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: Ашугян Т.А., гр.6303

Проверил: Кашапов А.И.

Самара 2025

СОДЕРЖАНИЕ

[Введение 16](#_Toc199593793)

[1 Структура API 17](#_Toc199593794)

[2 Обработка запросов и маршрутизация 18](#_Toc199593795)

[Приложение А 19](#_Toc199593796)

[Ссылка на GitHub с проектом 19](#_Toc199593797)

[Приложение Б 20](#_Toc199593798)

[Код файла “serializers.py” 21](#_Toc199593799)

[from datetime import time, date, datetime 21](#_Toc199593800)

[from rest\_framework import serializers, request 21](#_Toc199593801)

[from .models import Client, Master, Service, Appointment 21](#_Toc199593802)

[from django.contrib.auth.models import User 21](#_Toc199593803)

[class UserSerializer(serializers.ModelSerializer): 21](#_Toc199593804)

[class Meta: 21](#_Toc199593805)

[model = User 21](#_Toc199593806)

[fields = ['login', 'password', 'is\_client', 'is\_master'] 21](#_Toc199593807)

[class ClientRegisterSerializer(serializers.ModelSerializer): 21](#_Toc199593808)

[username = serializers.CharField(write\_only=True) 21](#_Toc199593809)

[password = serializers.CharField(write\_only=True) 21](#_Toc199593810)

[class Meta: 21](#_Toc199593811)

[model = Client 21](#_Toc199593812)

[fields = ['username', 'password', 'name', 'phone'] 21](#_Toc199593813)

[def create(self, validated\_data): 21](#_Toc199593814)

[username = validated\_data.pop('username') 21](#_Toc199593815)

[password = validated\_data.pop('password') 21](#_Toc199593816)

[user = User.objects.create\_user(username=username, password=password) 21](#_Toc199593817)

[client = Client.objects.create(user=user, \*\*validated\_data) 21](#_Toc199593818)

[return client 21](#_Toc199593819)

[class MasterSerializer(serializers.ModelSerializer): 21](#_Toc199593820)

[class Meta: 21](#_Toc199593821)

[model = Master 21](#_Toc199593822)

[fields = '\_\_all\_\_' 21](#_Toc199593823)

[class ServiceSerializer(serializers.ModelSerializer): 21](#_Toc199593824)

[class Meta: 21](#_Toc199593825)

[model = Service 21](#_Toc199593826)

[fields = '\_\_all\_\_' 21](#_Toc199593827)

[class AppointmentSerializer(serializers.ModelSerializer): 21](#_Toc199593828)

[client\_name = serializers.CharField(source='client.name', read\_only=True) 21](#_Toc199593829)

[master\_name = serializers.CharField(source='master.full\_name', read\_only=True) 22](#_Toc199593830)

[service\_info = serializers.CharField(source='service.info', read\_only=True) 22](#_Toc199593831)

[class Meta: 22](#_Toc199593832)

[model = Appointment 22](#_Toc199593833)

[fields = [ 22](#_Toc199593834)

['id', 'client', 'client\_name', 22](#_Toc199593835)

['master', 'master\_name', 22](#_Toc199593836)

['service\_info', 22](#_Toc199593837)

['date', 'time', 'status' 22](#_Toc199593838)

[] 22](#_Toc199593839)

[class AppointmentCreateSerializer(serializers.ModelSerializer): 22](#_Toc199593840)

[time = serializers.TimeField(format='%H:%M', input\_formats=['%H:%M', '%H:%M:%S']) 22](#_Toc199593841)

[class Meta: 22](#_Toc199593842)

[model = Appointment 22](#_Toc199593843)

[fields = ['date', 'time', 'service', 'master'] 22](#_Toc199593844)

[def validate(self, data): 22](#_Toc199593845)

[selected\_date = data['date'] 22](#_Toc199593846)

[selected\_time = data['time'] 22](#_Toc199593847)

[master = data['master'] 22](#_Toc199593848)

[selected\_date\_str = "2025-05-30" 22](#_Toc199593849)

[selected\_date = datetime.strptime(selected\_date\_str, "%Y-%m-%d").date() 22](#_Toc199593850)

[if selected\_date < date.today(): 22](#_Toc199593851)

[raise serializers.ValidationError({'date': 'Дата не может быть в прошлом.'}) 22](#_Toc199593852)

[exists = Appointment.objects.filter( 22](#_Toc199593853)

[master=master, 22](#_Toc199593854)

[date=selected\_date, 22](#_Toc199593855)

[time=selected\_time 22](#_Toc199593856)

[).exists() 22](#_Toc199593857)

[if exists: 22](#_Toc199593858)

[raise serializers.ValidationError({'time': 'Мастер уже занят на это время.'}) 22](#_Toc199593859)

