МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение   
высшего образования «Самарский национальный исследовательский университет   
имени академика С.П. Королева»

(Самарский университет)

Институт информатики и кибернетики

Кафедра технической кибернетики

Отчёт по курсу

"Технологии сетевого программирования"

Выполнили:

Обучающиеся группы № 6301-010302D

А.Л. Летов

Н.А. Стрельников

Самара 2025

СОДЕРЖАНИЕ

[**ВВЕДЕНИЕ** 3](#_Toc199769816)

[1 Первый этап разработки 4](#_Toc199769817)

[1.1 Основные технологии 4](#_Toc199769818)

[1.2 Основные сущности 4](#_Toc199769819)

[1.3 Схема базы данных 5](#_Toc199769820)

[2 Backend часть проекта 6](#_Toc199769821)

[2.1 API 6](#_Toc199769822)

[2.2 Миграции БД 7](#_Toc199769823)

[2.3 Модели 7](#_Toc199769824)

[2.4 Сериализаторы 8](#_Toc199769825)

[2.5 Представления 9](#_Toc199769826)

[3 Frontend часть проекта 11](#_Toc199769827)

[3.1 base.html 11](#_Toc199769828)

[3.2 index.html 11](#_Toc199769829)

[3.3 profile\friends\list.html 12](#_Toc199769830)

[3.4 groups\detail.html 12](#_Toc199769831)

[3.5 profile.html 13](#_Toc199769832)

[ПРИЛОЖЕНИЕ А Фрагменты кода backend части 14](#_Toc199769833)

[А.1 Программная реализация моделей 14](#_Toc199769834)

[А.2 Программная реализация сериализаторов пользователя 20](#_Toc199769835)

[А.3 Программная реализация представлений пользователя 22](#_Toc199769836)

[ПРИЛОЖЕНИЕ Б Фрагменты кода frontend части 27](#_Toc199769837)

[Б.1 base.html 27](#_Toc199769838)

[Б.2 index.html 32](#_Toc199769839)

[Б.3 profile/friends/list.html 49](#_Toc199769840)

[Б.4 groups/detail.html 57](#_Toc199769841)

[Б.5 profile.html 74](#_Toc199769842)

**ВВЕДЕНИЕ**

Нам было интересно разобраться в том, как работают современные социальные сети, а также как они устроены, поэтому в качестве проекта было решено попробовать реализовать собственный аналог социальной сети.

Ссылка на репозиторий git: https://github.com/nstude/Meetly

1. Первый этап разработки

Для начала необходимо было определиться с языками программирования, технологиями и фреймворками.

* 1. Основные технологии

Python – язык программирования, на котором написан весь бекенд нашего приложения. Выбор пал на данный язык, поскольку он имеет большое активное сообщество, поддержку со стороны разработчиков языка, а также мощные фреймворки (в том числе Django).

Django – это фреймворк для создания веб-приложений с помощью языка программирования Python. Django предоставляет встроенные механизмы для аутентификации и авторизации, что важно для нашего проекта.

Также Django позволяет легко работать с базой данных и строить сложные модели данных. Django ORM — инструмент, встроенный в фреймворк Django, который обеспечивает удобный способ взаимодействия с базой данных через объектно-ориентированный интерфейс.

Java Script – популярный язык программирования, который достаточно прост в освоении, а также обладает рядом преимуществ (наример, что важно для нашего проекта, ассинхронным взаимодействием)

PostgreSQL – это реляционная система управления базами данных для хранения данных.

* 1. Основные сущности

В нашей схеме представлено 5 сущностей:

Пользователь – посетитель сайта, имеющий свой профиль. Обладает возможностью подписаться на другого пользователя.

Группа – место для обмена сообщениями между пользователями. Создаются пользователями.

Пост – некоторый текст, который юзер может опубликовать. Посты видны только подписчикам пользователя. Создаются пользователями.

Сообщения (комментарии) – отправляются в группах (публикуются под постами).

Лайки – ставятся постам со стороны подписчиков пользователя.

* 1. Схема базы данных

Полная схема базы данных представлена на рисунке 1.

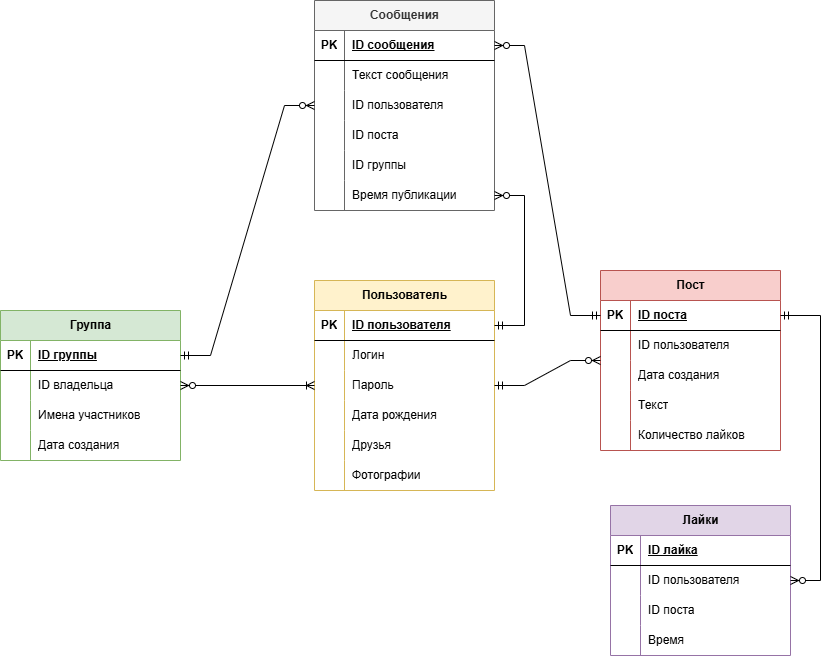


Рисунок 1 – Схема базы данных

1. Backend часть проекта
   1. API

Описание основных запросов представлено в таблице 1.

Таблица 1 – Описание основных запросов

|  |  |  |
| --- | --- | --- |
| Метод | Эндпоинт | Описание |
| POST | /users/create/ | Создание пользователя (происходит при создании профиля) |
| /profiles/create/ | Создание профиля |
| /posts/create/ | Создание поста |
| /groups/create/ | Создание группы |
| /messages/create/ | Создание сообщения |
| /likes/create/ | Создание лайка |
| GET | /users/ | Получить список всех пользователей |
| (Аналогично для остальных сущностей)… | … |
| GET | /users/<id>/ | Получить конкретного пользователя |
| … | … |
| PUT | /users/<id>/update/ | Обновление данных юзера |
| … | … |
| DELETE | /users/<id>/delete/ | Удаление пользователя |
| … | … |

В таблице 2 представлены дополнительные необходимые эндпоинты.

Таблица 2 – дополнительные запросы

|  |  |  |
| --- | --- | --- |
| Метод | Эндпоинт | Описание |
| POST | /profiles/friends/add/ | Добавление подписок (друзей) |
| /profiles/friends/remove/ | Удаление подписок (друзей) |
| /groups/<id>/members/add/ | Добавление участников в группу |
| /groups/<id>/members/remove/ | Удаление участников из группы |
| PUT | /users/change-password/ | Обновление пароля |

* 1. Миграции БД

Для синхронизации атрибутов моделей и базы данных в Django предусмотрены механизмы миграции.

С помощью команды makemigrations создаются миграции. Пример созданной миграции представлен на рисунке 2.

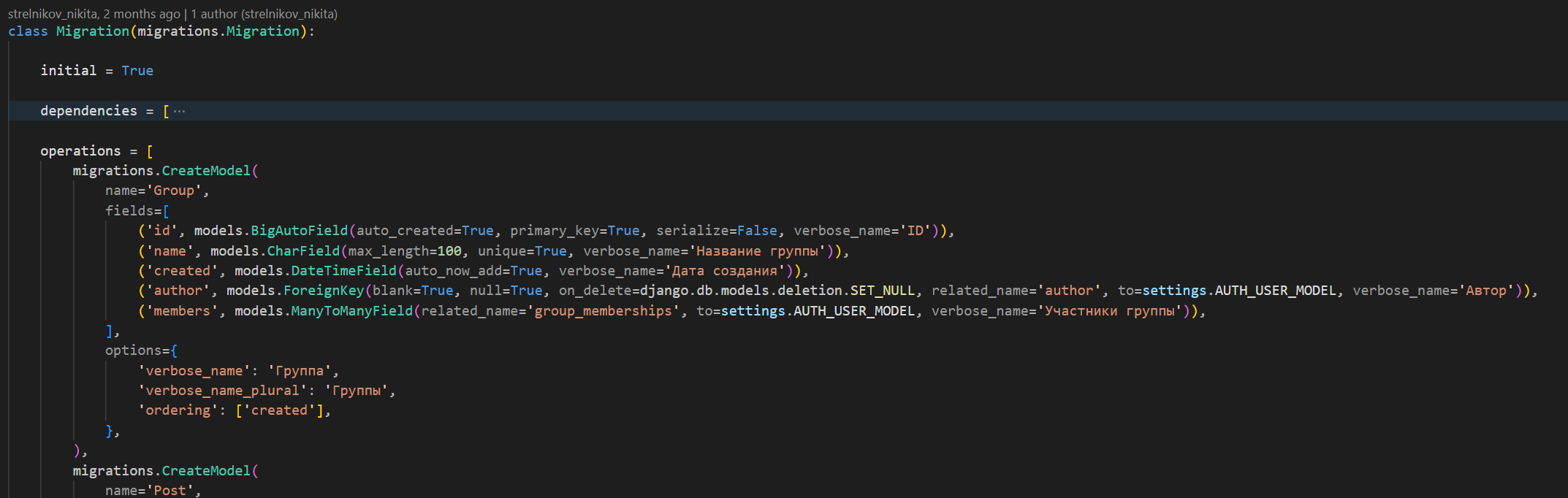


Рисунок 2 – Миграция сущности “Группа”

* 1. Модели

Для представления сущностей и связей БД в Django используются модели, которые заменяют иногда сложные и массивные SQL запросы.

Полная реализация моделей представлена в приложении А1.

Пример реализации модели профиля пользователя представлен на рисунке 3.

