

Министерство науки и высшего образования Российской Федерации

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

Институт информатики и кибернетики

Кафедра технической кибернетики

Финальный отчёт

Дисциплина: «Технологии сетевого программирования»

Финализация проекта «TravelFam – приложение для планирования семейных
путешествий»

Выполнили: Павлова Мария, Сокол Дарья

Группа: 6303-010302D

Самара 2025

ОСНОВНАЯ ИНФОРМАЦИЯ

Архитектура приложения

Проект представляет собой веб-приложение, реализующее планирование совместных путешествий для семей с функционалом создания поездок, управления местами, оценки достопримечательностей и организации семейных групп.

Архитектура построена по модели "клиент-сервер" и включает следующие ключевые компоненты:

1. Клиентская часть (HTML/CSS):
 - Отвечает за отображение форм регистрации, входа, создания семьи, создание поездки, оставление отзывов.
 - Реализует взаимодействие с сервером через HTML формы с методом POST с JWT-аутентификацией.
2. Серверная часть (Django + DRF + Jinja2):
 - Обработывает все HTTP-запросы.
 - Реализует REST API и HTML-шаблоны.
 - Обеспечивает JWT-аутентификацию управление пользователями, семьями, поездками и отзывами.
3. База данных (PostgreSQL):
 - Хранение данных о пользователях, семьях, путешествиях, местах и отзывах.
4. Docker-контейнеризация:
 - Отдельные сервисы для Django (web), PostgreSQL (db) и PgAdmin (pgadmin), управляемые через docker-compose.yml, работают в одной Docker-сети, обеспечивая их взаимодействие и изоляцию.

Взаимодействие компонентов

- Пользователь регистрируется или входит в систему, получая JWT токен.

- Создаёт семью, приглашает участников или присоединяется к существующей группе.
- Планирует путешествие: позволяет пользователю создать маршрут для определенной семьи, выбирая страну и город, добавляя места из каталога, устанавливая даты и состояние путешествия.
- Оставляет отзывы о местах.
- Все операции выполняются через защищённые API-запросы с токеном в заголовке.

Структура базы данных

Перечислим основные таблицы:

1. User:

- id, full_name, e-mail, login, password, preferences, create_date, is_superuser, is_active, is_staff

2. Family:

- id, name, create_date

3. Family_Member:

- id, user_id, family_id, role

4. Family_requests

- id, family_id, user_id, status, create_date

5. Trip:

- id, name_trip, country, city, start_date, end_date, family_id, status, family_member_id

6. Place:

- id, name_place, category, description, coordinates, cost

7. Trip_Place:

- id, trip_id, place_id

8. Review:

- id, user_id, place_id, mark, text

- POST /api/family/ — создать семью
- POST /api/family/<family_id>/request/ — запрос в семью
- POST /api/family/<family_id>/request/<user_id>/accept/ —

принять запрос в семью

- POST /api/family/<family_id>/request/<user_id>/decline/ —

отклонить запрос в семью

- GET /api/family/<user_id>/ — список семей пользователя
- GET /api/family/<family_id>/requests/ — список запросов в

семью

- DELETE /api/family/<family_id>/member/<pk>/ — удалить

участника из семьи

- GET /api/family/<family_id>/members/ — просмотр членов

семьи пользователя

4. Поездки

- POST /trip/ — создать поездку
- POST /api/trip/<pk>/repeat/ — повторить поездку
- GET /api/trips/<user_id>/ — список поездок пользователя
- DELETE /api/trip/<trip_id>/ — удалить поездку
- POST /api/trip/<pk>/add_place/ — добавить место в поездку
- GET /api/trip/<pk>/places/ — список мест в поездке
- DELETE /api/trip/<pk>/remove_place/ — удалить место из

поездки

5. Места

- POST /api/places/ — создать место
- GET /api/places/ — список мест
- GET /api/places/filter/ — фильтрация мест

6. Отзывы

- GET /api/reviews/ — все отзывы

- POST /api/place/<place_id>/reviews/ — создать отзыв
- GET /api/place/<place_id>/reviews/ — отзывы места
- GET /api/reviews/filter/ — фильтрация отзывов
- GET /api/reviews/<pk>/ — отзыв по id

Существуют эндпоинты, которые создаются автоматически при помощи Django REST Framework (например, DELETE api/family/<family_id>), но мы ими не пользовались, и они не требуются для нашего приложения.