[return data 23](#_Toc199593860)

[def create(self, validated\_data): 23](#_Toc199593861)

[return Appointment.objects.create(client=self.context['request'].user.client, \*\*validated\_data) 23](#_Toc199593862)

[Код файла “views.py” 23](#_Toc199593863)

[from datetime import datetime, timezone, timedelta 23](#_Toc199593864)

[from django.contrib.auth import login, authenticate, update\_session\_auth\_hash, logout 23](#_Toc199593865)

[from django.contrib.auth.decorators import login\_required 23](#_Toc199593866)

[from django.contrib.auth.password\_validation import validate\_password 23](#_Toc199593867)

[from django.core.exceptions import ValidationError 23](#_Toc199593868)

[from django.shortcuts import render, redirect 23](#_Toc199593869)

[from django.utils.timezone import now 23](#_Toc199593870)

[from django.views.decorators.csrf import csrf\_exempt 23](#_Toc199593871)

[from rest\_framework import viewsets, status 23](#_Toc199593872)

[from rest\_framework.permissions import IsAuthenticated, IsAdminUser 23](#_Toc199593873)

[from rest\_framework.views import APIView 23](#_Toc199593874)

[from rest\_framework.response import Response 23](#_Toc199593875)

[from rest\_framework\_simplejwt.exceptions import TokenError 23](#_Toc199593876)

[from .models import Client, Master, Service, Appointment 23](#_Toc199593877)

[from .permissions import IsClient, IsMaster 23](#_Toc199593878)

[from .serializers import \* 23](#_Toc199593879)

[from rest\_framework\_simplejwt.tokens import RefreshToken, AccessToken 23](#_Toc199593880)

[from django.contrib.auth.models import User 23](#_Toc199593881)

[def check\_and\_refresh\_token(request): 23](#_Toc199593882)

[access\_token = request.session.get('access\_token') 23](#_Toc199593883)

[refresh\_token = request.session.get('refresh\_token') 23](#_Toc199593884)

[if not access\_token or not refresh\_token: 23](#_Toc199593885)

[logout(request) 23](#_Toc199593886)

[return False 23](#_Toc199593887)

[try: 23](#_Toc199593888)

[access = AccessToken(access\_token) 23](#_Toc199593889)

[if access['exp'] < datetime.now(timezone.utc).timestamp(): 23](#_Toc199593890)

[raise TokenError("Access token expired") 23](#_Toc199593891)

[return True 23](#_Toc199593892)

[except TokenError: 24](#_Toc199593893)

[try: 24](#_Toc199593894)

[refresh = RefreshToken(refresh\_token) 24](#_Toc199593895)

[new\_access = refresh.access\_token 24](#_Toc199593896)

[request.session['access\_token'] = str(new\_access) 24](#_Toc199593897)

[return True 24](#_Toc199593898)

[except TokenError: 24](#_Toc199593899)

[logout(request) 24](#_Toc199593900)

[return False 24](#_Toc199593901)

[def generate\_time\_slots(start='09:00', end='18:00', step\_minutes=30): 24](#_Toc199593902)

[slots = [] 24](#_Toc199593903)

[current\_dt = datetime.combine(date.today(), datetime.strptime(start, "%H:%M").time()) 24](#_Toc199593904)

[end\_dt = datetime.combine(date.today(), datetime.strptime(end, "%H:%M").time()) 24](#_Toc199593905)

[while current\_dt <= end\_dt: 24](#_Toc199593906)

[slots.append(current\_dt.time()) 24](#_Toc199593907)

[current\_dt += timedelta(minutes=step\_minutes) 24](#_Toc199593908)

[return slots 24](#_Toc199593909)

[# class ClientViewSet(viewsets.ModelViewSet): 24](#_Toc199593910)

[# queryset = Client.objects.all() 24](#_Toc199593911)

[# serializer\_class = ClientRegisterSerializer 24](#_Toc199593912)

[class MasterViewSet(viewsets.ModelViewSet): 24](#_Toc199593913)

[permission\_classes = [IsAdminUser] 24](#_Toc199593914)

[queryset = Master.objects.all() 24](#_Toc199593915)

[serializer\_class = MasterSerializer 24](#_Toc199593916)

[class ServiceViewSet(viewsets.ReadOnlyModelViewSet): 24](#_Toc199593917)

[queryset = Service.objects.all() 24](#_Toc199593918)

[serializer\_class = ServiceSerializer 24](#_Toc199593919)

[class AppointmentViewSet(viewsets.ModelViewSet): 24](#_Toc199593920)

[queryset = Appointment.objects.all() 24](#_Toc199593921)

[serializer\_class = AppointmentSerializer 24](#_Toc199593922)

