: Якухин И.В., гр.6303

Проверил: Кашапов А.И.

Самара 2025

СОДЕРЖАНИЕ

[Введение 3](#_Toc199548329)

[1 Аутентификация и управление пользователем 4](#_Toc199548330)

[2 Примеры запросов 5](#_Toc199548331)

[Приложение А 7](#_Toc199548332)

[Ссылка на GitHub с проектом 7](#_Toc199548333)

[Приложение Б 8](#_Toc199548334)

[Фрагменты файла views.py” отвечающие за аутентификацию и управление пользователем 8](#_Toc199548335)

[Код файла “middleware.py” 9](#_Toc199548336)

Введение

Cardiogram – приложение с увлекательной игрой для запоминания английских слов с помощью карточек. Стек технологий приложения: Python, Django, React, PostgreSQL. Все компоненты конечной системы контенеризованы с помощью Docker для обеспечения изоляции окружения, удобства развертывания и масштабирования.

Учебный проект разработан с целью демонстрации навыков работы с базой данных и инструментами серверной и клиентской разработки, проектирования структуры приложений и API, контейнеризации.

Ссылка на GitHub страницу со всеми файлами проекта можно найти в приложении А.

1. Аутентификация и управление пользователем

API методы, отвечающие за аутентификацию и управление пользователем, включая методы, URL, параметры, форматы запросов и ответов, описана в таблице 1.

Таблица 1 – Описание API

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Действие | Метод | URL | Параметры | Формат запроса | Формат ответа |
| Регистрация | POST | auth/register | username,  fisrt\_name,  last\_name,  email,  password | {“username”: “Werd”,  “fisrt\_name”: “Иван”,  “last\_name”: “Якухин”,  “email": “werb32@mail.ru”,  “password”: “1234”  } | {“message”: “registration successful”} |
| Авторизация | POST | auth/login | username,  password | {“username”: “Werd”,  “password”: “1234”  } | {“message”: “login successful”} |
| Смена пароля | POST | auth/change-password | old\_password,  new-password | {  “old\_password”: “1234”,  “new-password”: “1111”  } | {“message”: “password changed successful”} |
| Выход из системы | POST | auth/logout |  |  | {“message”: “logout successful”} |

Согласно таблице, были реализованы методы регистрации пользователя, авторизации пользователя, изменения пароля и выхода из системы. Фрагмент кода, отвечающие за реализацию этих методов, приведен в приложении Б. Для проверки валидности JWT токенов и обновления access токена был написан middleware класс RefreshAccessTokenMiddleware, код которого также представлен в приложении Б. JWT токены как в API запросах, так и в middleware записываются в HTTPOnly Cookie файлы. Для ограничения доступа к определенным эндпоинтам были добавлено разрешение isAuthenticated, позволяющее доступ только аутентифицированным пользователям.

1. Примеры запросов

В данном разделе приведены скриншоты запросов к API методам, связанным с аутентификацией и управлением пользователем, и ответов на эти запросы в программе Postman. На рисунках 1 и 2 представлены запросы на вход и выход из системы соответственно.