Все API требуют JWT токен (в заголовке Authorization: Bearer)

Стек технологий

- Backend: Django (Python), DRF;
- Frontend: Jinja2, Bootstrap;
- Аутентификация: JWT;
- База данных: PostgreSQL, PgAdmin4;
- Контейнеризация: Docker

Контейнеризация и докеризация

Backend (Django):

- Использует Dockerfile с установкой зависимостей и запуском встроенного сервера Django (runserver).
- При старте автоматически выполняются миграции базы данных.

db (PostgreSQL):

- Использует официальный образ postgres:17.
- Настройки задаются через environment в docker-compose.yml.

docker-compose.yml:

- Управляет порядком запуска сервисов, подключением к общей Docker- сети и общим томом postgres_data для хранения данных базы.

ЛАБОРАТОРНАЯ РАБОТА 0

TravelFam – веб-приложение для создания семейных путешествий.

Цель проекта: создать удобную веб-платформу, которая позволит пользователям создавать семьи или присоединяться к существующей, создавать поездки для всех членов семьи, устанавливать даты и добавлять места в поездке, оставлять отзывы о всех местах, где побывали.

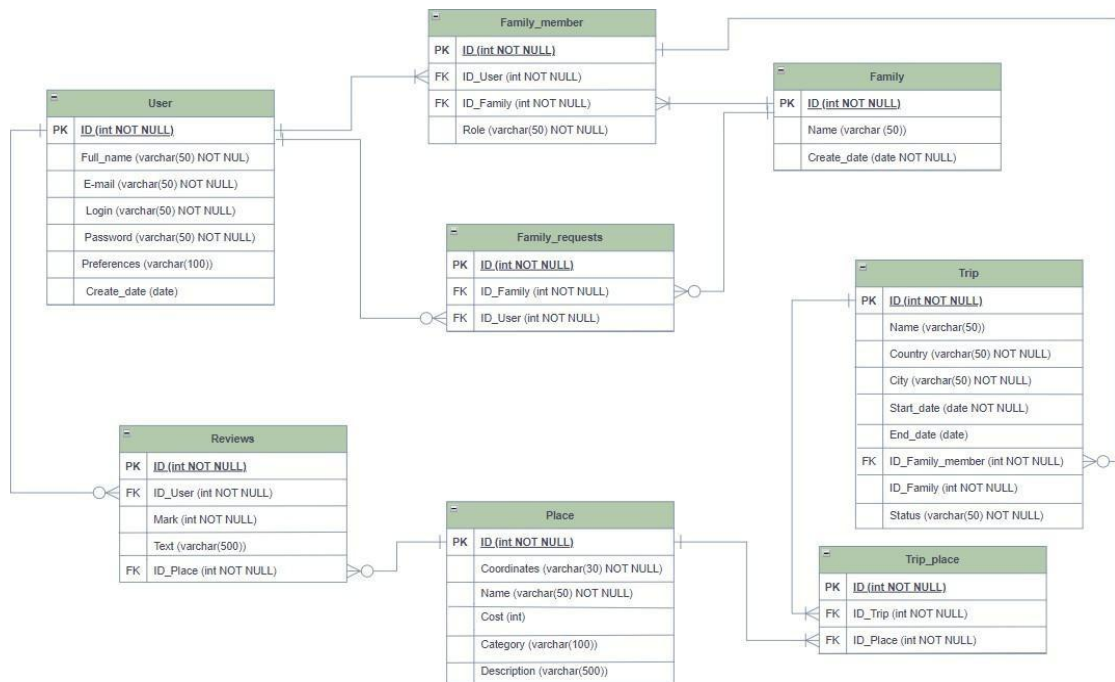


Рисунок 2 – Связь основных таблиц

Описание базы данных

1. User (Пользователи)

- ID: Уникальный идентификатор пользователя (первичный ключ)
- Full_name: Полное имя пользователя (обязательное)
- E-mail: Электронная почта пользователя (обязательное)
- Login: Логин для входа (обязательное)
- Password: Пароль (обязательное)
- Preferences: Предпочтения пользователя (необязательное)