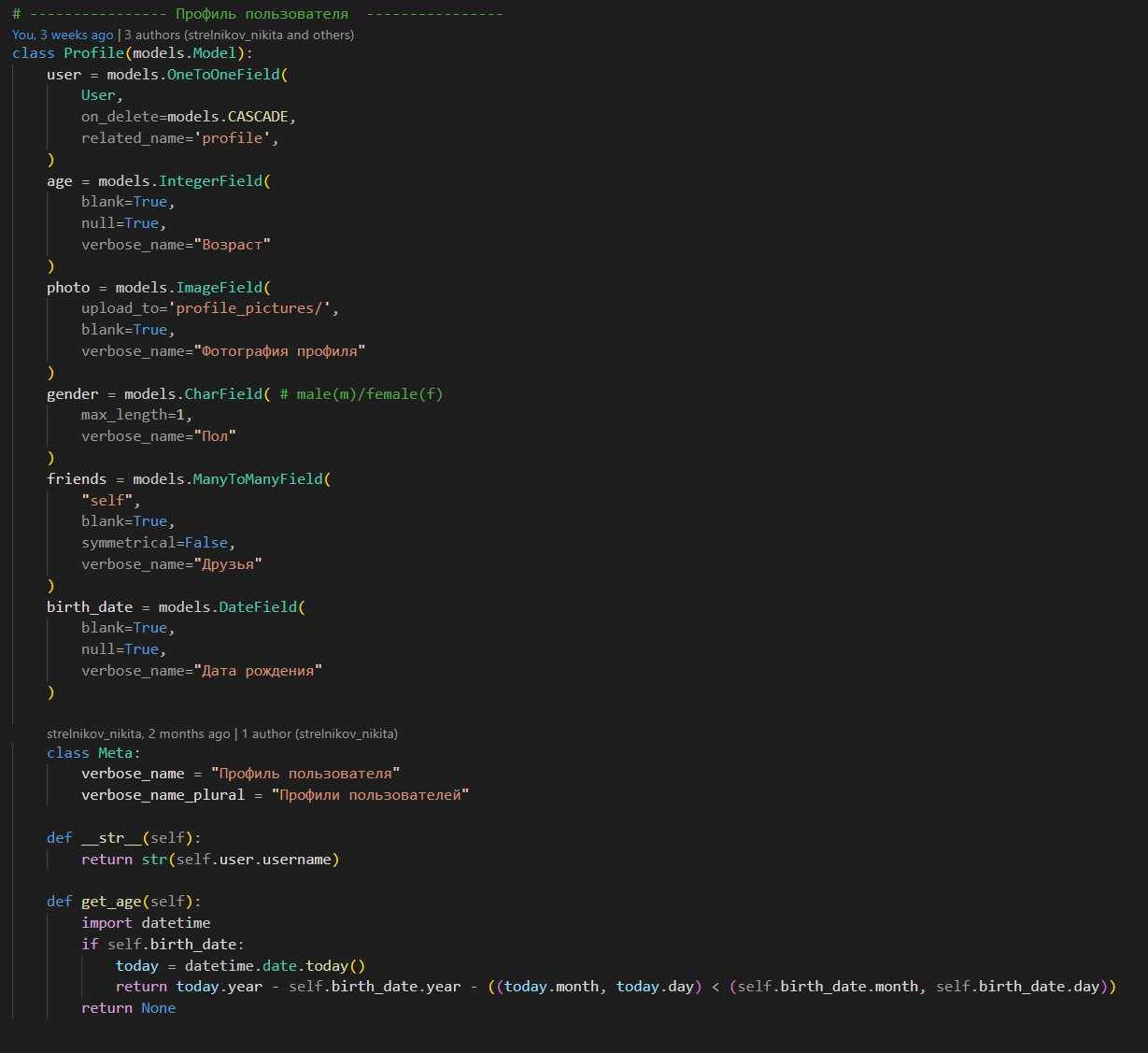


Рисунок 3 – Модель профиля

* 1. Сериализаторы

Для преобразования моделей в понятный для чтения тип данных, в Django используются сериализаторы. Для каждой модели определены все необходимые для конкретных эндпоинтов сериализаторы.

На рисунке 4 изображён сериализатор для просмотра полей пользователя.

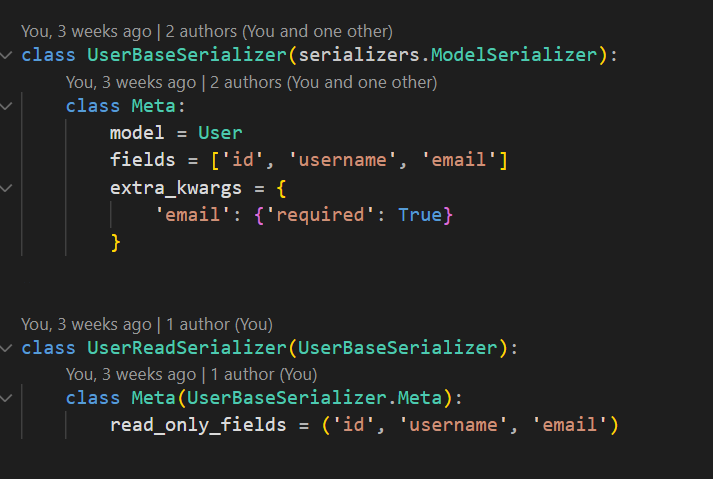


Рисунок 4 – Сериализатор для просмотра

Полный код сериализаторов для пользователя представлен в приложении А.2.

* 1. Представления

Для обработки запросов в Django используется удобный инструмент – представления. Существует всего два основных типа преставлений, у нас используются оба.

Первый для обработки запросов и возвращения определённых ответов (ок или ошибка).

Второй тип используется для рендера определённых html страниц.

На рисунке 5 представлен пример представления регистрации, возвращающего страницу, на рисунке 6 – ответ.

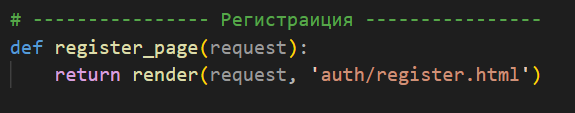


Рисунок 5 – Представление, возвращающее страницу



Рисунок 6 – Представление, возвращающее ответ

Полный код представлений для пользователя представлен в приложении А.3.

Для ограничения определённых эндпоинтов, было решено использовать декоратор @login\_required.

1. Frontend часть проекта

Frontend часть нашего проекта была написана с использованием CSS, HTML и JavaScript. Всего у нас было реализовано 11 HTML файлов. Хотелось бы рассказать про основные.

* 1. base.html

Основной шаблон, от которого расширяются другие шаблоны. В нем создается панель навигации. Функция checkAuthStatus() проверяет наличие access токена. Если он есть, то на панели навигации будут отображаться разделы «Профиль», «Друзья», «Группы», «Смена пароля». Если нет, то на панели навигации будут кнопки «Войти» и «Регистрация». Также есть функции logout() для выхода из аккаунта, tryRefreshToken() для обновления refresh токена.

* 1. index.html

Главная страница, на которой отображаются посты. У каждого поста есть кнопка лайка и кнопка комментариев. При нажатии на кнопку комментариев открывается список комментариев и текстовое поле с кнопкой отправки комментария. Также пользователь может удалить свои ранее созданные посты. При нажатии на кнопку создать пост открывается окно с текстовым полем и кнопкой отправки поста. Внешний вид страницы представлен на рисунке 7.

Изображение выглядит как снимок экрана, текст, программное обеспечение, Операционная система

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 7 – Интерфейс главной страницы

* 1. profile\friends\list.html

На этой странице отображаются подписки (друзья) пользователя. Для каждого профиля, на который подписан пользователь, есть кнопки «Профиль» (для перехода на страницу профиля этого пользователя) и «Удалить» (для удаления этого профиля из друзей). При нажатии на кнопку подписаться произойдет переход на страницу с отображением всех пользователей, которых нет в друзьях. Внешний вид страницы представлен на рисунке 8.

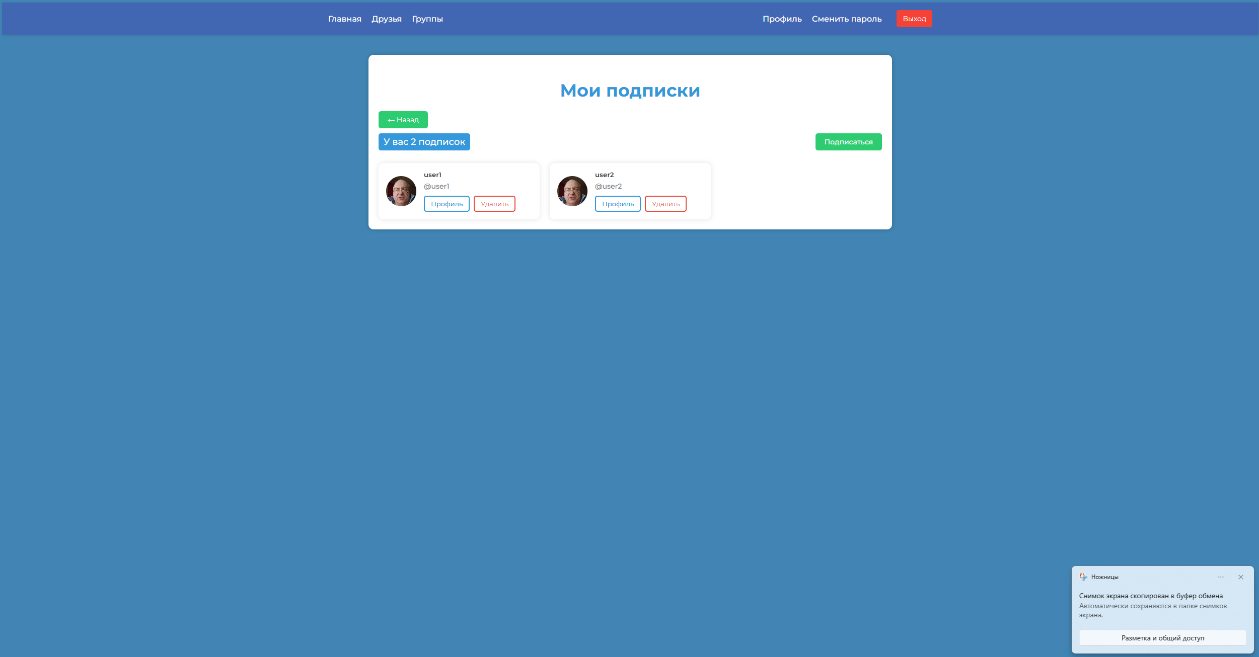


Рисунок 8 – Интерфейс страницы друзей

* 1. groups\detail.html

Страница группы, на которой в левой части отображаются сообщения группы, в правой части участники чата. Снизу текстовое поле с кнопкой для отправки сообщения в чат. Внешний вид страницы представлен на рисунке 9.