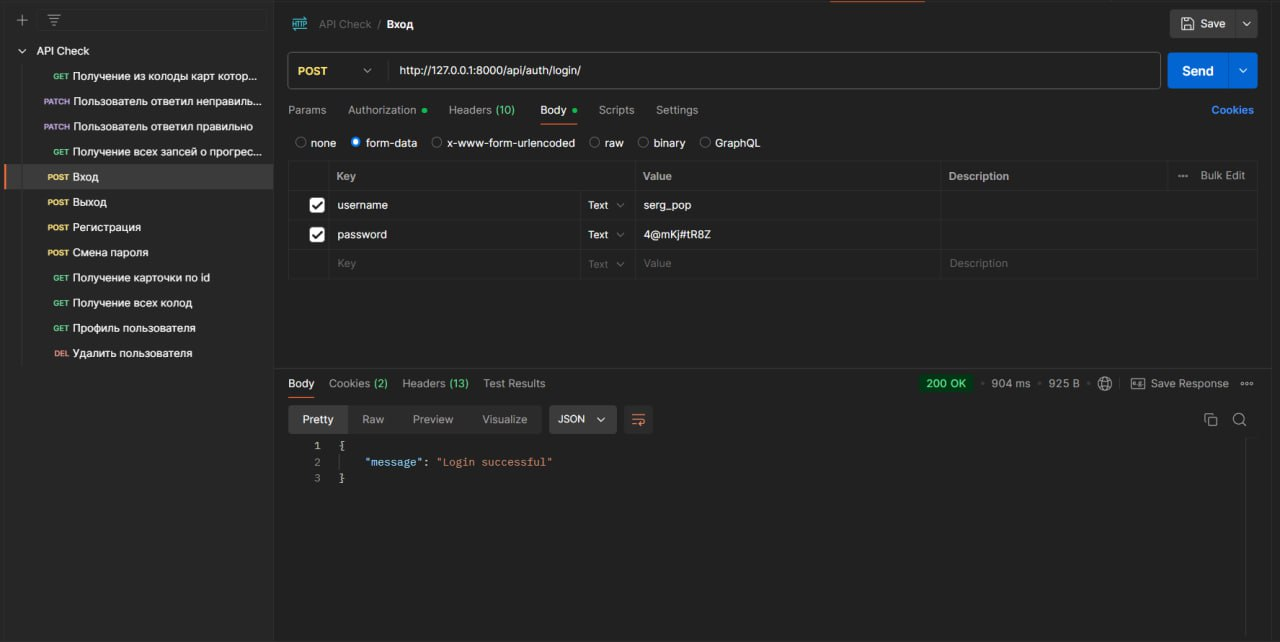


Рисунок 1 – Скриншот запроса на вход в систему

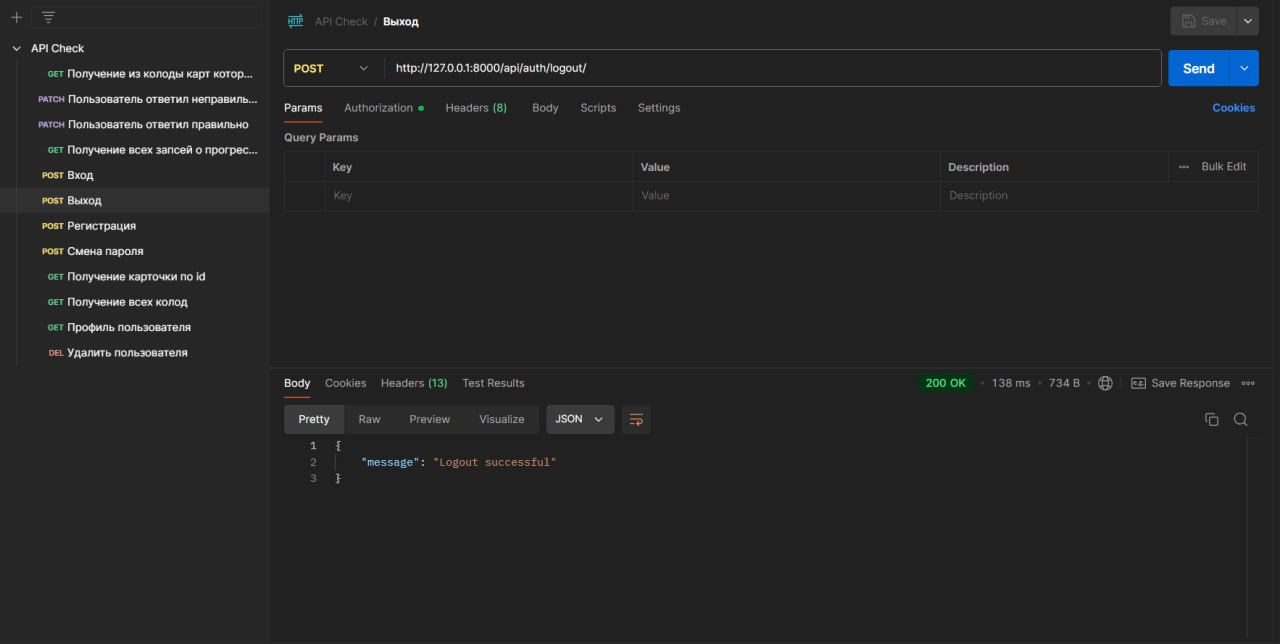


Рисунок 2 – Скриншот запроса на выход из системы

На рисунке 3 представлен не авторизированный запрос на смену пароля. Так как это защищенный эндпоинт на скриншоте можно видеть, что ответ возвращен со статусом 401 Unauthorized.