- Create_date: Дата создания аккаунта
2. *Family (Семья)*
- ID: Уникальный идентификатор семьи (первичный ключ)
 - Name: Название семьи (необязательное)
 - Create_date: Дата создания семейной группы (обязательное)
3. *Family_member (Члены семьи)*
- ID: Уникальный идентификатор (первичный ключ)
 - ID_User: Ссылка на пользователя (внешний ключ)
 - ID_Family: Ссылка на семью (внешний ключ)
 - Role: Роль в семье (обязательное)
4. *Family_requests (Запросы в семью)*
- ID: Уникальный идентификатор (первичный ключ)
 - ID_Family: Ссылка на семью (внешний ключ)
 - ID_User: Ссылка на пользователя (внешний ключ)
5. *Place (Места)*
- ID: Уникальный идентификатор места (первичный ключ)
 - Coordinates: Координаты места (обязательное)
 - Name: Название места (обязательное)
 - Cost: Стоимость (целое число)
 - Category: Категория места (необязательное)
 - Description: Описание места (необязательное)
6. *Trip (Поездки)*
- ID: Уникальный идентификатор поездки (первичный ключ)
 - Name: Название поездки (необязательное)
 - Country: Страна назначения (обязательное)
 - City: Город назначения (обязательное)
 - Start_date: Дата начала поездки (обязательное)

- End_date: Дата окончания поездки (необязательное)
 - ID_Family_member: Ссылка на члена семьи, создавшего поездку (внешний ключ)
 - ID_Family: Ссылка на семью (внешний ключ)
 - Status: Статус поездки (обязательное)
7. *Trip_place (Места в поездке)*
- ID: Уникальный идентификатор места в поездке (первичный ключ)
 - ID_Trip: Ссылка на поездку (внешний ключ)
 - ID_Place: Ссылка на место (внешний ключ)
8. *Reviews (Отзывы о местах)*
- ID: Уникальный идентификатор отзыва (первичный ключ)
 - ID_User: Ссылка на пользователя (внешний ключ)
 - Mark: Оценка (обязательное)
 - Text: (необязательно)
 - ID_Place: Ссылка на место (внешний ключ)

Взаимосвязи

1. Один пользователь может быть членом нескольких семей (через Family_member)
2. Одна семья может иметь несколько пользователей (через Family_member)
3. Пользователи могут отправлять запросы на вступление в семьи (через Family_requests)
4. Одна поездка принадлежит одной семье и создается одним членом семьи
5. В одной поездке может быть несколько мест (через Trip_place)

6. Одно место может находиться в нескольких поездках (через Trip_place)

Назначение системы

Система позволяет семьям:

- Создавать семейные группы
- Планировать совместные поездки
- Добавлять места для посещения
- Управлять статусами поездок
- Хранить информацию о точках интереса с их описанием и

стоимостью

ЛАБОРАТОРНАЯ РАБОТА 1

- Развернут PostgreSQL в Docker
 - Настроено подключение к БД через pgAdmin4
 - Разработаны ORM-модели
 - Настроены миграции
 - Настроено хеширование паролей
 - Написаны скрипты для заполнения базы данных тестовыми данными
- Реализован функционал для работы с данными в соответствии с тематикой приложения

Описание моделей (Приложение А)

Модель User (Пользователь)

Кастомная модель пользователя, расширяющая `AbstractBaseUser` и `PermissionsMixin`.

Поля:

- `email` - Email пользователя (уникальный, обязательный)
- `login` - Логин пользователя (уникальный, обязательный)
- `full_name` - Полное имя пользователя (обязательное)
- `preferences` - Предпочтения пользователя (необязательное)
- `create_date` - Дата создания аккаунта (автоматически устанавливается)
- `is_active` - Флаг активности пользователя
- `is_staff` - Флаг персонала

Методы:

- `create_user` - Создает обычного пользователя
- `create_superuser` - Создает суперпользователя

Модель Family (Семья)

Поля:

- name - Название семьи
- create_date - Дата создания (автоматически устанавливается)

Модель FamilyMember (Член семьи)

Связывает пользователей с семьями.