[class ServiceListView(APIView): 24](#_Toc199593923)

[def get(self, request): 24](#_Toc199593924)

[services = Service.objects.all() 25](#_Toc199593925)

[serializer = ServiceSerializer(services, many=True) 25](#_Toc199593926)

[return Response(serializer.data) 25](#_Toc199593927)

[class MyAppointmentsView(APIView): 25](#_Toc199593928)

[permission\_classes = [IsAuthenticated] 25](#_Toc199593929)

[def get(self, request): 25](#_Toc199593930)

[user = request.user 25](#_Toc199593931)

[if hasattr(user, 'client'): 25](#_Toc199593932)

[appointments = Appointment.objects.filter(client=user.client) 25](#_Toc199593933)

[elif hasattr(user, 'master'): 25](#_Toc199593934)

[appointments = Appointment.objects.filter(master=user.master) 25](#_Toc199593935)

[else: 25](#_Toc199593936)

[return Response({'error': 'Пользователь не является ни клиентом, ни мастером'}, status=403) 25](#_Toc199593937)

[serializer = AppointmentSerializer(appointments.order\_by("date", "time"), many=True) 25](#_Toc199593938)

[return Response(serializer.data) 25](#_Toc199593939)

[class ClientRegisterView(APIView): 25](#_Toc199593940)

[def post(self, request): 25](#_Toc199593941)

[serializer = ClientRegisterSerializer(data=request.data) 25](#_Toc199593942)

[if serializer.is\_valid(): 25](#_Toc199593943)

[serializer.save() 25](#_Toc199593944)

[return Response({'message': 'Клиент зарегистрирован'}, status=status.HTTP\_201\_CREATED) 25](#_Toc199593945)

[return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST) 25](#_Toc199593946)

[class LoginView(APIView): 25](#_Toc199593947)

[def post(self, request): 25](#_Toc199593948)

[login = request.data.get('login') 25](#_Toc199593949)

[password = request.data.get('password') 25](#_Toc199593950)

[user = authenticate(request, username=login, password=password) 25](#_Toc199593951)

[if user is None: 25](#_Toc199593952)

[return Response({'error': 'Неверный логин или пароль'}, status=401) 25](#_Toc199593953)

[if user.is\_client: 25](#_Toc199593954)

[user\_type = 'client' 25](#_Toc199593955)

[elif user.is\_master: 26](#_Toc199593956)

[user\_type = 'master' 26](#_Toc199593957)

[else: 26](#_Toc199593958)

[user\_type = 'unknown' 26](#_Toc199593959)

[refresh = RefreshToken.for\_user(user) 26](#_Toc199593960)

[refresh['user\_type'] = user\_type 26](#_Toc199593961)

[access = refresh.access\_token 26](#_Toc199593962)

[return Response({ 26](#_Toc199593963)

['refresh': str(refresh), 26](#_Toc199593964)

['access': str(access), 26](#_Toc199593965)

['user\_type': user\_type 26](#_Toc199593966)

[}, status=status.HTTP\_200\_OK) 26](#_Toc199593967)

[class ChangePasswordView(APIView): 26](#_Toc199593968)

[permission\_classes = [IsAuthenticated] 26](#_Toc199593969)

[def post(self, request): 26](#_Toc199593970)

[user = request.user 26](#_Toc199593971)

[old\_password = request.data.get("old\_password") 26](#_Toc199593972)

[new\_password = request.data.get("new\_password") 26](#_Toc199593973)

[if not user.check\_password(old\_password): 26](#_Toc199593974)

[return Response({"detail": "Неверный текущий пароль"}, status=status.HTTP\_400\_BAD\_REQUEST) 26](#_Toc199593975)

[if not new\_password: 26](#_Toc199593976)

[return Response({"detail": "Новый пароль не может быть пустым"}, status=status.HTTP\_400\_BAD\_REQUEST) 26](#_Toc199593977)

[if old\_password == new\_password: 26](#_Toc199593978)

[return Response({"detail": "Новый пароль не должен совпадать с текущим"}, status=status.HTTP\_400\_BAD\_REQUEST) 26](#_Toc199593979)

[try: 26](#_Toc199593980)

[validate\_password(new\_password, user) 26](#_Toc199593981)

[except ValidationError as e: 26](#_Toc199593982)

[return Response({"detail": e.messages}, status=status.HTTP\_400\_BAD\_REQUEST) 26](#_Toc199593983)

[user.set\_password(new\_password) 26](#_Toc199593984)

[user.save() 27](#_Toc199593985)

[return Response({"detail": "Пароль успешно изменён"}) 27](#_Toc199593986)

[class LogoutView(APIView): 27](#_Toc199593987)

[permission\_classes = [IsAuthenticated] 27](#_Toc199593988)