Изображение выглядит как снимок экрана, текст, программное обеспечение, Значок на компьютере

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 9 – Интерфейс страницы группы

* 1. profile.html

Это страница профиля пользователя, где он может просматривать свои данные, редактировать каждое поле отдельно, загружать и обновлять своё фото профиля. Внешний вид страницы представлен на рисунке 10.

Изображение выглядит как текст, снимок экрана, программное обеспечение, дизайн

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 9 – Интерфейс страницы пользователя

ПРИЛОЖЕНИЕ А  
Фрагменты кода backend части

А.1 Программная реализация моделей

from django.db import models

from django.contrib.auth.models import User

from django.contrib.contenttypes.fields import GenericForeignKey, GenericRelation

from django.contrib.contenttypes.models import ContentType

# ---------------- Профиль пользователя ----------------

class Profile(models.Model):

user = models.OneToOneField(

User,

on\_delete=models.CASCADE,

related\_name='profile',

)

age = models.IntegerField(

blank=True,

null=True,

verbose\_name="Возраст"

)

photo = models.ImageField(

upload\_to='profile\_pictures/',

blank=True,

verbose\_name="Фотография профиля"

)

gender = models.CharField( # male(m)/female(f)

max\_length=1,

verbose\_name="Пол"

)

friends = models.ManyToManyField(

"self",

blank=True,

symmetrical=False,

verbose\_name="Друзья"

)

birth\_date = models.DateField(

blank=True,

null=True,

verbose\_name="Дата рождения"

)

class Meta:

verbose\_name = "Профиль пользователя"

verbose\_name\_plural = "Профили пользователей"

def \_\_str\_\_(self):

return str(self.user.username)

def get\_age(self):

import datetime

if self.birth\_date:

today = datetime.date.today()

return today.year - self.birth\_date.year - ((today.month, today.day) < (self.birth\_date.month, self.birth\_date.day))

return None

# ---------------- Пост ----------------

class Post(models.Model):

author = models.ForeignKey(

User,

on\_delete=models.CASCADE,

related\_name='posts',

verbose\_name="Автор"

)

content = models.TextField(

verbose\_name="Содержание"

)

published = models.DateTimeField(

auto\_now\_add=True,

verbose\_name="Дата создания"

)

likes = models.IntegerField(

default=0

)

like\_list = GenericRelation(

'Like'

)

class Meta:

verbose\_name = "Пост"

verbose\_name\_plural = "Посты"

ordering = ['-published']

def \_\_str\_\_(self):

return self.content

# ---------------- Группа ----------------

class Group(models.Model):

name = models.CharField(

max\_length=100,

verbose\_name="Название группы"

)

author = models.ForeignKey(

User,

on\_delete=models.SET\_NULL,

null=True,

blank=True,

related\_name='author',

verbose\_name="Автор"

)

members = models.ManyToManyField(

User,

related\_name='group\_memberships',

verbose\_name="Участники группы"

)

created = models.DateTimeField(

auto\_now\_add=True,

verbose\_name="Дата создания"

)

class Meta:

verbose\_name = "Группа"

verbose\_name\_plural = "Группы"

ordering = ['created']

def \_\_str\_\_(self):

return self.name

# ---------------- Сообщение ----------------

class Message(models.Model):

author = models.ForeignKey(

User,

on\_delete=models.CASCADE,

verbose\_name="Автор"

)

content = models.TextField(

verbose\_name="Содержание"

)

timestamp = models.DateTimeField(

auto\_now\_add=True,

verbose\_name="Дата и время отправки"

)

group = models.ForeignKey(Group,

on\_delete=models.CASCADE,

null=True,

blank=True,

related\_name='messages',

verbose\_name="Группа"

)

post = models.ForeignKey(Post,

on\_delete=models.CASCADE,

null=True,

blank=True,

related\_name='comments',

verbose\_name="Пост"

)

likes = models.IntegerField(

default=0

)

like\_list = GenericRelation(

'Like'

)

class Meta:

verbose\_name = "Сообщение"

verbose\_name\_plural = "Сообщения"

ordering = ['timestamp']

def \_\_str\_\_(self):

if self.group:

return f"Сообщение от {self.author.username} в {self.group.name}"

elif self.post:

return f"Комментарий от {self.author.username} к посту {self.post.content}"

def save(self, \*args, \*\*kwargs): # Проверяем, что сообщение привязано либо к группе, либо к посту, но не к обоим.

if self.group and self.post:

raise ValueError("Сообщение должно быть связано либо с группой, либо с постом, но не с обоими.")

if not self.group and not self.post:

raise ValueError("Сообщение должно быть связано либо с группой, либо с постом.")

super().save(\*args, \*\*kwargs)

# ---------------- Лайки ----------------

class Like(models.Model):

user = models.ForeignKey(

User,

on\_delete=models.CASCADE,

related\_name='likes',

verbose\_name="Пользователь"

)

content\_type = models.ForeignKey(

ContentType,

on\_delete=models.CASCADE

)

object\_id = models.PositiveIntegerField(

)

content\_object = GenericForeignKey(

'content\_type',

'object\_id'

)

timestamp = models.DateTimeField(

auto\_now\_add=True,

verbose\_name="Дата и время лайка"

)

class Meta:

verbose\_name = "Лайк"

verbose\_name\_plural = "Лайки"

unique\_together = ('user', 'content\_type', 'object\_id')

def \_\_str\_\_(self):

return f"Лайк от {self.user.username} для {self.content\_type.model} с ID {self.object\_id}"

А.2 Программная реализация сериализаторов пользователя

from rest\_framework import serializers

from django.contrib.auth.models import User

from django.core.exceptions import ObjectDoesNotExist

from rest\_framework.exceptions import PermissionDenied, ValidationError

class UserBaseSerializer(serializers.ModelSerializer):

class Meta:

model = User

fields = ['id', 'username', 'email']

extra\_kwargs = {

'email': {'required': True}

}

class UserReadSerializer(UserBaseSerializer):

class Meta(UserBaseSerializer.Meta):

read\_only\_fields = ('id', 'username', 'email')

class UserCreateSerializer(UserBaseSerializer):

class Meta(UserBaseSerializer.Meta):

fields = UserBaseSerializer.Meta.fields + ['password']

extra\_kwargs = {

'password': {'write\_only': True},

'id': {'read\_only': True}

}

def create(self, validated\_data):

user = User.objects.create\_user(

username=validated\_data['username'],

email=validated\_data['email'],

password=validated\_data['password']

)

return user

class UserUpdateSerializer(UserBaseSerializer):

class Meta(UserBaseSerializer.Meta):

extra\_kwargs = {

\*\*UserBaseSerializer.Meta.extra\_kwargs,

'username': {'required': False},

'email': {'required': False}

}

def update(self, instance, validated\_data):

if instance != self.context['request'].user and not self.context['request'].user.is\_superuser:

raise PermissionDenied("Вы можете редактировать только свои данные")

return super().update(instance, validated\_data)

class UserDeleteSerializer(UserBaseSerializer):

class Meta(UserBaseSerializer.Meta):

fields = []

def delete(self):

user = self.instance

if hasattr(user, 'is\_active'):

user.is\_active = False

user.save()

else:

user.delete()

А.3 Программная реализация представлений пользователя

from django.db.models import Q

from django.http import JsonResponse

from django.shortcuts import get\_object\_or\_404

from django.contrib.auth.models import User

from django\_filters.rest\_framework import DjangoFilterBackend

from django.contrib.contenttypes.models import ContentType

from rest\_framework.exceptions import PermissionDenied

from rest\_framework import status, generics, permissions

from rest\_framework.views import APIView

from rest\_framework.response import Response

from rest\_framework.pagination import PageNumberPagination

from rest\_framework.permissions import IsAuthenticated

from source.api.models import User, Profile, Post, Group, Message, Like

from source.api.serializers.auth\_serializers import ChangePasswordSerializer

from source.api.serializers.user\_serializers import (

UserReadSerializer,

UserCreateSerializer,

UserUpdateSerializer,

UserDeleteSerializer

)

# ---------------- Пользователь ----------------

class IsOwnerOrAdminUser(permissions.BasePermission):

def has\_object\_permission(self, request, view, obj):

return request.user == obj

def user\_delete\_fields(data):

data.pop('id', None)

data.pop('email', None)

return data

class UserRetrieveView(generics.RetrieveAPIView):

queryset = User.objects.all()

permission\_classes = [IsAuthenticated]

serializer\_class = UserReadSerializer

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

instance = self.get\_object()

if request.user == instance or request.user.is\_superuser:

return super().retrieve(request, \*args, \*\*kwargs)

serializer = self.get\_serializer(instance)

data = user\_delete\_fields(serializer.data)

return Response(data)

"""

sensitive\_fields = ['phone', 'last\_login', 'date\_joined']

for field in sensitive\_fields:

filtered\_data.pop(field, None)

"""

class UserRetrieveAllView(generics.ListAPIView):

queryset = User.objects.all()

permission\_classes = [IsAuthenticated]

pagination\_class = PageNumberPagination

serializer\_class = UserReadSerializer

filter\_backends = [DjangoFilterBackend]

filterset\_fields = ['username']

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

queryset = self.filter\_queryset(self.get\_queryset())

page = self.paginate\_queryset(queryset)

if page is not None:

serializer = self.get\_serializer(page, many=True)

data = self.process\_data(serializer.data, request.user)

return self.get\_paginated\_response(data)

serializer = self.get\_serializer(queryset, many=True)

data = self.process\_data(serializer.data, request.user)

return Response(data)

def process\_data(self, data, current\_user):

result = []

for user\_data in data:

if user\_data['id'] == current\_user.id or current\_user.is\_superuser:

result.append(user\_data)

else:

filtered\_data = user\_delete\_fields(user\_data.copy())

result.append(filtered\_data)

return result

class UserCreateView(generics.CreateAPIView):

queryset = User.objects.all()

serializer\_class = UserCreateSerializer

class UserUpdateView(generics.UpdateAPIView):

queryset = User.objects.all()

permission\_classes = [IsAuthenticated]

serializer\_class = UserUpdateSerializer

def perform\_update(self, serializer):

instance = serializer.instance

if not (self.request.user == instance or self.request.user.is\_superuser):

raise PermissionDenied("Вы не имеете прав для редактирования этого аккаунта")

serializer.save()

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

instance = self.get\_object()

