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 №2

Тема: «Разработка API»

Дисциплина: «Технологии сетевого программирования»

Выполнил: Якухин И.В., гр.6303

Проверил: Кашапов А.И.

Самара 2025

СОДЕРЖАНИЕ

[Введение 3](#_Toc199546386)

[1 Структура API 4](#_Toc199546387)

[2 Обработка запросов и маршрутизация 7](#_Toc199546388)

[3 Примеры запросов 8](#_Toc199546389)

[Приложение А 10](#_Toc199546390)

[Ссылка на GitHub с проектом 10](#_Toc199546391)

[Приложение Б 11](#_Toc199546392)

[Код файла “serializers.py” 11](#_Toc199546393)

[Код файла “views.py” 12](#_Toc199546394)

[Код файла “urls.py” 16](#_Toc199546395)

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. Структура API

Структура API, включая методы, URL, параметры, форматы запросов и ответов, описана в таблице 1.

Таблица 1 – Описание API

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Действие | Метод | URL | Параметры | Формат запроса | Формат ответа |
| Получение информации о карточке | GET | card/<id> | Id – номер карточки в таблице базы данных |  | {  “id”: 23,  “front\_text”: “forest”,  “back\_text”: “Лес”,  “example\_usage”: “Forest is beautiful! (Лес прекрасен!)”  } |
| Получение информации о колодах | GET | deck/ |  |  | [{  “id”: 4,  “name”: “Nature”,  “description”: “Колода содержит слова, связанные с темой природы.”,  “cover”: “http://127.0.0.1/deck/forest.png”  },  {  …  }] |
| Получение из колоды всех карточек, которых нет у пользователя | GET | deck/<id> | Id – номер колоды в таблице базы данных |  | [{  “id”: 23,  “front\_text”: “forest”,  “back\_text”: “Лес”,  “example\_usage”: “Forest is beautiful! (Лес прекрасен!)”  },  {  “id”: 23,  “front\_text”: “nature”,  “back\_text”: “природа”,  “example\_usage”: “Nature is beautiful! (Природа прекрасна!)”  },…] |
| 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”} |
| Изменение личных данных | POST | auth/edit | username,  fisrt\_name,  last\_name,  email | {“username” : “Werd”,  “fisrt\_name”: “Иван”,  “last\_name”: “Якухин”,  “email": “werb32@mail.ru”  } | {“message”: “Profile editing completed successfully!”} |
| Получение данных о прогрессе пользователя | GET | progress/ |  |  | [{  “user”: 23,  “card”: 34,  “attempts”: 23,  “successful\_attempts”: 10  },  {  “user”: 23,  “card”: 32,  “attempts”: 21,  “successful\_attempts”: 5  },…] |
| Добавление карточки пользователю | POST | progress/ | card | {“card”: 23} | {“message”: “Card added to your deck successfully!  ”} |
| Получение профиля пользователя | GET | profile/ |  |  | {“username” : “Werd”,  “fisrt\_name”: “Иван”,  “last\_name”: “Якухин”,  “email": “werb32@mail.ru”  } |
| Пользователь правильно выбрал карточку | PATCH | profile/right | card | {“card”: 23} | {“message”: “the card information has been updated”} |
| Пользователь неправильно выбрал карточку | PATCH | profile/wrong | card | {“card”: 23} | {“message”: “the card information has been updated”} |
| Удаление аккаунта пользователя | DELETE | delete/ |  |  | {'message': 'Account deleted successfully!'} |

1. Обработка запросов и маршрутизация

Для реализации API методов был создан отдельный модуль api. С использованием классов из библиотеки Django-rest-framework были созданы сериализаторы для корректной обработки данных, view-классы, регламентирующие формат запросов и ответов и порядок обработки данных, и пути маршрутизации запросов. Код из файлов “serializers.py”, “views.py”, “urls.py” приведены в приложении Б.

1. Примеры запросов

В данном разделе приведены скриншоты запросов к API и ответов на эти запросы в программе Postman. На рисунках 1 и 2 представлены неавторизованный и авторизованный запросы к ресурсу, требующему аутентификацию пользователя, соответственно.