[def post(self, request): 27](#_Toc199593989)

[try: 27](#_Toc199593990)

[refresh\_token = request.data["refresh"] 27](#_Toc199593991)

[token = RefreshToken(refresh\_token) 27](#_Toc199593992)

[token.blacklist() 27](#_Toc199593993)

[return Response({"detail": "Вы успешно вышли из системы"}, status=status.HTTP\_205\_RESET\_CONTENT) 27](#_Toc199593994)

[except Exception: 27](#_Toc199593995)

[return Response({"detail": "Ошибка отзыва токена"}, status=status.HTTP\_400\_BAD\_REQUEST) 27](#_Toc199593996)

[class ClientAppointmentCreateView(APIView): 27](#_Toc199593997)

[permission\_classes = [IsAuthenticated, IsClient] 27](#_Toc199593998)

[def post(self, request): 27](#_Toc199593999)

[serializer = AppointmentCreateSerializer(data=request.data, context={'request': request}) 27](#_Toc199594000)

[if serializer.is\_valid(): 27](#_Toc199594001)

[serializer.save() 27](#_Toc199594002)

[return Response({'detail': 'Запись успешно создана'}, status=status.HTTP\_201\_CREATED) 27](#_Toc199594003)

[return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST) 27](#_Toc199594004)

[# HTML 27](#_Toc199594005)

[@csrf\_exempt 27](#_Toc199594006)

[def register\_view(request): 27](#_Toc199594007)

[if request.method == 'POST': 27](#_Toc199594008)

[name = request.POST.get('name') 27](#_Toc199594009)

[phone = request.POST.get('phone') 27](#_Toc199594010)

[login\_ = request.POST.get('login') 27](#_Toc199594011)

[password = request.POST.get('password') 27](#_Toc199594012)

[if User.objects.filter(username=login\_).exists(): 27](#_Toc199594013)

[return render(request, 'register.html', {'error': 'Логин уже занят'}) 27](#_Toc199594014)

[user = User.objects.create\_user( 28](#_Toc199594015)

[username=login\_, 28](#_Toc199594016)

[password=password, 28](#_Toc199594017)

[) 28](#_Toc199594018)

[user.is\_client = True 28](#_Toc199594019)

[user.save() 28](#_Toc199594020)

[Client.objects.create( 28](#_Toc199594021)

[user=user, 28](#_Toc199594022)

[name=name, 28](#_Toc199594023)

[phone=phone 28](#_Toc199594024)

[) 28](#_Toc199594025)

[return redirect('login') 28](#_Toc199594026)

[return render(request, 'register.html') 28](#_Toc199594027)

[def login\_view(request): 28](#_Toc199594028)

[if request.method == 'POST': 28](#_Toc199594029)

[login\_ = request.POST.get('login') 28](#_Toc199594030)

[password = request.POST.get('password') 28](#_Toc199594031)

[user = authenticate(request, username=login\_, password=password) 28](#_Toc199594032)

[if user is not None: 28](#_Toc199594033)

[login(request, user) 28](#_Toc199594034)

[refresh = RefreshToken.for\_user(user) 28](#_Toc199594035)

[request.session['access\_token'] = str(refresh.access\_token) 28](#_Toc199594036)

[request.session['refresh\_token'] = str(refresh) 28](#_Toc199594037)

[if user.is\_superuser or user.is\_staff: 28](#_Toc199594038)

[return redirect('admin\_dashboard') 28](#_Toc199594039)

[return redirect('services') 28](#_Toc199594040)

[else: 28](#_Toc199594041)

[return render(request, 'login.html', {'error': 'Неверный логин или пароль'}) 28](#_Toc199594042)

[return render(request, 'login.html') 28](#_Toc199594043)

[@login\_required 28](#_Toc199594044)

[def appointments\_view(request): 28](#_Toc199594045)

[if not check\_and\_refresh\_token(request): 28](#_Toc199594046)

[return redirect('login') 28](#_Toc199594047)

[user = request.user 28](#_Toc199594048)

[if hasattr(user, 'client'): 28](#_Toc199594049)

[appointments = Appointment.objects.filter(client=user.client).order\_by("date", "time") 29](#_Toc199594050)

[elif hasattr(user, 'master'): 29](#_Toc199594051)

[appointments = Appointment.objects.filter(master=user.master).order\_by("date", "time") 29](#_Toc199594052)

[else: 29](#_Toc199594053)

[appointments = [] 29](#_Toc199594054)

[return render(request, 'appointments.html', {'appointments': appointments}) 29](#_Toc199594055)

[@login\_required 29](#_Toc199594056)

[def appointment\_create\_view(request): 29](#_Toc199594057)

[if not hasattr(request.user, 'client'): 29](#_Toc199594058)