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

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

self.perform\_update(serializer)

return Response(

{"detail": "Данные успешно обновлены"},

status=status.HTTP\_200\_OK

)

class UserDestroyView(generics.DestroyAPIView):

queryset = User.objects.all()

serializer\_class = UserDeleteSerializer

permission\_classes = [IsAuthenticated]

def perform\_destroy(self, instance):

if not (self.request.user == instance or self.request.user.is\_superuser):

raise PermissionDenied("Вы не имеете прав для удаления этого аккаунта")

if hasattr(instance, 'is\_active'):

instance.is\_active = False

instance.save()

else:

instance.delete()

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

instance = self.get\_object()

self.perform\_destroy(instance)

return Response(

{"detail": "Аккаунт успешно удален"},

status=status.HTTP\_204\_NO\_CONTENT

)

# Эндпонит для вывода групп конкретного юзера

class UserGroupsRetrieveView(generics.ListAPIView):

serializer\_class = GroupNameOnlyReadSerializer

def get\_queryset(self):

user\_id = self.kwargs['user\_id']

return Group.objects.filter(members\_\_id=user\_id)

# Эндпонит для вывода постов конкретного юзера

class UserPostsRetrieveView(generics.ListAPIView):

serializer\_class = PostContentOnlyReadSerializer

def get\_queryset(self):

user\_id = self.kwargs['user\_id']

return Post.objects.filter(author\_\_id=user\_id)

ПРИЛОЖЕНИЕ Б  
Фрагменты кода frontend части

Б.1 base.html

{% load static %}

<!DOCTYPE html>

<html lang="ru">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>{% block title %}Meetly{% endblock %}</title>

    <link rel="stylesheet" href='{% static "css/style.css" %}'>

    {% block extra\_head %}{% endblock %}

</head>

<body>

    <nav class="main-nav">

        <div class="nav-container">

            <div class="nav-links" id="nav-links">

                <a href="{% url 'index\_page' %}">Главная</a>

            </div>

            <div class="nav-auth" id="nav-auth">

                <a href="{% url 'login\_page' %}">Войти</a>

                <a href="{% url 'register\_page' %}" class="register-btn">Регистрация</a>

            </div>

        </div>

    </nav>

    {% block content %}{% endblock %}

    <script>

        document.addEventListener('DOMContentLoaded', function() {

            const accessToken = localStorage.getItem('access');

            if (accessToken) {

                checkAuthStatus();

            }

        });

        async function checkAuthStatus() {

            const accessToken = localStorage.getItem('access');

            const storedUsername = localStorage.getItem('username');

            const usernameDisplay = document.getElementById('username-display');

            if (usernameDisplay && storedUsername) {

                usernameDisplay.textContent = storedUsername;

            }

            if (!accessToken) {

                showNotAuth();

                return;

            }

            try {

                const response = await fetch('{% url "current\_user" %}', {

                    headers: {

                        'Authorization': 'Bearer ' + accessToken,

                        'Content-Type': 'application/json'

                    }

                });

                if (response.status === 401) {

                    await tryRefreshToken();

                    return;

                }

                if (!response.ok) {

                    throw new Error('Ошибка проверки авторизации');

                }

                const data = await response.json();

                updateAuthUI(data);

            } catch (error) {

                console.error('Ошибка проверки авторизации:', error);

                showNotAuth();

            }

        }

        function updateAuthUI(userData) {

            const usernameDisplay = document.getElementById('username-display');

            const navAuth = document.getElementById('nav-auth');

            const navLinks = document.getElementById('nav-links');

            localStorage.setItem('username', userData.username);

            if (usernameDisplay) {

                usernameDisplay.textContent = userData.username;

            }

            if (navLinks) {

                navLinks.innerHTML = `

                    <a href="{% url 'index\_page' %}">Главная</a>

                    <a href="{% url 'friends\_list' %}">Друзья</a>

                    <a href="{% url 'groups\_list' %}">Группы</a>

                `;

            }

            if (navAuth) {

                navAuth.innerHTML = `

                    <a href="#" onclick="showProfile()" data-profile-id="{{ request.user.profile.id }}">Профиль</a>

                    <a href="{% url 'change\_password\_page' %}" style="margin-right: 10px;">Сменить пароль</a>

                    <button class="logout-btn" onclick="logout()">Выход</button>

                `;

            }

        }

        function showNotAuth() {

            const usernameDisplay = document.getElementById('username-display');

            const navAuth = document.getElementById('nav-auth');

            const navLinks = document.getElementById('nav-links');

            usernameDisplay.textContent = 'Гость';

            usernameDisplay.classList.remove('loading');

            navLinks.innerHTML = `

                <a href="{% url 'index\_page' %}">Главная</a>

            `;

            navAuth.innerHTML = `

                <a href="{% url 'login\_page' %}">Войти</a>

                <a href="{% url 'register\_page' %}" class="register-btn">Регистрация</a>

            `;

        }

        function showProfile() {

            const button = document.querySelector('[onclick="showProfile()"]');

            const profileId = button.dataset.profileId;

            if (profileId) {

                window.location.href = `/page/profiles/${profileId}/`;

            } else {

                alert('Ошибка: не удалось получить ID профиля');

            }

        }

        function logout() {

            const refresh = localStorage.getItem("refresh")

            const access = localStorage.getItem("access");

            fetch("/auth/logout/", {

                method: "POST",

                headers: {

                    "Content-Type": "application/json",

                    "Authorization": "Bearer " + access,

                },

                body: JSON.stringify({

                    refresh: refresh

                })

            })

            .then(response => {

                if (response.ok) {

                    localStorage.removeItem("access");

                    localStorage.removeItem("refresh");

                    localStorage.removeItem("username");

                    window.location.href = "{% url 'login\_page' %}?logout=1&nocache=" + new Date().getTime();

                } else {

                    return response.json().then(data => {

                        alert(data.error || "Ошибка выхода");

                    });

                }

            })

            .catch(error => {

                console.error("Logout error:", error);

                alert("Произошла ошибка");

            });

        }

        async function tryRefreshToken() {

            const refreshToken = localStorage.getItem('refresh');

            if (!refreshToken) {

                showNotAuth();

                return false;

            }

            try {

                const response = await fetch('/auth/token/refresh/', {

                    method: 'POST',

                    headers: {

                        'Content-Type': 'application/json'

                    },

                    body: JSON.stringify({ refresh: refreshToken })

                });

                if (!response.ok) {

                    throw new Error('Не удалось обновить токен');

                }

                const data = await response.json();

                localStorage.setItem('access', data.access);

                return true;

            } catch (error) {

                console.error('Ошибка обновления токена:', error);

                localStorage.removeItem('access');

                localStorage.removeItem('refresh');

                showNotAuth();

                return false;

            }

        }

    </script>

    {% block scripts %}{% endblock %}

</body>

</html>

Б.2 index.html

{% extends 'base.html' %}

{% block content %}

<style>

    .index-page {

        display: flex;

        flex-direction: column;

        min-height: 100vh;

    }

    .index-page header h1 {

        font-size: 40px;

        text-align: center;

        color: #fff;

        margin: 40px 0 20px;

    }

    .index-page nav {

        background-color: #fff;

        padding: 10px 20px;

        border-bottom: 1px solid #ddd;

        display: flex;

        justify-content: space-between;

        align-items: center;

    }

    .index-page nav a {

        margin-right: 30px;

        text-decoration: none;

        color: #4285B4;

        font-size: 16px;

    }

    .index-page nav a:hover {

        color:rgb(0, 0, 0);

    }

    .index-page .container {

        display: flex;

        gap: 20px;

        padding: 30px 20px;

        flex-grow: 1;

    }

    .index-page .sidebar {

        width: 25%;

        background-color: #f7f7f7;

        border-radius: 8px;

        padding: 20px;

    }

    .index-page .sidebar h3 {

        font-size: 30px;

        text-align: center;

        margin-top: 15px;

    }

    .index-page .sidebar p {

        font-size: 20px;

        color: #555;

    }

    .index-page .feed {

        width: 75%;

    }

    .index-page .post

    {

        width: 66%;

        margin: 20px;

        background-color: white;

        padding: 15px;

        border-radius: 5px;

        box-shadow: 0 1px 3px rgba(0,0,0,0.1);

        position: relative;

        display: flex;

        flex-direction: column;

    }

    .logout-btn {

        background-color: #f44336;

        color: white;

        border: none;

        padding: 8px 12px;

        cursor: pointer;

        border-radius: 4px;

        font-size: 14px;

    }

    .logout-btn:hover {

        background-color:rgb(151, 46, 38);

    }

    .register-btn {

        background-color: #4CAF50;

        color: white;

        padding: 6px 12px;

        border-radius: 4px;

        font-size: 14px;

        margin-left: 10px;

    }

    .like-btn

    {

        background-color: #4267B2;

        color: white;

        border: none;

        padding: 8px 12px;

        cursor: pointer;

        border-radius: 4px;

    }

    .like-btn.liked

    {

        background-color: #f44336;

    }

    .like-count

    {

        margin-left: 10px;

    }

    .post-header {

        display: flex;

        justify-content: space-between;

        align-items: flex-start;

    }

    .post-author {

        font-weight: bold;

    }

    .post-date {

        font-size: 12px;

        color: #888;

    }

    .post-footer {

        display: flex;

        justify-content: space-between;

        align-items: center;

        margin-top: 15px;

    }

    .post-actions {

        display: flex;

        gap: 10px;

    }

    .post-body {

        position: relative;

    }

    .comment-btn {

        background-color: #999;

        color: white;

        border: none;

        padding: 8px 12px;

        cursor: pointer;

        border-radius: 4px;

    }

    .comment-btn:hover {

        background-color: #666;

    }

    .comment-section {

        display: none;

    }

    .comment-section.visible {

        display: block;

    }

    .create-post-btn {

        background-color: #3498db;

        color: white;

        padding: 10px 20px;

        border: none;

        border-radius: 5px;

        font-size: 16px;

        cursor: pointer;

        margin: 20px auto;

        display: block;

    }

    .modal-overlay {

        position: fixed;

        top: 0;

        left: 0;

        width: 100%;

        height: 100%;

        background-color: rgba(0, 0, 0, 0.5); /\* затемнение \*/

        display: flex;

        justify-content: center;

        align-items: center;

        z-index: 1000;

    }

    .modal {

        background-color: white;

        padding: 20px;

        width: 400px;

        border-radius: 8px;

        position: relative;

        box-shadow: 0 4px 12px rgba(0,0,0,0.3);

        animation: fadeIn 0.3s ease-in-out;