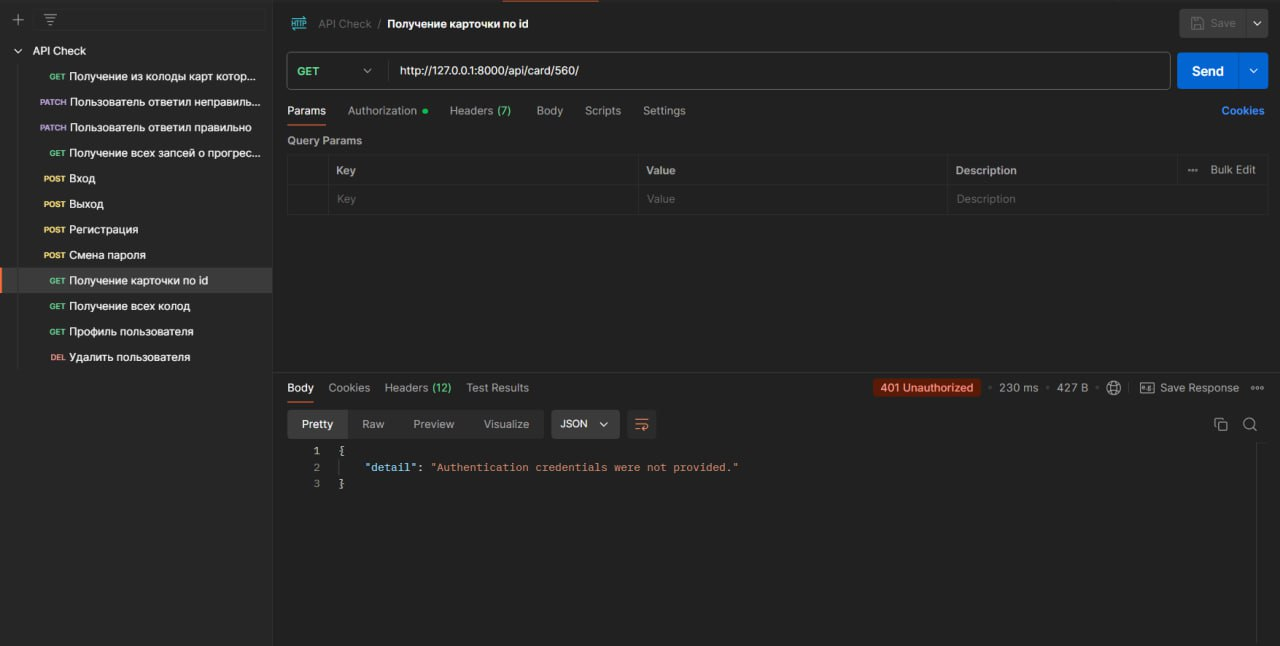


Рисунок 1 – Скриншот неавторизованного запроса информации о карточке

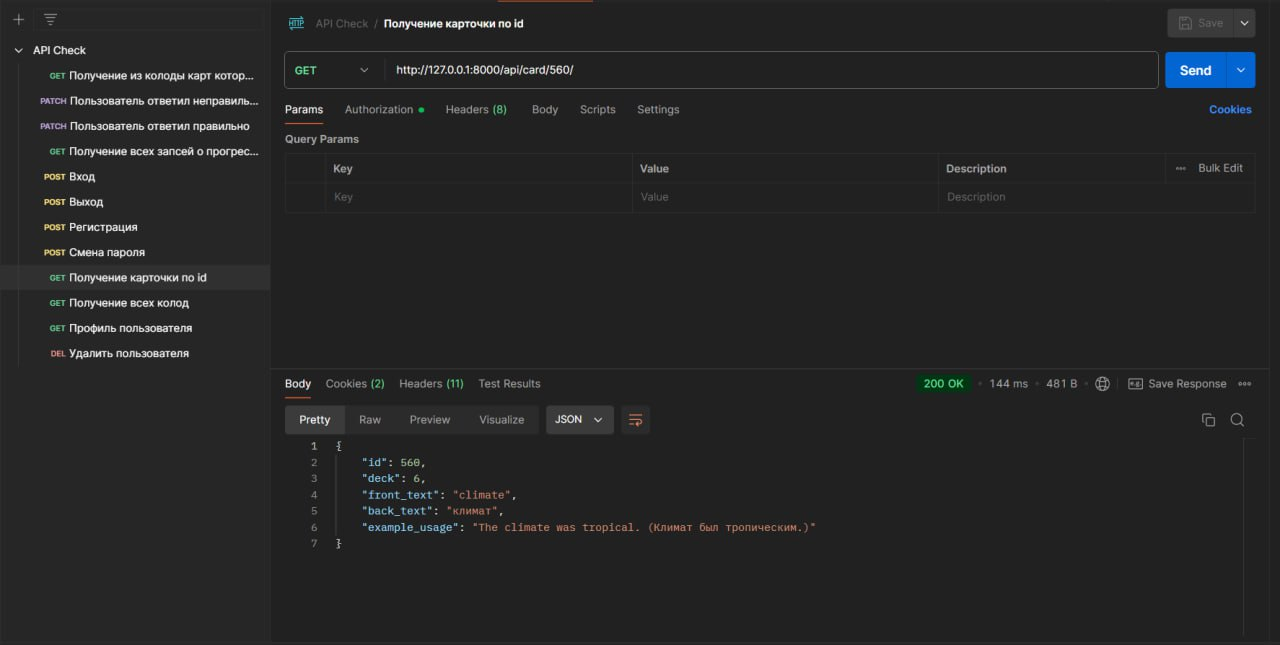


Рисунок 2 – Скриншот авторизованного запроса информации о карточке

На рисунке 3 представлен запрос к ресурсу без ограничений в доступе. Можно заметить, что информация необходимая для аутентификации пользователя, передаваемая в Cookie файлах на 3 рисунке отсутствует, однако запрос все равно успешно выполняется, поскольку доступ к эндпоинту не ограничен.

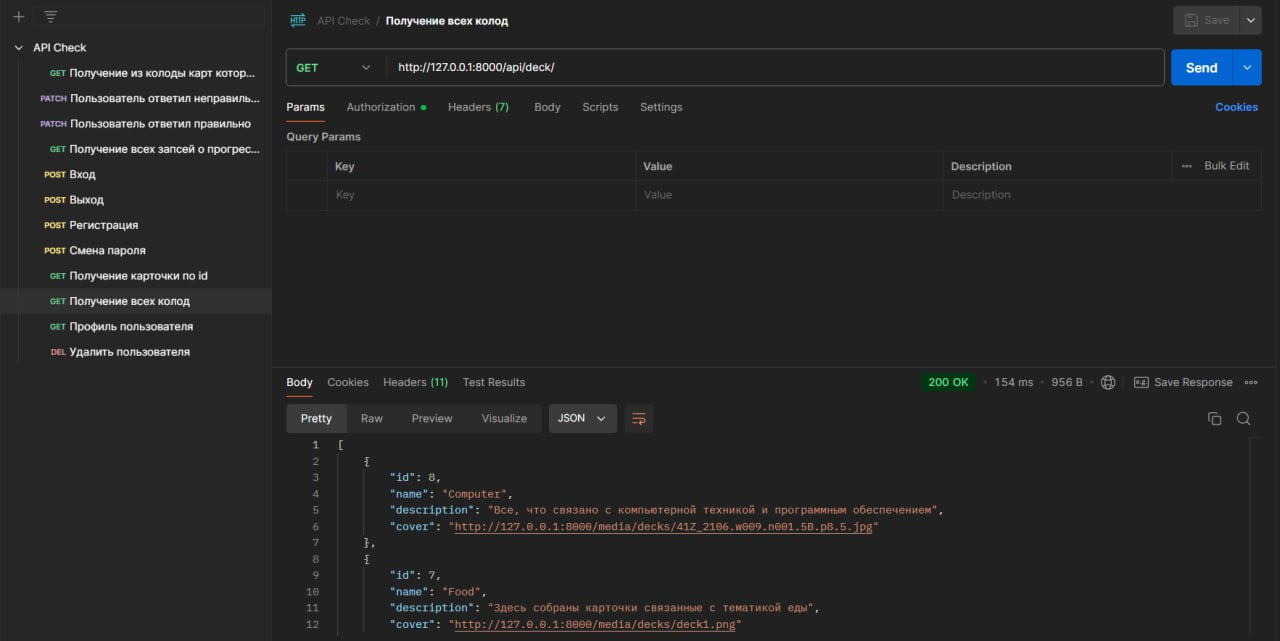


Рисунок 3 – Скриншот неавторизованного запроса информации о колодах

Приложение А

Ссылка на GitHub с проектом

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

Приложение Б

Код файла “serializers.py”

from rest\_framework.serializers import ModelSerializer, Serializer, CharField, ValidationError

from cards.models import Card

from decks.models import Deck

from users\_progress.models import Progress

from django.contrib.auth import get\_user\_model

class CardSerializer(ModelSerializer):

class Meta:

model = Card

fields = ['id', 'deck', 'front\_text', 'back\_text', 'example\_usage']

class DeckSerializer(ModelSerializer):

class Meta:

model = Deck

fields = ['id', 'name', 'description', 'cover']

class UserProgressSerializer(ModelSerializer):

class Meta:

model = Progress

fields = ['id', 'user', 'card','attempts', 'successful\_attempts']

class UserSerializer(ModelSerializer):

class Meta:

model = get\_user\_model()

fields = ['username', 'email', 'first\_name', 'last\_name']

class ChangePasswordSerializer(Serializer):

old\_password = CharField(required=True)

new\_password = CharField(required=True)

def validate\_old\_password(self, value):

user = self.context['user']

if not user.check\_password(value):

raise ValidationError("Current password is incorrect.")