[return redirect('appointments') 29](#_Toc199594059)

[services = Service.objects.all() 29](#_Toc199594060)

[masters = Master.objects.all() 29](#_Toc199594061)

[time\_slots = None 29](#_Toc199594062)

[errors = None 29](#_Toc199594063)

[selected\_date = '' 29](#_Toc199594064)

[selected\_master = '' 29](#_Toc199594065)

[if request.method == 'GET': 29](#_Toc199594066)

[selected\_date = request.GET.get('date') 29](#_Toc199594067)

[selected\_master = request.GET.get('master') 29](#_Toc199594068)

[if selected\_date and selected\_master: 29](#_Toc199594069)

[try: 29](#_Toc199594070)

[date\_obj = datetime.strptime(selected\_date, "%Y-%m-%d").date() 29](#_Toc199594071)

[if date\_obj < date.today(): 29](#_Toc199594072)

[errors = {'date': ['Нельзя выбрать прошедшую дату.']} 29](#_Toc199594073)

[else: 29](#_Toc199594074)

[master = Master.objects.get(id=selected\_master) 29](#_Toc199594075)

[all\_slots = generate\_time\_slots() 29](#_Toc199594076)

[busy\_slots = Appointment.objects.filter( 29](#_Toc199594077)

[date=date\_obj, 29](#_Toc199594078)

[master=master 29](#_Toc199594079)

[).values\_list('time', flat=True) 29](#_Toc199594080)

[time\_slots = [slot for slot in all\_slots if slot not in busy\_slots] 30](#_Toc199594081)

[except (ValueError, Master.DoesNotExist): 30](#_Toc199594082)

[time\_slots = [] 30](#_Toc199594083)

[elif request.method == 'POST': 30](#_Toc199594084)

[selected\_date = request.POST.get('date', '') 30](#_Toc199594085)

[selected\_master = request.POST.get('master', '') 30](#_Toc199594086)

[serializer = AppointmentCreateSerializer(data=request.POST, context={'request': request}) 30](#_Toc199594087)

[if serializer.is\_valid(): 30](#_Toc199594088)

[serializer.save() 30](#_Toc199594089)

[return redirect('appointments') 30](#_Toc199594090)

[else: 30](#_Toc199594091)

[errors = serializer.errors 30](#_Toc199594092)

[return render(request, 'appointment\_create.html', { 30](#_Toc199594093)

['services': services, 30](#_Toc199594094)

['masters': masters, 30](#_Toc199594095)

['time\_slots': time\_slots, 30](#_Toc199594096)

['errors': errors, 30](#_Toc199594097)

['selected\_date': selected\_date, 30](#_Toc199594098)

['selected\_master': selected\_master, 30](#_Toc199594099)

[}) 30](#_Toc199594100)

[@login\_required 30](#_Toc199594101)

[def change\_password\_view(request): 30](#_Toc199594102)

[if not check\_and\_refresh\_token(request): 30](#_Toc199594103)

[return redirect('login') 30](#_Toc199594104)

[if request.method == 'POST': 30](#_Toc199594105)

[old\_password = request.POST.get('old\_password') 30](#_Toc199594106)

[new\_password = request.POST.get('new\_password') 30](#_Toc199594107)

[user = request.user 30](#_Toc199594108)

[if not user.check\_password(old\_password): 30](#_Toc199594109)

[return render(request, 'change\_password.html', {'error': 'Неверный текущий пароль'}) 30](#_Toc199594110)

[if old\_password == new\_password: 30](#_Toc199594111)

[return render(request, 'change\_password.html', {'error': 'Новый пароль не должен совпадать с текущим'}) 30](#_Toc199594112)

[# try: 31](#_Toc199594113)

[# validate\_password(new\_password, user) 31](#_Toc199594114)

[# except ValidationError as e: 31](#_Toc199594115)

[# return render(request, 'change\_password.html', {'error': e.messages}) 31](#_Toc199594116)

[user.set\_password(new\_password) 31](#_Toc199594117)

[user.save() 31](#_Toc199594118)

[update\_session\_auth\_hash(request, user) 31](#_Toc199594119)

[return render(request, 'change\_password.html', {'success': 'Пароль успешно изменён'}) 31](#_Toc199594120)

[return render(request, 'change\_password.html') 31](#_Toc199594121)

[def services\_view(request): 31](#_Toc199594122)

[services = Service.objects.all() 31](#_Toc199594123)

[return render(request, 'services.html', {'services': services}) 31](#_Toc199594124)

[@login\_required 31](#_Toc199594125)

[def logout\_view(request): 31](#_Toc199594126)

[request.session.flush() 31](#_Toc199594127)

[return redirect('login') 31](#_Toc199594128)

[Код файла “urls.py” 31](#_Toc199594129)

Введение

AutoService – приложение для записи на услуги в автосервис. Стек технологий приложения: Python, Django, PostgreSQL. Все компоненты конечной системы контейнеризованны с помощью Docker для обеспечения изоляции окружения, удобства развертывания и масштабирования.