    }

    .modal-header {

        position: relative;

        text-align: center;

        margin-bottom: 10px;

    }

    .modal-header span {

        display: block;

        width:100%;

        font-size: 18px;

        font-weight: bold;

    }

    .close-btn {

        position: absolute;

        right: 0;

        top: 0;

    }

    .modal textarea {

        width: 95%;

        max-width: 700px;

        margin: 0 auto 10px auto;

        padding: 10px;

        resize: none;

        display: block;

    }

    .send-btn {

        background-color: white;

        color: #007BFF; /\* Синий цвет текста \*/

        border: 2px solid #007BFF;

        padding: 10px 20px;

        border-radius: 5px;

        cursor: pointer;

        font-weight: bold;

        transition: background-color 0.3s, color 0.3s;

        display: block;       /\* Превращает кнопку в блочный элемент \*/

        margin: 0 auto;

        margin-top: 20px;

    }

    .send-btn:hover {

        background-color: #007BFF;

        color: white;

    }

    @keyframes fadeIn {

        from { opacity: 0; transform: scale(0.95); }

        to { opacity: 1; transform: scale(1); }

    }

    .delete-btn {

        position: absolute;

        bottom: 10px;

        right: 10px;

        background-color: #e74c3c;

        color: white;

        border: none;

        padding: 6px 12px;

        border-radius: 4px;

        cursor: pointer;

        font-size: 14px;

    }

    .delete-btn:hover {

        background-color: #c0392b;

    }

</style>

<div class="index-page">

    <div class="create-post-container">

        <button class="create-post-btn" onclick="openModal()">Создать пост</button>

    </div>

    <div id="modal-overlay" class="modal-overlay" style="display: none;">

        <div class="modal">

            <div class="modal-header">

                <span>Создание поста</span>

                <button class="close-btn" onclick="closeModal()">✖</button>

            </div>

            <textarea id="post-content" rows="5" placeholder="Напишите пост..."></textarea>

            <button class="send-btn" onclick="submitPost()">Отправить</button>

        </div>

    </div>

    <div class="container">

        <aside class="sidebar">

            <h3>Мой профиль</h3>

            <p id="username-display" class="loading">Загрузка...</p>

        </aside>

        <div class="feed" id="post-feed">

            <p class="loading">Загрузка постов...</p>

        </div>

    </div>

</div>

<script>

    document.addEventListener('DOMContentLoaded', function() {

        const accessToken = localStorage.getItem('access');

        if (accessToken) {

            setTimeout(() => {

                loadPosts();

            }, 300);

        } else {

            loadPosts();

        }

    });

    async function loadPosts() {

        const accessToken = localStorage.getItem('access');

        const postFeed = document.getElementById('post-feed');

        postFeed.innerHTML = '<p class="loading">Загрузка постов...</p>';

        try {

            const response = await fetch('/api/posts/', {

                headers: {

                    'Authorization': 'Bearer ' + accessToken,

                    'Content-Type': 'application/json'

                }

            });

            if (!response.ok) {

                throw new Error('Ошибка загрузки постов');

            }

            const data = await response.json();

            const posts = data.results || data;

            if (posts.length === 0) {

                postFeed.innerHTML = '<p>Постов нет.</p>';

                return;

            }

            postFeed.innerHTML = '';

            posts.forEach(post => {

                const postDiv = document.createElement('div');

                postDiv.className = 'post';

                postDiv.innerHTML = `

                    <div class="post-body">

                        <div class="post-header">

                            <div class="post-author">${post.author.username || 'Автор неизвестен'}</div>

                            <div class="post-date">${new Date(post.published).toLocaleString()}</div>

                        </div>

                        <div class="post-content">

                            <p>${post.content}</p>

                        </div>

                        <div class="post-footer">

                            <div class="post-actions">

                                <button class="like-btn ${post.user\_liked ? 'liked' : ''}" data-post-id="${post.id}">

                                    ❤️ ${post.likes || 0}

                                </button>

                                <button class="comment-btn">💬 Комментарии</button>

                            </div>

                            ${post.author.username === localStorage.getItem("username") ? `

                                <button class="delete-btn" onclick="deletePost(${post.id})">✖</button>

                            ` : ''}

                        </div>

                    </div>

                    <div class="comment-section" id="comments-${post.id}" style="margin-top: 10px;"></div>

                `;

            postDiv.querySelector('.comment-btn').addEventListener('click', () => {

                openComments(post.id);

            });

            postDiv.querySelector('.like-btn').addEventListener('click', (e) => {

                toggleLike(post.id, e.target);

            });

            postFeed.appendChild(postDiv);

            });

        } catch (error) {

            console.error('Ошибка при загрузке постов:', error);

            postFeed.innerHTML = '<p>Ошибка загрузки постов.</p>';

        }

    }

    async function toggleLike(postId, button) {

        const accessToken = localStorage.getItem('access');

        try {

            const response = await fetch(`/api/posts/${postId}/like/`, {

                method: 'POST',

                headers: {

                    'Authorization': 'Bearer ' + accessToken,

                    'Content-Type': 'application/json'

                }

            });

            if (!response.ok) {

                throw new Error('Ошибка при обновлении лайка');

            }

            const data = await response.json();

            button.classList.toggle('liked', data.liked);

            button.innerHTML = `❤️ ${data.likes}`;

        } catch (error) {

            console.error('Ошибка при обработке лайка:', error);

        }

    }

    async function openComments(postId) {

        const commentsContainer = document.getElementById(`comments-${postId}`);

        if (commentsContainer.classList.contains('visible')) {

            commentsContainer.classList.remove('visible');

            return;

        }

        commentsContainer.classList.add('visible');

        if (commentsContainer.dataset.loaded === 'true') {

            return;

        }

        const accessToken = localStorage.getItem('access');

        commentsContainer.innerHTML = '<p class="loading">Загрузка комментариев...</p>';

        try {

            const postResponse = await fetch(`/api/posts/${postId}/`, {

                method: 'GET',

                headers: {

                    'Authorization': 'Bearer ' + accessToken,

                    'Content-Type': 'application/json'

                }

            });

            if (!postResponse.ok) throw new Error('Ошибка при получении поста');

            const postData = await postResponse.json();

            const commentIds = postData.comments;

            if (!commentIds || commentIds.length === 0) {

                commentsContainer.innerHTML = '<p>Комментариев нет.</p>';

                commentsContainer.innerHTML += getCommentFormHtml(postId);

                commentsContainer.dataset.loaded = 'true';

                return;

            }

            commentsContainer.innerHTML = '<h4>Комментарии:</h4>';

            for (const commentId of commentIds) {

                try {

                    const commentResponse = await fetch(`/api/messages/${commentId}/`, {

                        method: 'GET',

                        headers: {

                            'Authorization': 'Bearer ' + accessToken,

                            'Content-Type': 'application/json'

                        }

                    });

                    if (!commentResponse.ok) continue;

                    const comment = await commentResponse.json();

                    const authorName = comment.author?.username || 'Аноним';

                    const commentDiv = document.createElement('div');

                    commentDiv.className = 'comment';

                    commentDiv.innerHTML = `<strong>${authorName}:</strong><p>${comment.content}</p>`;

                    commentsContainer.appendChild(commentDiv);

                } catch (err) {

                    console.error(`Ошибка загрузки комментария ${commentId}:`, err);

                }

            }

            commentsContainer.innerHTML += getCommentFormHtml(postId);

            commentsContainer.dataset.loaded = 'true';

        } catch (error) {

            console.error('Ошибка загрузки комментариев:', error);

            commentsContainer.innerHTML = '<p>Не удалось загрузить комментарии.</p>';

        }

    }

    function getCommentFormHtml(postId) {

        return `

            <div class="comment-input-wrapper" style="margin-top: 15px;">

                <textarea id="new-comment-${postId}" rows="3" placeholder="Напишите комментарий..." style="width: 100%; max-width:673px"></textarea>

                <div style = "text-align:center; width:100%">

                    <button

                        class="send-btn"

                        onclick="sendComment(${postId})"

                        style="margin-top: 5px;"

                    >

                    Отправить

                    </button>

                <div>

            </div>

        `;

    }

    async function sendComment(postId) {

        const accessToken = localStorage.getItem('access');

        const textarea = document.getElementById(`new-comment-${postId}`);

        const content = textarea.value.trim();

        if (!content)

        {

            alert('Пожалуйста, введите текст комментария');

            return;

        }

        try

        {

            const userResp = await fetch('{% url "current\_user" %}', {

                headers: {

                    'Authorization': 'Bearer ' + accessToken,

                    'Content-Type': 'application/json'

                }

            });

            if (!userResp.ok) {

                console.error("Failed to fetch current user:", userResp.status, userResp.statusText);

                throw new Error('Не удалось получить текущего пользователя');

            }

            const currentUser = await userResp.json();

            console.log("Current user:", currentUser);

            const body = {

                content: content,

                timestamp: new Date().toISOString(),

                author: {

                    id: currentUser.id,

                    username: currentUser.username,

                    email: currentUser.email

                },

                post: postId

            };

            const response = await fetch('/api/messages/create/', {

                method: 'POST',

                headers: {

                    'Authorization': 'Bearer ' + accessToken,

                    'Content-Type': 'application/json'

                },

                body: JSON.stringify(body)

            });

            if (!response.ok) {

                console.error("Error sending comment:", response.status, response.statusText);

                throw new Error('Ошибка при отправке комментария');

            }

            textarea.value = '';

            const commentsContainer = document.getElementById(`comments-${postId}`);

            if (commentsContainer) {

                commentsContainer.dataset.loaded = 'false';

            }

            await openComments(postId);

        } catch (error) {

            console.error("General error:", error);

            alert('Не удалось отправить комментарий');

        }

    }

    function openModal() {

        document.getElementById('modal-overlay').style.display = 'flex';

    }

    function closeModal() {

        document.getElementById('modal-overlay').style.display = 'none';

        document.getElementById('post-content').value = '';

    }

    async function submitPost() {

        const content = document.getElementById('post-content').value.trim();

        const accessToken = localStorage.getItem('access');

        if (!content) {

            alert('Пожалуйста, введите текст поста');

            return;

        }

        try {

            const response = await fetch('/api/posts/create/', {

                method: 'POST',

                headers: {

                    'Authorization': 'Bearer ' + accessToken,

                    'Content-Type': 'application/json'

                },

                body: JSON.stringify({ content: content })

            });

            if (!response.ok) {

                throw new Error('Ошибка при создании поста');

            }

            closeModal();

            // Обновляем ленту постов

            await loadPosts();