Поля:

- user - Ссылка на пользователя (ForeignKey)
- family - Ссылка на семью (ForeignKey)
- role - Роль в семье (по умолчанию 'member')

Модель FamilyRequests (Запросы в семью)

Управляет запросами на вступление в семью.

Поля:

- family - Ссылка на семью (ForeignKey)
- user - Ссылка на пользователя (ForeignKey)
- create_date - Дата создания запроса
- status - Статус запроса (выбор из: 'в ожидании', 'принят', 'отклонён')
- *Meta:*
- unique_together - Гарантирует уникальность пары family-user

Модель Place (Место)

Хранит информацию о местах для посещения.

Поля:

- coordinates - Координаты места
- name - Название места
- cost - Стоимость посещения

- category - Категория места
- description - Описание места

Модель Reviews (Отзывы)

Хранит отзывы пользователей о местах.

Поля:

- user - Автор отзыва (ForeignKey)
- mark - Оценка места
- text - Текст отзыва
- place - Ссылка на место (ForeignKey)

Модель Trip (Поездка)

Хранит информацию о запланированных поездках.

Поля:

- name - Название поездки
- country - Страна назначения
- city - Город назначения
- start_date - Дата начала
- end_date - Дата окончания (необязательное)
- family_member - Создатель поездки (ForeignKey к

FamilyMember)

- family - Семья, к которой относится поездка (ForeignKey)
- status - Статус поездки
- places - ManyToMany связь с местами через TripPlace

Модель TripPlace (Место в поездке)

Промежуточная модель для связи поездок и мест.

Поля:

- trip - Ссылка на поездку (ForeignKey)
- place - Ссылка на место (ForeignKey)

Особенности реализации:

- Кастомная модель пользователя с email в качестве USERNAME_FIELD
- Все строковые поля имеют ограничения по длине
- Для дат создания используется auto_now_add или timezone.now
- Используются ForeignKey для связей между моделями
- Для связи поездок и мест используется промежуточная модель TripPlace
- Добавлены методы str для удобного отображения объектов
- Для запросов в семью реализованы статусы через choices
- Отзывы имеют собственную модель с оценкой и текстом
- Для связи пользователя и семьи используется промежуточная модель FamilyMember

Разработан набор Django представлений (views) для серверной части веб-приложения. Представления обеспечивают взаимодействие с пользователем через HTML-шаблоны, обрабатывают HTTP-запросы и интегрируются с REST API для управления данными.

Список представлений:

1. Аутентификация и управление профилем:

- register_view: Регистрация нового пользователя.
- login_view: Вход пользователя в систему.
- logout_view: Выход из системы.
- profile_view: Просмотр профиля пользователя.
- change_password_view: Смена пароля.

- `update_profile_view`: Обновление данных профиля.
2. Управление семьями:
- `create_family_view`: Создание новой семейной группы.
 - `families_view`: Просмотр списка семей пользователя и отправка запросов на вступление.
 - `family_members_view`: Управление членами семьи и запросами на вступление.
3. Управление поездками:
- `create_trip_view`: Создание новой поездки.
 - `trips_view`: Просмотр списка поездок пользователя.
 - `trip_details_view`: Просмотр деталей поездки и изменение её статуса.
 - `trip_add_place_view`: Добавление места в поездку.
 - `trip_remove_place_view`: Удаление места из поездки.
 - `repeat_trip_custom_view`: Создание копии существующей поездки.
 - `delete_trip_view`: Удаление поездки.
4. Управление местами и отзывами:
- `places_view`: Просмотр списка мест с возможностью фильтрации по категории.
 - `place_reviews_view`: Просмотр отзывов о конкретном месте.
 - `reviews_view`: Просмотр всех отзывов с фильтрацией по оценке.
 - `create_review_view`: Создание нового отзыва о месте.