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 |
|  |  |  |  |  |  |

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.

Приложение А

Ссылка на GitHub с проектом

https://github.com/Tigr195/AutoService

Приложение Б

Код файла “serializers.py”

from datetime import time, date, datetime

from rest\_framework import serializers, request

from .models import Client, Master, Service, Appointment

from django.contrib.auth.models import User

class UserSerializer(serializers.ModelSerializer):

class Meta:

model = User

fields = ['login', 'password', 'is\_client', 'is\_master']

class ClientRegisterSerializer(serializers.ModelSerializer):

username = serializers.CharField(write\_only=True)

password = serializers.CharField(write\_only=True)

class Meta:

model = Client

fields = ['username', 'password', 'name', 'phone']

def create(self, validated\_data):

username = validated\_data.pop('username')

password = validated\_data.pop('password')

user = User.objects.create\_user(username=username, password=password)

client = Client.objects.create(user=user, \*\*validated\_data)

return client

class MasterSerializer(serializers.ModelSerializer):

class Meta:

model = Master

fields = '\_\_all\_\_'

class ServiceSerializer(serializers.ModelSerializer):

class Meta:

model = Service

fields = '\_\_all\_\_'

class AppointmentSerializer(serializers.ModelSerializer):

client\_name = serializers.CharField(source='client.name', read\_only=True)

master\_name = serializers.CharField(source='master.full\_name', read\_only=True)

service\_info = serializers.CharField(source='service.info', read\_only=True)

class Meta:

model = Appointment

fields = [

'id', 'client', 'client\_name',

'master', 'master\_name',

'service\_info',

'date', 'time', 'status'

]

class AppointmentCreateSerializer(serializers.ModelSerializer):

time = serializers.TimeField(format='%H:%M', input\_formats=['%H:%M', '%H:%M:%S'])

class Meta:

model = Appointment

fields = ['date', 'time', 'service', 'master']

def validate(self, data):

selected\_date = data['date']

selected\_time = data['time']

master = data['master']

selected\_date\_str = "2025-05-30"

selected\_date = datetime.strptime(selected\_date\_str, "%Y-%m-%d").date()

if selected\_date < date.today():

raise serializers.ValidationError({'date': 'Дата не может быть в прошлом.'})

exists = Appointment.objects.filter(

master=master,

date=selected\_date,

time=selected\_time

).exists()

if exists:

raise serializers.ValidationError({'time': 'Мастер уже занят на это время.'})

return data

def create(self, validated\_data):

return Appointment.objects.create(client=self.context['request'].user.client, \*\*validated\_data)

Код файла “views.py”

from datetime import datetime, timezone, timedelta

from django.contrib.auth import login, authenticate, update\_session\_auth\_hash, logout

from django.contrib.auth.decorators import login\_required

from django.contrib.auth.password\_validation import validate\_password

from django.core.exceptions import ValidationError

from django.shortcuts import render, redirect

from django.utils.timezone import now

from django.views.decorators.csrf import csrf\_exempt

from rest\_framework import viewsets, status

from rest\_framework.permissions import IsAuthenticated, IsAdminUser

from rest\_framework.views import APIView

from rest\_framework.response import Response

from rest\_framework\_simplejwt.exceptions import TokenError

from .models import Client, Master, Service, Appointment

from .permissions import IsClient, IsMaster

from .serializers import \*

from rest\_framework\_simplejwt.tokens import RefreshToken, AccessToken

from django.contrib.auth.models import User

def check\_and\_refresh\_token(request):

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

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

if not access\_token or not refresh\_token:

logout(request)

return False

try:

access = AccessToken(access\_token)

if access['exp'] < datetime.now(timezone.utc).timestamp():

raise TokenError("Access token expired")

return True

except TokenError:

try:

refresh = RefreshToken(refresh\_token)

new\_access = refresh.access\_token

request.session['access\_token'] = str(new\_access)

return True

except TokenError:

logout(request)

return False

def generate\_time\_slots(start='09:00', end='18:00', step\_minutes=30):

slots = []

current\_dt = datetime.combine(date.today(), datetime.strptime(start, "%H:%M").time())

end\_dt = datetime.combine(date.today(), datetime.strptime(end, "%H:%M").time())

while current\_dt <= end\_dt:

slots.append(current\_dt.time())

current\_dt += timedelta(minutes=step\_minutes)

return slots

# class ClientViewSet(viewsets.ModelViewSet):

# queryset = Client.objects.all()

# serializer\_class = ClientRegisterSerializer

class MasterViewSet(viewsets.ModelViewSet):