        } catch (error) {

            console.error('Ошибка создания поста:', error);

            alert('Не удалось создать пост');

        }

    }

    async function deletePost(postId) {

        const accessToken = localStorage.getItem('access');

        if (!confirm('Вы уверены, что хотите удалить этот пост?')) return;

        try {

            const response = await fetch(`/api/posts/${postId}/delete/`, {

                method: 'DELETE',

                headers: {

                    'Authorization': 'Bearer ' + accessToken,

                    'Content-Type': 'application/json'

                }

            });

            if (!response.ok) {

                throw new Error('Ошибка при удалении поста');

            }

            // Перезагрузка постов после удаления

            await loadPosts();

        } catch (error) {

            console.error('Ошибка удаления поста:', error);

            alert('Не удалось удалить пост');

        }

    }

</script>

{% endblock %}

Б.3 profile/friends/list.html

{% extends 'base.html' %}

{% block content %}

<style>

    .profiles-container h2 {

        color: #3498db;

        font-size: 35px;

        text-align: center;

        margin-bottom: 20px;

    }

    .friends-stats {

        display: flex;

        align-items: center;

        justify-content: space-between;

        margin-bottom: 25px;

    }

    .friends-stats .badge {

        background-color: #3498db;

        color: white;

        font-size: 18px;

        padding: 6px 10px;

        border-radius: 5px;

        align-items: center;

    }

    .friends-stats a.btn {

        background-color: #2ecc71;

        color: white;

        font-size: 14px;

        padding: 8px 18px;

        border-radius: 5px;

        text-decoration: none;

        transition: background-color 0.3s ease;

    }

    .friends-stats a.btn:hover {

        background-color: rgb(30, 124, 69);

    }

    .friend-card {

        background: #fff;

        border-radius: 10px;

        padding: 15px;

        box-shadow: 0 0 10px rgba(0,0,0,0.1);

        display: flex;

        align-items: center;

        transition: transform 0.2s ease;

    }

    .friend-card:hover {

        transform: translateY(-5px);

        box-shadow: 0 8px 15px rgba(0,0,0,0.15);

    }

    .friend-avatar {

        width: 60px;

        height: 60px;

        border-radius: 50%;

        overflow: hidden;

        margin-right: 15px;

        flex-shrink: 0;

    }

    .friend-avatar img {

        width: 100%;

        height: 100%;

        object-fit: cover;

    }

    .friend-info {

        flex-grow: 1;

    }

    .friend-info h5 {

        margin: 0 0 6px;

        color: #333;

        font-weight: 600;

    }

    .friend-info p {

        margin: 0;

        color: #777;

        font-size: 14px;

    }

    .friend-actions {

        display: flex;

        gap: 8px;

        margin-top: 10px;

    }

    .friend-actions a,

    .friend-actions button {

        font-size: 13px;

        padding: 6px 12px;

        border-radius: 5px;

        cursor: pointer;

        border: 2px solid transparent;

        transition: background-color 0.3s ease, color 0.3s ease, border-color 0.3s ease;

        text-decoration: none;

        user-select: none;

    }

    .friend-actions a.btn-outline-primary {

        background-color: transparent;

        color: #3498db;

        border-color: #3498db;

    }

    .friend-actions a.btn-outline-primary:hover {

        background-color: #3498db;

        color: white;

        border-color: #2980b9;

    }

    .friend-actions button.btn-outline-danger {

        background-color: transparent;

        color: #e74c3c;

        border-color: #e74c3c;

    }

    .friend-actions button.btn-outline-danger:hover {

        background-color: #e74c3c;

        color: white;

        border-color: #c0392b;

    }

    .empty-friends {

        text-align: center;

        padding: 50px 20px;

        background: rgb(44, 131, 218);

        border-radius: 10px;

        color: white;

    }

    .empty-friends i {

        font-size: 60px;

        color: rgb(47, 125, 193);

        margin-bottom: 20px;

    }

</style>

<div class="profiles-container">

    {% csrf\_token %}

    <h2>Мои подписки</h2>

    <button onclick="history.back()" class="btn-back">

        ← Назад

    </button>

    {% if error %}

        <div class="alert alert-danger">{{ error }}</div>

    {% endif %}

    <div class="friends-stats">

        <span class="badge">У вас {{ friends\_count }} подписок</span>

        <a href="{% url 'add\_friends' %}" class="btn">Подписаться</a>

    </div>

    {% if friends %}

        <div class="profile-grid">

            {% for friend in friends %}

                <div class="friend-card">

                    <div class="friend-avatar">

                        <img src="{{ friend.avatar|default:'/media/profile\_pictures/tankist.png' }}" alt="{{ friend.username }}">

                    </div>

                    <div class="friend-info">

                        <h5>{{ friend.full\_name|default:friend.username }}</h5>

                        <p>@{{ friend.username }}</p>

                        <div class="friend-actions">

                            <a href="{% url 'profile\_page' friend.profile.id %}" class="btn-outline-primary">Профиль</a>

                            <button class="btn-outline-danger remove-friend"

                                    data-profileid="{{ request.user.profile.id }}"

                                    data-friendid="{{ friend.profile.id }} "

                                    data-friendname="{{ friend.profile.user.username }} ">

                                Удалить

                            </button>

                        </div>

                    </div>

                </div>

            {% endfor %}

        </div>

    {% else %}

        <div class="empty-friends">

            <i class="bi bi-people-fill"></i>

            <p>У вас пока нет подписок</p>

        </div>

    {% endif %}

</div>

<script>

    window.addEventListener("pageshow", function (event) {

        if (event.persisted || performance.getEntriesByType("navigation")[0].type === "back\_forward") {

            location.reload();

        }

    });

    document.querySelectorAll('.remove-friend').forEach(btn => {

        btn.addEventListener('click', function() {

            const profile\_id = parseInt(this.dataset.profileid);

            const friend\_id = parseInt(this.dataset.friendid);

            const friend\_name = this.dataset.friendname;

            const csrfToken = document.querySelector('[name=csrfmiddlewaretoken]').value;

            if (confirm(`Удалить ${friend\_name} из друзей?`)) {

                fetch(`/api/profiles/${profile\_id}/friends/remove/`, {

                    method: 'POST',

                    headers: {

                        'X-CSRFToken': csrfToken,

                        'Content-Type': 'application/json',

                        'Authorization': 'Bearer ' + localStorage.getItem('access')

                    },

                    body: JSON.stringify({

                        friends\_id: [friend\_id]

                    })

                })

                .then(response => response.json())

                .then(data => {

                    if (data.status) {

                        location.reload();

                    }

                    else {

                        alert("Произошла ошибка при удалении");

                    }

                })

                .catch(error => {

                    console.error("Logout error:", error);

                    alert("Произошла ошибка при удалении");

                });

            }

        });

    });

</script>

{% endblock %}

Б.4 groups/detail.html

{% extends "base.html" %}

{% block content %}

<style>

    .group-container {

        max-width: 90%;

        max-height: 1300px;

        margin: 40px auto;

        padding: 50px;

        background-color: #fff;

        border-radius: 10px;

        box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);

    }

    .row {

        display: flex;

        gap: 30px;

        margin: 0 -15px;

        align-items: flex-start;

    }

    .col-md-8 {

        flex: 0 0 65%;

        height: 100%;

        background-color:rgb(106, 148, 179);

        border-radius: 8px;

        padding: 0 15px;

    }

    .col-md-4 {

        flex: 0 0 30%;

        height: 100%;

        background-color:rgb(106, 148, 179);

        border-radius: 8px;

        padding: 30px;

        margin-right: 40px

    }

    .col-md-4 h3 {

        margin-top: 20px;

        margin-bottom: 15px;

        color: #34495e;

    }

    h1 {

        font-size: 32px;

        color: #2c3e50;

        margin-bottom: 10px;

    }

    p {

        color: black;

        margin-bottom: 25px;

    }

    .messages-container {

        flex-grow: 1; /\* Растягиваем на всю доступную высоту \*/

        max-height: 750px;

        overflow-y: auto;

        margin-bottom: 30px;

        padding-right: 10px;

    }

    .card {

        background-color: #f8f9fa;

        border: 1px solid #e1e4e8;

        border-radius: 8px;

        margin-bottom: 5px;

    }

    .card-body {

        margin-left: 15px;

        margin-bottom: 5px;

        padding: 0px;

    }

    .card-title {

        font-weight: bold;

        margin-bottom: 5px;

        color: #34495e;

    }

    .card-text {

        margin-bottom: 15px;

    }

    .text-muted {

        font-size: 12px;

        color: #aaa;

    }

    form textarea.form-control {

        width: 100%;

        min-height: 15%;

        resize: vertical;

        border-radius: 5px;

        border: 1px solid #ccc;

        padding: 10px;

        margin-bottom: 10px;

        box-sizing: border-box;

    }

    form .btn {

        background-color: #2ecc71;

        border: none;

        color: white;

        padding: 10px 20px;

        font-size: 14px;

        border-radius: 5px;

        transition: background-color 0.3s ease;

        margin-bottom: 20px;

    }

    form .btn:hover {

        background-color: #27ae60;

    }

    .member-list {

        border-radius: 5px;

        padding: 20px;

        height: 100%;

    }

    .list-group {

        border-radius: 8px;

        overflow: hidden;

    }

    .list-group-item {

        color: #333;

        padding: 10px 15px;

    }

    .list-group-item:not(:last-child) {

        border-bottom: none;

    }

    .member-card {

        background: #fff;

        border-radius: 10px;

        padding: 15px;

        box-shadow: 0 0 10px rgba(0,0,0,0.1);

        display: flex;

        align-items: center;

        transition: transform 0.2s ease;

        margin-bottom: 5px;

    }

    .member-card:hover {

        transform: translateY(-5px);

        box-shadow: 0 8px 15px rgba(0,0,0,0.15);

    }

    .member-avatar {

        width: 60px;

        height: 60px;

        border-radius: 50%;

        overflow: hidden;

        margin-right: 15px;

        flex-shrink: 0;

    }

    .member-avatar img {

        width: 100%;

        height: 100%;

        object-fit: cover;

    }

    .member-info {

        flex-grow: 1;

    }

    .member-info h5 {

        margin: 0 0 6px;

        color: #333;

        font-weight: 600;

    }

    .member-info p {

        margin: 0;

        color: #777;

        font-size: 14px;

    }

    .member-actions {

        display: flex;

        gap: 8px;

        margin-top: 10px;