Пример представления в приложении Б.

ЛАБОРАТОРНАЯ РАБОТА 2

- Разработаны CRUD-методы для работы с моделями
- Настроены маршруты и обработка запросов
- Для тестирования API использовался Postman (для проверки запросов)
- Описана структура API
- Написаны Serializers и Urls

Структура API

Аутентификация

- POST /api/login/ — вход по email и паролю
- POST /api/register/ — регистрация
- POST /api/logout/ — выход
- POST /api/token/refresh/ — обновление токенов

Пользователь

- PUT /api/user/<user_id>/ — обновление данных пользователя
- GET /api/user/<user_id>/ — просмотр информации о пользователе
- POST /api/user/<user_id>/change-password/ — смена пароля

Семьи

- POST /api/family/ — создать семью
- POST /api/family/<family_id>/request/ — запрос в семью
- POST /api/family/<family_id>/request/<user_id>/accept/ — принять запрос в семью
- POST /api/family/<family_id>/request/<user_id>/decline/ — отклонить запрос в семью
- GET /api/family/<user_id>/ — список семей пользователя

- GET /api/family/<family_id>/requests/ — список запросов в семью
- DELETE /api/family/<family_id>/member/<pk>/ — удалить участника из семьи
- GET /api/family/<family_id>/members/ — просмотр членов семьи пользователя

Поездки

- POST /trip/ — создать поездку
- POST /api/trip/<pk>/repeat/ — повторить поездку
- GET /api/trips/<user_id>/ — список поездок пользователя
- DELETE /api/trip/<trip_id>/ — удалить поездку
- POST /api/trip/<pk>/add_place/ — добавить место в поездку
- GET /api/trip/<pk>/places/ — список мест в поездке
- DELETE /api/trip/<pk>/remove_place/ — удалить место из поездки

Места

- POST /api/places/ — создать место
- GET /api/places/ — список мест
- GET api/places/filter/ — фильтрация мест

Отзывы

- GET /api/reviews/ — все отзывы
- POST /api/place/<place_id>/reviews/ — создать отзыв
- GET /api/place/<place_id>/reviews/ — отзывы места
- GET /api/reviews/filter/ — фильтрация отзывов
- GET /api/reviews/<pk>/ — отзыв по id

Пример Serializer и Urls представлены в приложении В.

ЛАБОРАТОРНАЯ РАБОТА 3

Сделано:

- Регистрация нового пользователя
- Вход в систему и получение JWT-токена
- Проверка валидности токена
- Ограничение доступа к определённым эндпоинтам
- Проверка работы через Postman