return value

def validate(self, data):

old\_password = data.get('old\_password')

new\_password = data.get('new\_password')

if old\_password == new\_password:

raise ValidationError("New password must be different from the old password.")

return data

Код файла “views.py”

from api.serializers import CardSerializer, DeckSerializer, UserProgressSerializer, UserSerializer, ChangePasswordSerializer

from rest\_framework.response import Response

from rest\_framework import permissions, generics, views

from cards.models import Card

from decks.models import Deck

from users\_progress.models import Progress

from django.contrib.auth import get\_user\_model

from rest\_framework\_simplejwt.tokens import RefreshToken

from django.contrib.auth import authenticate

class CardRetrieve(generics.RetrieveAPIView):

queryset = Card.objects.all()

serializer\_class = CardSerializer

permission\_classes = [permissions.IsAuthenticated]

class DeckList(generics.ListAPIView):

queryset = Deck.objects.all()

serializer\_class = DeckSerializer

class DeckRetrieve(generics.RetrieveAPIView):

queryset = Deck.objects.all()

serializer\_class = DeckSerializer

permission\_classes = [permissions.IsAuthenticated]

def get(self, request, \*args, \*\*kwargs):

deck = self.get\_object()

queryset = (Card.objects.filter(deck = deck)

.exclude(id\_\_in = [progress.card.id for progress in

Progress.objects.filter(user = request.user)]))

serializer = CardSerializer(queryset, many = True)

return Response(serializer.data)

class UserProgressListCreate(generics.ListCreateAPIView):

serializer\_class = UserProgressSerializer

permission\_classes = (permissions.IsAuthenticated,)

def get\_queryset(self):

if self.request.user.is\_staff:

return Progress.objects.all()

return Progress.objects.filter(user = self.request.user.id)

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

progress\_record = Progress.objects.create(

user = request.user,

card = Card.objects.get(id = request.data.get('card')),

attempts = 0,

successful\_attempts = 0

)

progress\_record.save()

return Response("Card added to your deck successfully!",200)

class UserAnswerCheck(generics.UpdateAPIView):

serializer\_class = UserProgressSerializer

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

if action == "right":

progress = Progress.objects.get(card = request.data.get('card\_id'), user = request.user)

progress.attempts += 1

progress.successful\_attempts += 1

serializer = self.get\_serializer(progress, data=request.data, partial= True)

serializer.is\_valid(raise\_exception=True)

self.perform\_update(serializer)

return Response(serializer.data)

elif action == "wrong":

progress = Progress.objects.get(card = request.data.get('card\_id'), user = request.user)

progress.attempts += 1

serializer = self.get\_serializer(progress, data=request.data, partial=True)

serializer.is\_valid(raise\_exception=True)

self.perform\_update(serializer)

return Response(serializer.data)

class UserList(generics.ListAPIView):

serializer\_class = UserSerializer

permission\_classes = (permissions.IsAuthenticated,)

def get\_queryset(self):

return get\_user\_model().objects.filter(id = self.request.user.id)

class UserDelete(views.APIView):

def delete(self, request, \*args, \*\*kwargs):

user = request.user

if not user.is\_authenticated:

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

user.delete()

response = Response({'message': 'Account deleted successfully!'}, 200)

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

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

return response

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

elif action == 'edit':

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

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

user = request.user

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

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

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

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

user.save()

return Response("Profile editing completed successfully!",200)

else:

return Response(f"Bad Request. We dont have {action}/ page...", status = 400)

Код файла “urls.py”

from django.urls import path

from api.views import UserProgressListCreate, UserAnswerCheck, CardRetrieve, DeckList

from api.views import DeckRetrieve, UserList, UserAuth, UserDelete

urlpatterns = [

path('card/<int:pk>/', CardRetrieve.as\_view()),

path('deck/', DeckList.as\_view()),

path('deck/<int:pk>/', DeckRetrieve.as\_view()),

path('auth/<str:action>/', UserAuth.as\_view()),

path('profile/', UserList.as\_view()),

path('progress/', UserProgressListCreate.as\_view()),

path('progress/<str:action>/', UserAnswerCheck.as\_view()),

path('delete/', UserDelete.as\_view()),

]