    }

    .member-actions a,

    .member-actions button {

        font-size: 13px;

        padding: 6px 12px;

        border-radius: 5px;

        cursor: pointer;

        border: 2px solid transparent;

        transition: background-color 0.3s ease, color 0.3s ease, border-color 0.3s ease;

        text-decoration: none;

        user-select: none;

    }

    .member-actions a.btn-outline-primary {

        background-color: transparent;

        color: #3498db;

        border-color: #3498db;

    }

    .member-actions a.btn-outline-primary:hover {

        background-color: #3498db;

        color: white;

        border-color: #2980b9;

    }

    .member-actions .btn-add {

        background-color: #2ecc71;

        color: white;

        font-size: 14px;

        padding: 8px 18px;

        border-radius: 5px;

        transition: background-color 0.3s ease;

        border: none;

    }

    .member-actions .btn-add:hover {

        background-color:rgb(30, 124, 69);

    }

    .member-actions button.btn-outline-danger {

        background-color: transparent;

        color: #e74c3c;

        border-color: #e74c3c;

    }

    .member-actions button.btn-outline-danger:hover {

        background-color: #e74c3c;

        color: white;

        border-color: #c0392b;

    }

    @media (max-width: 992px) {

        .row {

            flex-direction: column;

        }

        .col-md-8,

        .col-md-4 {

            width: 100%;

            flex: 0 0 100%;

        }

    }

</style>

<div class="group-container">

    <button onclick="window.location.href='{% url 'groups\_list' %}'" class="btn-back">

        ← Назад

    </button>

    <div class="row">

        <div class="col-md-8">

            <h1>{{ group.name }}</h1>

            <p>Создатель: {{ group.author.username }}</p>

            {% if request.user in group.members.all and request.user != group.author %}

                <form action="{% url 'leave\_group' group.id %}" method="post" class="mb-3">

                    {% csrf\_token %}

                    <button type="submit" class="btn btn-danger">Выйти из группы</button>

                </form>

            {% endif %}

            <div class="messages-container mb-4">

                {% for message in messages %}

                    <div class="card mb-2">

                        <div class="card-body">

                            <h5 class="card-title">{{ message.author.username }}</h5>

                            <p class="card-text">{{ message.content }}</p>

                            <small class="text-muted">{{ message.timestamp }}</small>

                        </div>

                    </div>

                {% empty %}

                    <p class="text-muted">Сообщений пока нет</p>

                {% endfor %}

            </div>

            <form id="send-message-form">

                {% csrf\_token %}

                <input type="hidden" name="group\_id" value="{{ group.id }}">

                <div class="form-group">

                    <textarea class="form-control" id="content" name="content" placeholder="Напишите сообщение..." required></textarea>

                </div>

                <button type="submit" class="btn">

                    Отправить

                </button>

            </form>

        </div>

        <div class="col-md-4 member-list">

            <div class="members-management">

                <div class="add-members-form mb-4">

                    <h3>Добавить друзей в группу</h3>

                    {% for profile in user.profile.friends.all %}

                        {% if profile.user not in group.members.all %}

                            <div class="member-card">

                                <div class="member-avatar">

                                    <img src="{{ profile.avatar.url|default:'/media/profile\_pictures/kruto.png' }}" alt="Avatar">

                                </div>

                                <div class="member-info">

                                    <h5>{{ profile.full\_name|default:profile.user.username }}</h5>

                                    <div class="member-actions">

                                        <a href="{% url 'profile\_page' profile.id %}" class="btn-outline-primary">Профиль</a>

                                        <button onclick="handleAddMembers({{ group.id }}, {{ profile.user.id }})" class="btn-add">

                                            Добавить в группу

                                        </button>

                                    </div>

                                </div>

                            </div>

                        {% endif %}

                    {% endfor %}

                </div>

                <div class="members-list">

                    <h3>Участники</h3>

                    <div class="member-grid">

                        {% for member in group.members.all %}

                            <div class="member-card">

                                <div class="member-avatar">

                                    <img src="{{ member.profile.avatar|default:'/media/profile\_pictures/kruto.png' }}" alt="{{ member.username }}">

                                </div>

                                <div class="member-info">

                                    <h5>{{ member.full\_name|default:member.username }}</h5>

                                    <div class="member-actions">

                                        {% if member != group.author %}

                                            <a href="{% url 'profile\_page' member.profile.id %}" class="btn-outline-primary">Профиль</a>

                                            {% if user == group.author %}

                                                <button onclick="handleRemoveMembers({{ group.id }}, {{ member.id }})" class="btn-outline-danger">

                                                    Удалить из группы

                                                </button>

                                            {% endif %}

                                        {% endif %}

                                    </div>

                                </div>

                            </div>

                        {% endfor %}

                    </div>

                </div>

            </div>

        </div>

    </div>

</div>

<script>

    document.getElementById('send-message-form').addEventListener('submit', async function(e) {

        e.preventDefault();

        const group\_id = parseInt('{{ group.id }}');

        const content = document.getElementById('content').value;

        const csrfToken = document.querySelector('[name=csrfmiddlewaretoken]').value;

        try {

            const response = await fetch(`/page/groups/${group\_id}/send/`, {

                method: 'POST',

                headers: {

                    'X-CSRFToken': csrfToken,

                    'Authorization': 'Bearer ' + localStorage.getItem('access')

                },

                body: JSON.stringify({

                    content: content

                })

            });

            if (response.ok) {

                location.reload();

            } else {

                const errorData = await response.json();

                alert("Ошибка: " + (errorData.message || "Неизвестная ошибка"));

            }

        } catch (error) {

            console.error('Error:', error);

            alert("Ошибка соединения с сервером");

        }

    });

    async function handleAddMembers(groupId, memberId) {

        try {

            await addGroupMembers(groupId, memberId);

            alert('Участники успешно добавлены!');

            location.reload();

        } catch (error) {

            console.error('Ошибка добавления:', error);

        }

    }

    async function handleRemoveMembers(groupId, memberId) {

        if (!confirm(`Вы уверены, что хотите удалить участника ${memberId}?`)) {

            return;

        }

        try {

            await removeGroupMembers(groupId, memberId);

            alert('Участники успешно удалены!');

            location.reload();

        } catch (error) {

            console.error('Ошибка удаления:', error);

        }

    }

    async function addGroupMembers(groupId, memberId) {

        try {

            const csrfToken = document.querySelector('[name=csrfmiddlewaretoken]').value;

            const response = await fetch(`/api/groups/${groupId}/members/add/`, {

                method: 'POST',

                headers: {

                    'Content-Type': 'application/json',

                    'X-CSRFToken': csrfToken,

                    'Authorization': 'Bearer ' + localStorage.getItem('access')

                },

                body: JSON.stringify({

                    members\_id: [memberId]

                })

            });

            const contentType = response.headers.get('content-type');

            const isJson = contentType && contentType.includes('application/json');

            const responseData = isJson ? await response.json() : null;

            if (!response.ok) {

                const errorMsg = responseData?.detail ||

                                responseData?.message ||

                                `HTTP error! status: ${response.status}`;

                throw new Error(errorMsg);

            }

            console.log('Участники успешно добавлены:', responseData);

            return responseData;

        } catch (error) {

            console.error('Ошибка добавления участников:', error);

            alert(error.message);

            throw error;

        }

    }

    async function removeGroupMembers(groupId, memberId) {

        try {

            const csrfToken = document.querySelector('[name=csrfmiddlewaretoken]').value;

            const response = await fetch(`/api/groups/${groupId}/members/remove/`, {

                method: 'POST',

                headers: {

                    'Content-Type': 'application/json',

                    'X-CSRFToken': csrfToken,

                    'Authorization': 'Bearer ' + localStorage.getItem('access')

                },

                body: JSON.stringify({

                    members\_id: [memberId]

                })

            });

            if (!response.ok) {

                const errorData = await response.json();

                throw new Error(errorData.detail || 'Ошибка при удалении участников');

            }

            const data = await response.json();

            console.log('Успешно удалены:', data);

            return data;

        } catch (error) {

            console.error('Ошибка:', error);

            alert(error.message);

            throw error;

        }

    }

</script>

{% endblock %}

Б.5 profile.html

{% extends 'base.html' %}

{% block content %}

<style>

    body {

        background-color: rgb(71, 115, 180); /\* Синий фон всей страницы \*/

        margin: 0;

        font-family: Arial, sans-serif;

    }

    .profile-wrapper {

        display: flex;

        justify-content: center;

        align-items: flex-start; /\* Выравнивание сверху \*/

        min-height: 100vh;

        padding: 0; /\* Убираем отступ сверху \*/

    }

    .profile-container {

        background-color: white;

        padding: 30px;

        border-radius: 8px;

        box-shadow: 0 4px 12px rgba(0, 0, 0, 0.15);

        max-width: 600px;

        width: 100%;

        text-align: center;

        margin-top: 30px;

    }

    .profile-container form {

        display: inline-block;

        text-align: left;

        margin: auto;

        width: 100%;

    }

    .custom-button {

        background-color: rgb(71, 115, 180); /\* Синий фон \*/

        color: white;

        border: 2px solid rgb(71, 115, 180);

        border-radius: 4px;

        padding: 10px 16px;

        cursor: pointer;

        font-size: 14px;

        transition: background-color 0.3s, border-color 0.3s;

        width: 150px;

        margin: 10px 5px 0 5px; /\* Небольшие отступы только по бокам и сверху \*/

    }

    .custom-button:hover {

        background-color: #0056b3;

        border-color: #0056b3;

    }

    form input,

    form select {

        width: 100%;

        padding: 8px;

        margin-bottom: 12px;

        border: 1px solid #ccc;

        border-radius: 4px;

    }

    h2 {

        text-align: center;

        color: rgb(71, 115, 180);

    }

    #profile-photo {

        display: block;

        margin: 0 auto 20px auto;

    }

    .custom-button.small {

        font-size: 12px;

        padding: 4px 10px;

        margin-left: 10px;

        margin-top: 0;

    }

    .field-section {

        display: flex;

        align-items: center;

        justify-content: space-between;

        margin-bottom: 15px;

    }

    .field-section p {

        margin: 0;

        flex: 1;

    }

    .field-controls {

        margin-left: 10px;

    }

    .field-section .custom-button.small {

        margin: 0;

    }

</style>

<div class="profile-wrapper">

    <div class="profile-container">

        <h2>Профиль пользователя</h2>

        <div class="profile-info" id="profile-view">

            {% if profile.photo %}

                <img id="profile-photo" src="{{ profile.photo.url }}" alt="Фото профиля" style="max-width: 250px; border-radius: 8px;">