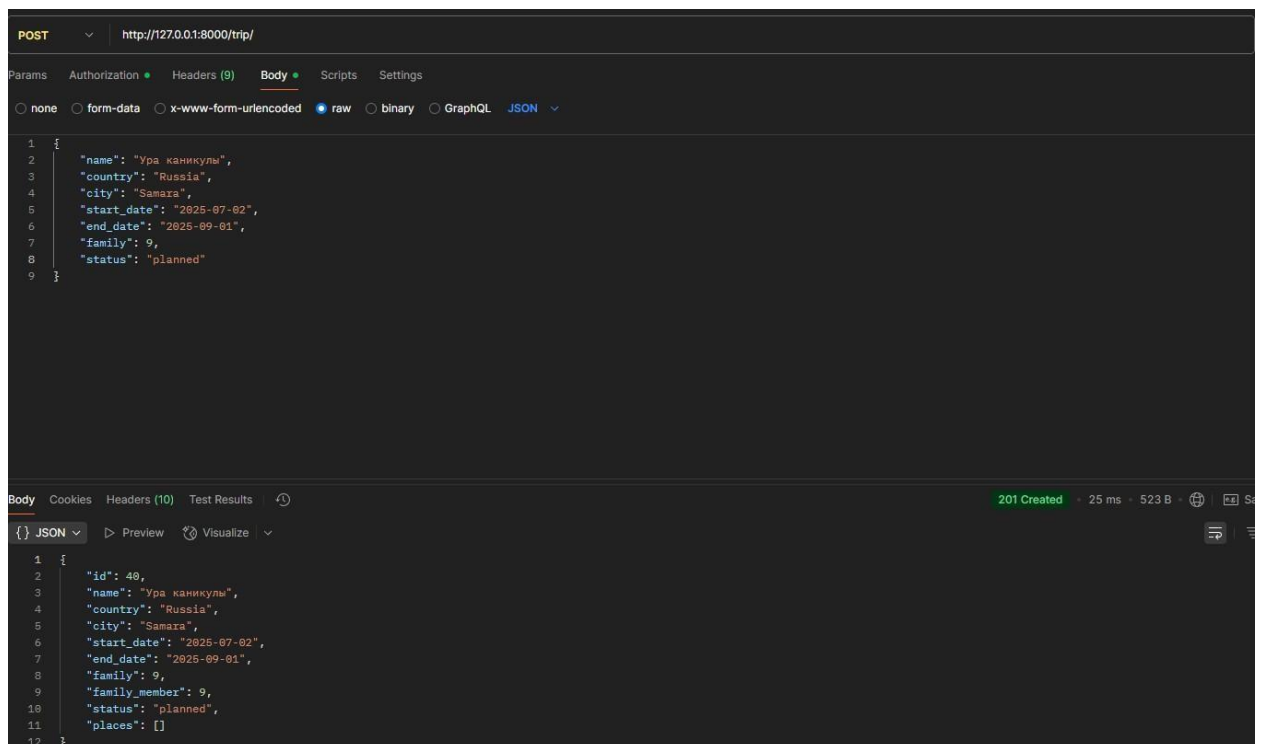


Рисунок 3 – Работа Postman

Проверка работы аутентификации через Postman

POST /api/login/

Вход в систему и получение JWT.

Request body (JSON):{

"email": "aaaaa@gmail.com",

"password": "aaaaa"

}

Responses:

200 OK + {accessToken, refreshToken}

401 Unauthorized + {"detail": "No active account found with the given credentials"}

POST /api/register/

Регистрация нового пользователя.

Request body (JSON):{

"email": "aaaaa@gmail.com",

"login": "aaaaa",

"full_name": "aaaaa",

"password": "aaaaa"

}

Responses:

201 Created + {

"email": "aaaaa@gmail.com",

"login": "aaaaa",

"full_name": "aaaaa"

}

400 Bad Request + {"email": ["Enter a valid email address."]}

POST /api/token/refresh/

Обновление accessToken по refresh-токену.

Request body: строка с refresh-токеном

Responses: 200 OK + {accessToken, refreshToken}

401 Unauthorized при невалидном токене

JWT-система при аутентификации или регистрации выдает два токена: `accessToken` и `refreshToken`. `AccessToken` имеет короткий срок жизни и передается в заголовке `Authorization` для доступа к защищенным данным, а `refreshToken` хранится дольше и используется для обновления `accessToken` после его истечения. Также настроен `blacklist`, чтобы по истечению времени жизни токен был невалиден.

ЛАБОРАТОРНАЯ РАБОТА 4

Разработка пользовательского интерфейса – Шаблонизатор: Jinja2 (Django)

base: base.html – шаблон, который определяет общую структуру HTML-страницы и используется для наследования другими шаблонами, чтобы избежать дублирования кода.

Основные страницы (templates/*.html)

Логин (login.html)

- Располагается внутри общего base.html через наследование с `{% extends 'base.html' %}` и переопределение блока `{% block content %}`.
- Содержит форму с полями email и пароль, отправляющую данные методом POST на `{% url 'login' %}`.
- После отправки данные проверяются через Django REST Framework и Simple JWT.
- Показывает ссылку на страницу регистрации через `Зарегистрироваться`.