permission\_classes = [IsAdminUser]

queryset = Master.objects.all()

serializer\_class = MasterSerializer

class ServiceViewSet(viewsets.ReadOnlyModelViewSet):

queryset = Service.objects.all()

serializer\_class = ServiceSerializer

class AppointmentViewSet(viewsets.ModelViewSet):

queryset = Appointment.objects.all()

serializer\_class = AppointmentSerializer

class ServiceListView(APIView):

def get(self, request):

services = Service.objects.all()

serializer = ServiceSerializer(services, many=True)

return Response(serializer.data)

class MyAppointmentsView(APIView):

permission\_classes = [IsAuthenticated]

def get(self, request):

user = request.user

if hasattr(user, 'client'):

appointments = Appointment.objects.filter(client=user.client)

elif hasattr(user, 'master'):

appointments = Appointment.objects.filter(master=user.master)

else:

return Response({'error': 'Пользователь не является ни клиентом, ни мастером'}, status=403)

serializer = AppointmentSerializer(appointments.order\_by("date", "time"), many=True)

return Response(serializer.data)

class ClientRegisterView(APIView):

def post(self, request):

serializer = ClientRegisterSerializer(data=request.data)

if serializer.is\_valid():

serializer.save()

return Response({'message': 'Клиент зарегистрирован'}, status=status.HTTP\_201\_CREATED)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

class LoginView(APIView):

def post(self, request):

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

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

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

if user is None:

return Response({'error': 'Неверный логин или пароль'}, status=401)

if user.is\_client:

user\_type = 'client'

elif user.is\_master:

user\_type = 'master'

else:

user\_type = 'unknown'

refresh = RefreshToken.for\_user(user)

refresh['user\_type'] = user\_type

access = refresh.access\_token

return Response({

'refresh': str(refresh),

'access': str(access),

'user\_type': user\_type

}, status=status.HTTP\_200\_OK)

class ChangePasswordView(APIView):

permission\_classes = [IsAuthenticated]

def post(self, request):

user = request.user

old\_password = request.data.get("old\_password")

new\_password = request.data.get("new\_password")

if not user.check\_password(old\_password):

return Response({"detail": "Неверный текущий пароль"}, status=status.HTTP\_400\_BAD\_REQUEST)

if not new\_password:

return Response({"detail": "Новый пароль не может быть пустым"}, status=status.HTTP\_400\_BAD\_REQUEST)

if old\_password == new\_password:

return Response({"detail": "Новый пароль не должен совпадать с текущим"}, status=status.HTTP\_400\_BAD\_REQUEST)

try:

validate\_password(new\_password, user)

except ValidationError as e:

return Response({"detail": e.messages}, status=status.HTTP\_400\_BAD\_REQUEST)

user.set\_password(new\_password)

user.save()

return Response({"detail": "Пароль успешно изменён"})

class LogoutView(APIView):

permission\_classes = [IsAuthenticated]

def post(self, request):

try:

refresh\_token = request.data["refresh"]

token = RefreshToken(refresh\_token)

token.blacklist()

return Response({"detail": "Вы успешно вышли из системы"}, status=status.HTTP\_205\_RESET\_CONTENT)

except Exception:

return Response({"detail": "Ошибка отзыва токена"}, status=status.HTTP\_400\_BAD\_REQUEST)

class ClientAppointmentCreateView(APIView):

permission\_classes = [IsAuthenticated, IsClient]

def post(self, request):

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

if serializer.is\_valid():

serializer.save()

return Response({'detail': 'Запись успешно создана'}, status=status.HTTP\_201\_CREATED)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

# HTML

@csrf\_exempt

def register\_view(request):

if request.method == 'POST':

name = request.POST.get('name')

phone = request.POST.get('phone')

login\_ = request.POST.get('login')

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

if User.objects.filter(username=login\_).exists():

return render(request, 'register.html', {'error': 'Логин уже занят'})

user = User.objects.create\_user(

username=login\_,

password=password,

)

user.is\_client = True

user.save()

Client.objects.create(

user=user,

name=name,

phone=phone

)

return redirect('login')

return render(request, 'register.html')

def login\_view(request):

if request.method == 'POST':

login\_ = request.POST.get('login')

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

user = authenticate(request, username=login\_, password=password)

if user is not None:

login(request, user)

refresh = RefreshToken.for\_user(user)

request.session['access\_token'] = str(refresh.access\_token)

request.session['refresh\_token'] = str(refresh)

if user.is\_superuser or user.is\_staff:

return redirect('admin\_dashboard')

return redirect('services')

else:

return render(request, 'login.html', {'error': 'Неверный логин или пароль'})

return render(request, 'login.html')