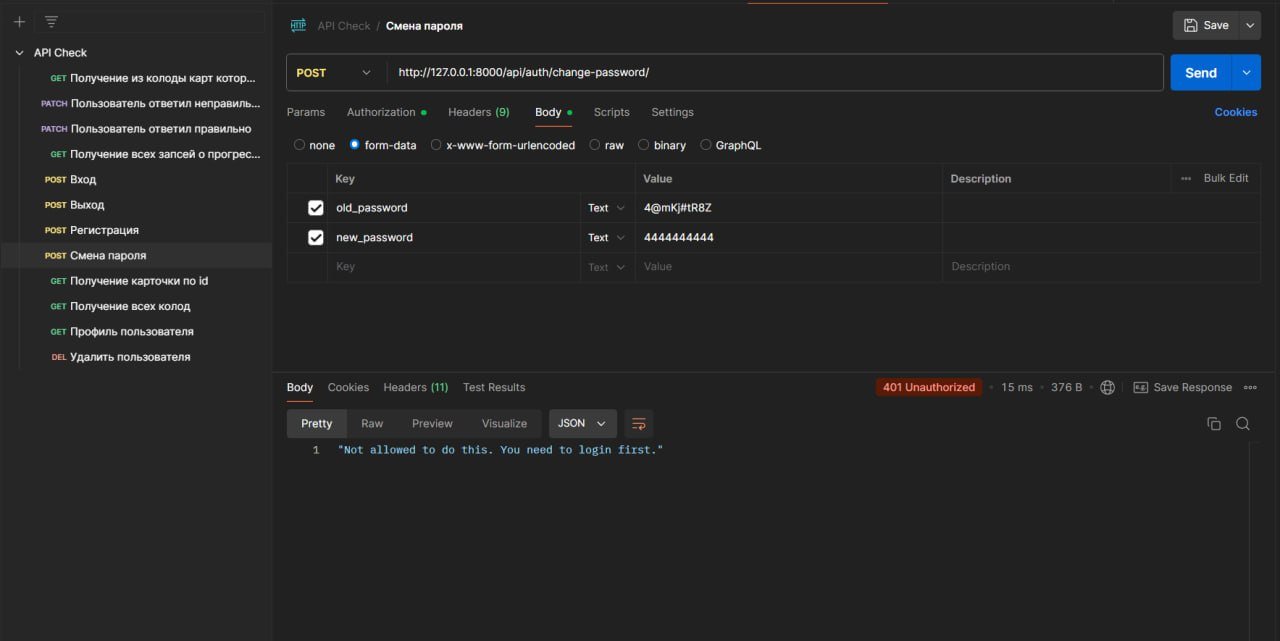


Рисунок 3 – Скриншот неавторизованного запроса на смену пароля

На рисунке 4 представлен тот же запрос, но на этот раз авторизованный, видно, что ответ пришел с кодом 200, сигнализирующим, что все в порядке.