            {% else %}

                <p id="profile-photo-placeholder">Фото не загружено.</p>

            {% endif %}

            <!-- Имя пользователя -->

            <div class="field-section" id="username-section">

                <p><strong>Имя пользователя:</strong>

                    <span id="username-text">{{ profile.user.username }}</span>

                    {% if request.user == profile.user %}

                    <button class="custom-button small" onclick="editField('username')">Изменить</button>

                    {% endif %}

                </p>

                <div id="username-edit" style="display:none;">

                    <input type="text" id="username-input" value="{{ profile.user.username }}">

                    <button class="custom-button small" onclick="saveField('username')">Сохранить</button>

                    <button class="custom-button small" onclick="cancelField('username')">Отмена</button>

                </div>

            </div>

            <!-- Email -->

            <div class="field-section" id="email-section">

                <p><strong>Email:</strong>

                    <span id="email-text">{{ profile.user.email }}</span>

                    {% if request.user == profile.user %}

                    <button class="custom-button small" onclick="editField('email')">Изменить</button>

                    {% endif %}

                </p>

                <div id="email-edit" style="display:none;">

                    <input type="email" id="email-input" value="{{ profile.user.email }}">

                    <button class="custom-button small" onclick="saveField('email')">Сохранить</button>

                    <button class="custom-button small" onclick="cancelField('email')">Отмена</button>

                </div>

            </div>

            <!-- Возраст -->

            <div class="field-section" id="age-section">

                <p><strong>Возраст:</strong>

                    <span id="age-text">{{ profile.age|default:"Не указано" }}</span>

                    {% if request.user == profile.user %}

                    <button class="custom-button small" onclick="editField('age')">Изменить</button>

                    {% endif %}

                </p>

                <div id="age-edit" style="display:none;">

                    <input type="number" id="age-input" value="{{ profile.age }}">

                    <button class="custom-button small" onclick="saveField('age')">Сохранить</button>

                    <button class="custom-button small" onclick="cancelField('age')">Отмена</button>

                </div>

            </div>

            <!-- Пол -->

            <div class="field-section" id="gender-section">

                <p><strong>Пол:</strong>

                    <span id="gender-text">

                        {% if profile.gender == "M" %}Мужской{% elif profile.gender == "F" %}Женский{% elif profile.gender == "O" %}Другое{% else %}Не указано{% endif %}

                    </span>

                    {% if request.user == profile.user %}

                    <button class="custom-button small" onclick="editField('gender')">Изменить</button>

                    {% endif %}

                </p>

                <div id="gender-edit" style="display:none;">

                    <select id="gender-input">

                        <option value="">Выбрать</option>

                        <option value="M" {% if profile.gender == "M" %}selected{% endif %}>Мужской</option>

                        <option value="F" {% if profile.gender == "F" %}selected{% endif %}>Женский</option>

                        <option value="O" {% if profile.gender == "O" %}selected{% endif %}>Другое</option>

                    </select>

                    <button class="custom-button small" onclick="saveField('gender')">Сохранить</button>

                    <button class="custom-button small" onclick="cancelField('gender')">Отмена</button>

                </div>

            </div>

            <!-- Дата рождения -->

            <div class="field-section" id="birth\_date-section">

                <p><strong>Дата рождения:</strong>

                    <span id="birth\_date-text">{{ profile.birth\_date|date:"Y-m-d"|default:"Не указано" }}</span>

                    {% if request.user == profile.user %}

                    <button class="custom-button small" onclick="editField('birth\_date')">Изменить</button>

                    {% endif %}

                </p>

                <div id="birth\_date-edit" style="display:none;">

                    <input type="date" id="birth\_date-input" value="{{ profile.birth\_date|date:'Y-m-d' }}">

                    <button class="custom-button small" onclick="saveField('birth\_date')">Сохранить</button>

                    <button class="custom-button small" onclick="cancelField('birth\_date')">Отмена</button>

                </div>

            </div>

            <div class="field-section" id="email-section">

                <p><strong>Дата рождения:</strong>

                    <span id="email-text">{{ profile.user.email }}</span>

                    {% if request.user == profile.user %}

                    <button class="custom-button small" onclick="editField('email')">Изменить</button>

                    {% endif %}

                </p>

                <div id="email-edit" style="display:none;">

                    <input type="email" id="email-input" value="{{ profile.user.email }}">

                    <button class="custom-button small" onclick="saveField('email')">Сохранить</button>

                    <button class="custom-button small" onclick="cancelField('email')">Отмена</button>

                </div>

            </div>

        </div>

        {% if request.user == profile.user %}

        <form id="edit-form" style="display:none;" onsubmit="submitEdit(event)">

            <label>Имя пользователя:</label>

            <input type="text" id="username-input" name="username" value="{{ profile.user.username }}">

            <label>Email:</label>

            <input type="email" id="email-input" name="email" value="{{ profile.user.email }}">

            <label>Возраст:</label>

            <input type="number" id="age-input" name="age" value="{{ profile.age }}">

            <label>Пол:</label>

            <select id="gender-input" name="gender">

                <option value="">Выбрать</option>

                <option value="M" {% if profile.gender == 'M' %}selected{% endif %}>Мужской</option>

                <option value="F" {% if profile.gender == 'F' %}selected{% endif %}>Женский</option>

                <option value="O" {% if profile.gender == 'O' %}selected{% endif %}>Другое</option>

            </select>

            <label>Дата рождения:</label>

            <input type="date" id="birthdate-input" name="birth\_date" value="{{ profile.birth\_date|date:'Y-m-d' }}">

            <div style="text-align: center;">

                <button type="submit" class="custom-button">Сохранить</button>

                <button type="button" onclick="cancelEdit()" class="custom-button">Отмена</button>

            </div>

        </form>

        <button class="custom-button" id="uploadPhotoBtn">Загрузить фото</button>

        <input type="file" id="photoInput" style="display:none" accept="image/\*" />

        {% endif %}

    </div>

</div>

<script>

    function editField(field) {

        document.getElementById(`${field}-text`).style.display = 'none';

        document.querySelector(`#${field}-section .custom-button`).style.display = 'none';

        document.getElementById(`${field}-edit`).style.display = 'block';

    }

    function cancelField(field) {

        document.getElementById(`${field}-text`).style.display = 'inline';

        document.querySelector(`#${field}-section .custom-button`).style.display = 'inline-block';

        document.getElementById(`${field}-edit`).style.display = 'none';

    }

    async function saveField(field) {

        const accessToken = localStorage.getItem('access');

        const profileId = "{{ profile.id }}";

        const url = `/api/profiles/${profileId}/update/`;

        let value;

        if (field === 'gender') {

            value = document.getElementById(`${field}-input`).value || null;

        } else if (field === 'age') {

            value = parseInt(document.getElementById(`${field}-input`).value) || null;

        } else {

            value = document.getElementById(`${field}-input`).value.trim();

        }

        // Подготовим тело запроса

        let body = {};

        if (field === 'username' || field === 'email') {

            body['user'] = {};

            body['user'][field] = value;

        } else {

            body[field] = value;

        }

        try {

            const response = await fetch(url, {

                method: 'PATCH',

                headers: {

                    'Authorization': 'Bearer ' + accessToken,

                    'Content-Type': 'application/json'

                },

                body: JSON.stringify(body)

            });

            if (!response.ok) {

                const error = await response.json();

                alert("Ошибка: " + JSON.stringify(error));

                return;

            }

            alert('Изменения сохранены');

            window.location.reload();

        } catch (err) {

            console.error('Ошибка при обновлении:', err);

            alert('Ошибка обновления поля');

        }

    }

    function enableEdit() {

        document.getElementById('profile-view').style.display = 'none';

        document.getElementById('edit-form').style.display = 'block';

    }

    function cancelEdit() {

        document.getElementById('edit-form').style.display = 'none';

        document.getElementById('profile-view').style.display = 'block';

    }

    async function submitEdit(event) {

        event.preventDefault();

        const username = document.getElementById('username-input').value;

        const email = document.getElementById('email-input').value;

        const age = document.getElementById('age-input').value;

        const gender = document.getElementById('gender-input').value;

        const birthDate = document.getElementById('birthdate-input').value;

        const accessToken = localStorage.getItem('access');

        const profileId = "{{ profile.id }}";

        const url = `/api/profiles/${profileId}/update/`;

        const response = await fetch(url, {

            method: 'PUT',

            headers: {

                'Content-Type': 'application/json',

                'Authorization': 'Bearer ' + accessToken

            },

            body: JSON.stringify({

                user: {

                    username: username,

                    email: email

                },

                age: age ? parseInt(age) : null,

                gender: gender || null,

                birth\_date: birthDate || null

            })

        });

        if (response.ok) {

            alert("Профиль обновлён!");

            window.location.reload();

        } else {

            const data = await response.json();

            alert("Ошибка: " + JSON.stringify(data));

        }

    }

    {% if request.user == profile.user %}

    const uploadBtn = document.getElementById('uploadPhotoBtn');

    const photoInput = document.getElementById('photoInput');

    const profileId = "{{ profile.id }}";

    uploadBtn.addEventListener('click', () => {

        photoInput.click();

    });

    photoInput.addEventListener('change', async () => {

        const file = photoInput.files[0];

        if (!file) return;

        const formData = new FormData();

        formData.append('photo', file);

        const accessToken = localStorage.getItem('access');

        const url = `/api/profiles/${profileId}/update/`;

        try {

            const response = await fetch(url, {

                method: 'PUT',

                headers: {

                    'Authorization': 'Bearer ' + accessToken

                },

                body: formData

            });

            if (!response.ok) {

                const errorData = await response.json();

                alert('Ошибка загрузки фото: ' + JSON.stringify(errorData));

                return;

            }

            const data = await response.json();

            alert('Фото успешно обновлено!');

            const img = document.getElementById('profile-photo');

            const placeholder = document.getElementById('profile-photo-placeholder');

            if (img) {

                img.src = data.photo;

            } else if (placeholder) {

                placeholder.style.display = 'none';

                const newImg = document.createElement('img');

                newImg.id = 'profile-photo';

                newImg.src = data.photo;

                newImg.style.maxWidth = '150px';

                newImg.style.borderRadius = '8px';

                newImg.style.display = 'block';

                newImg.style.margin = '0 auto';

                placeholder.parentNode.insertBefore(newImg, placeholder);

            }

        } catch (err) {

            alert('Ошибка при загрузке фото');

            console.error(err);

        }

    });

    {% endif %}

</script>

{% endblock %}