@login\_required

def appointments\_view(request):

if not check\_and\_refresh\_token(request):

return redirect('login')

user = request.user

if hasattr(user, 'client'):

appointments = Appointment.objects.filter(client=user.client).order\_by("date", "time")

elif hasattr(user, 'master'):

appointments = Appointment.objects.filter(master=user.master).order\_by("date", "time")

else:

appointments = []

return render(request, 'appointments.html', {'appointments': appointments})

@login\_required

def appointment\_create\_view(request):

if not hasattr(request.user, 'client'):

return redirect('appointments')

services = Service.objects.all()

masters = Master.objects.all()

time\_slots = None

errors = None

selected\_date = ''

selected\_master = ''

if request.method == 'GET':

selected\_date = request.GET.get('date')

selected\_master = request.GET.get('master')

if selected\_date and selected\_master:

try:

date\_obj = datetime.strptime(selected\_date, "%Y-%m-%d").date()

if date\_obj < date.today():

errors = {'date': ['Нельзя выбрать прошедшую дату.']}

else:

master = Master.objects.get(id=selected\_master)

all\_slots = generate\_time\_slots()

busy\_slots = Appointment.objects.filter(

date=date\_obj,

master=master

).values\_list('time', flat=True)

time\_slots = [slot for slot in all\_slots if slot not in busy\_slots]

except (ValueError, Master.DoesNotExist):

time\_slots = []

elif request.method == 'POST':

selected\_date = request.POST.get('date', '')

selected\_master = request.POST.get('master', '')

serializer = AppointmentCreateSerializer(data=request.POST, context={'request': request})

if serializer.is\_valid():

serializer.save()

return redirect('appointments')

else:

errors = serializer.errors

return render(request, 'appointment\_create.html', {

'services': services,

'masters': masters,

'time\_slots': time\_slots,

'errors': errors,

'selected\_date': selected\_date,

'selected\_master': selected\_master,

})

@login\_required

def change\_password\_view(request):

if not check\_and\_refresh\_token(request):

return redirect('login')

if request.method == 'POST':

old\_password = request.POST.get('old\_password')

new\_password = request.POST.get('new\_password')

user = request.user

if not user.check\_password(old\_password):

return render(request, 'change\_password.html', {'error': 'Неверный текущий пароль'})

if old\_password == new\_password:

return render(request, 'change\_password.html', {'error': 'Новый пароль не должен совпадать с текущим'})

# try:

# validate\_password(new\_password, user)

# except ValidationError as e:

# return render(request, 'change\_password.html', {'error': e.messages})

user.set\_password(new\_password)

user.save()

update\_session\_auth\_hash(request, user)

return render(request, 'change\_password.html', {'success': 'Пароль успешно изменён'})

return render(request, 'change\_password.html')

def services\_view(request):

services = Service.objects.all()

return render(request, 'services.html', {'services': services})

@login\_required

def logout\_view(request):

request.session.flush()

return redirect('login')

Код файла “urls.py”

from django.contrib import admin

from django.contrib.auth import views as auth\_views

from django.urls import path, include

from rest\_framework.routers import DefaultRouter

from rest\_framework\_simplejwt.views import TokenRefreshView, TokenObtainPairView

from Auto\_Service.views import \*

router = DefaultRouter()

#router.register(r'clients', ClientViewSet)

router.register(r'masters', MasterViewSet)

router.register(r'services', ServiceViewSet)

router.register(r'appointments', AppointmentViewSet)

urlpatterns = [

path('admin/', admin.site.urls),

path('api/', include(router.urls)),

path('api/auth/client/register/', ClientRegisterView.as\_view(), name='client-register'),

path('api/auth/login/', TokenObtainPairView.as\_view(), name='login'),

path('api/auth/token/refresh/', TokenRefreshView.as\_view(), name='token-refresh'),

path('api/user/my\_appointments/', MyAppointmentsView.as\_view(), name='appointments'),

path('api/user/change-password/', ChangePasswordView.as\_view(), name='change-password'),

path('api/user/logout/', LogoutView.as\_view(), name='logout'),

path('api/all\_services/', ServiceListView.as\_view(), name='services'),

path('api/appointments/create/', ClientAppointmentCreateView.as\_view(), name='create-appointment'),

# HTML

path('register/', register\_view, name='register'),

path('login/', login\_view, name='login'),

path('my\_appointments/', appointments\_view, name='appointments'),

path('appointments/create/', appointment\_create\_view, name='create\_appointment'),

path('change-password/', change\_password\_view, name='change\_password'),

path('services/', services\_view, name='services'),

path('logout/', logout\_view, name='logout'),

#admin

path('admin-panel/', include('Auto\_Service.admin\_urls')),

]