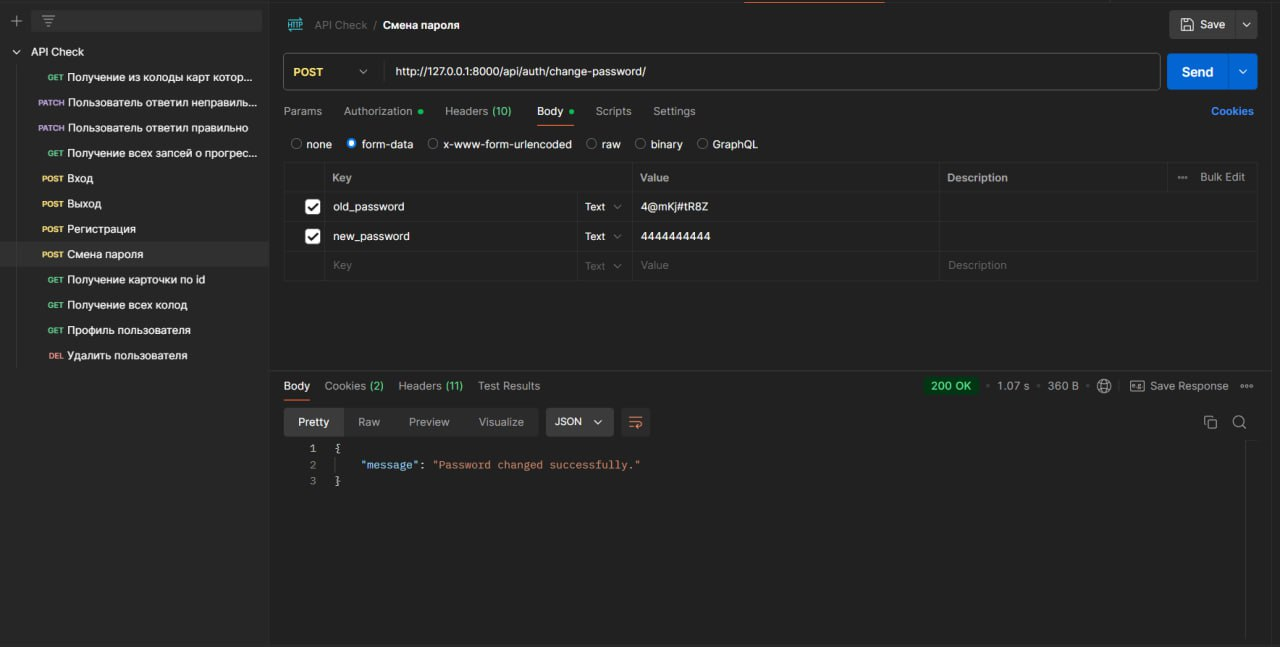


Рисунок 4 – авторизованный запрос на смену пароля

Приложение А

Ссылка на GitHub с проектом

https://github.com/BruhMano/CardioGram.git

Приложение Б

Фрагменты файла views.py” отвечающие за аутентификацию и управление пользователем

class UserAuth(views.APIView):

def post(self, request, action, \*args, \*\*kwargs):

if action == "change-password":

if not request.user.is\_authenticated:

return Response("Not allowed to do this. You need to login first.", 401)

user = request.user

serializer = ChangePasswordSerializer(data=request.data, context={'user': user})

if serializer.is\_valid():

new\_password = serializer.validated\_data['new\_password']

user.set\_password(new\_password)

user.save()

return Response({"message": "Password changed successfully."}, status=200)

return Response(serializer.errors, status=400)

elif action == "logout":

if not request.user.is\_authenticated:

return Response("Not allowed to do this. You need to login first.", 401)

refresh\_token = request.COOKIES.get('refresh\_token')

if refresh\_token:

token = RefreshToken(refresh\_token)

token.blacklist()

response = Response({'message': 'Logout successful'})

response.delete\_cookie('refresh\_token')

response.delete\_cookie('access\_token')

return response

elif action == "login":

username = request.data.get('username')

password = request.data.get('password')

user = authenticate(username=username, password=password)

if user is None:

return Response({'error': 'Invalid credentials. Try again!'}, status=401)

refresh = RefreshToken.for\_user(user)

response = Response()

response.set\_cookie(

key='refresh\_token',

value=str(refresh),

secure=False,

httponly=True,

samesite='Lax'

)

response.set\_cookie(

key='access\_token',

value=str(refresh.access\_token),

secure=False,

httponly=True,

samesite='Lax'

)

response.data = {

'message': 'Login successful'

}

return response

elif action == "register":

if get\_user\_model().objects.filter(username = request.data.get('username')).exists():

return Response("User already exists!", 400)

user = get\_user\_model().objects.create(

username = request.data.get('username'),

email = request.data.get('email'),

first\_name = request.data.get('first\_name'),

last\_name = request.data.get('last\_name')

)

user.set\_password(request.data.get('password'))

user.save()

login\_res = self.post(request, action='login')

return login\_res

Код файла “middleware.py”

from rest\_framework\_simplejwt.exceptions import InvalidToken, TokenError

from rest\_framework\_simplejwt.tokens import RefreshToken

from rest\_framework\_simplejwt.authentication import JWTAuthentication

class RefreshAccessTokenMiddleware:

def \_\_init\_\_(self, get\_response):

self.get\_response = get\_response

def \_\_call\_\_(self, request):

access\_token = request.COOKIES.get('access\_token')

if access\_token:

try:

request.META['HTTP\_AUTHORIZATION'] = f'Bearer {access\_token}'

jwt\_auth = JWTAuthentication()

jwt\_auth.get\_validated\_token(access\_token)

except (InvalidToken, TokenError):

refresh\_token = request.COOKIES.get('refresh\_token')

if refresh\_token:

try:

new\_access\_token = RefreshToken(refresh\_token).access\_token

request.META['HTTP\_AUTHORIZATION'] = f'Bearer {new\_access\_token}'

response = self.get\_response(request)

response.set\_cookie(

key='access\_token',

value=str(new\_access\_token),

httponly=True,

samesite='Lax'

)

response.status\_code = 200

return response

except (InvalidToken, TokenError):

response.data = {'error': 'Invalid refresh token. You need to login again!'}

response.status\_code = 401

response.delete\_cookie('refresh\_token')

response.delete\_cookie('access\_token')

return response

response = self.get\_response(request)

return response