Профиль (profile.html)

- Располагается внутри общего base.html через наследование с `{% extends 'base.html' %}` и переопределение блока `{% block content %}`.
- Содержит карточку с данными пользователя (имя, email, логин, предпочтения), отображаемыми через переменные `{{ user.full_name }}`, `{{ user.email }}`, `{{ user.login }}`, `{{ user.preferences }}`.
- После загрузки данные отображаются статически, без отправки формы (данные подгружаются через представление `profile_view`).
- Показывает ссылки на страницы смены пароля (``) и обновления профиля (`<a href="{% url`

'update-profile' %}") с использованием Bootstrap-классов (btn btn-primary, btn btn-secondary).

Семьи (families.html)

- Располагается внутри общего base.html через наследование с {% extends 'base.html' %} и переопределение блока {% block content %}.
- Содержит список семей пользователя в виде list-group, отображаемых через цикл {% for family in families %}, с кнопкой для перехода к участникам (). Если семей нет, показывается сообщение "Вы не состоите ни в одной семье."
- После отправки формы делает POST на {% url 'families' %} (обрабатывается в families_view), где отправляется запрос на вступление в семью по указанному family_id.
- Показывает ссылку на страницу создания семьи через Создать семью с использованием Bootstrap-классов (btn btn-primary).

Код этих страниц представлен в приложении Д.

Настройка взаимодействия с сервером

- Все страницы рендерятся на сервере через Jinja2-шаблоны и отправляются из Django-представлений (например, render(request, 'reviews.html', context) в reviews_view).
- Формы (если есть) отправляют данные на URL-эндпоинты через POST/GET, определённые в urls.py (в данном случае используется GET-запрос с параметром mark).
- Данные из представлений передаются в шаблоны через контекст и отображаются через {{ variable }} (например, {{ reviews }} и {{ mark }} в reviews.html).

Работа с аутентификацией

- Используется Simple JWT: `access_token` хранится в `request.session`, валидация выполняется через API-запрос с заголовком `Authorization: Bearer {token}` (например, в `reviews_view`). Если токен истёк (статус 401), вызывается `refresh_token` для обновления.
- В шаблонах для отображения контента по ролям используется `{% if condition %}`, где условие задаётся в представлении (в данном случае роли не проверяются, но могут быть добавлены, например, `is_creator`).

ЛАБОРАТОРНАЯ РАБОТА 5

Настройка базы данных в Docker

- Создан Docker-контейнер с PostgreSQL
- В файле `docker-compose.yml` добавлен сервис `db`, использующий официальный образ `postgres:17`.
 - Для хранения данных используется `volume (postgres_data)`.
 - Настроено подключение к базе данных через переменные окружения:
 - В секции `environment` для `POSTGRES` заданы переменные `POSTGRES_USER`, `POSTGRES_PASSWORD` и `POSTGRES_DB`.

Упаковка в Docker

Написан Dockerfile:

- Базовый образ: `python:3.12-slim` — минималистичный образ Python.
- Рабочая директория: Установлена в `/code`.
- Зависимости: копируется `requirements.txt`, устанавливаются через `pip install --no-cache-dir`.
- Утилиты: устанавливается `wait-for-it.sh` для ожидания сервисов, затем `curl` удаляется.
- Копирование кода: Все файлы проекта копируются в `/code/`.
- Окружение: `PYTHONDONTWRITEBYTECODE=1` и `PYTHONUNBUFFERED=1` для оптимизации и логов.
- Порт: экспонируется 8000.
- Запуск: `python manage.py runserver 0.0.0.0:8000`.
- Сборка: `docker build -t my-django-app`, запуск: `docker run -p 8000:8000 my-django-app`.

Создан файл `docker-compose.yml` для управления сервисами:

- Версия: используется 3.8.
- Сервисы:
 - db: Образ postgres:17, с переменными POSTGRES_USER=myuser, POSTGRES_PASSWORD=mypassword, POSTGRES_DB=mydatabase. Порт: 5433:5432. Данные хранятся в томе postgres_data.
 - pgadmin: Образ dpage/pgadmin4, с переменными PGADMIN_DEFAULT_EMAIL=myuser@example.com, PGADMIN_DEFAULT_PASSWORD=mypassword. Порт: 5050:80. Зависит от db.
 - web: Сборка из текущей директории (.) с Dockerfile, команда python manage.py runserver 0.0.0.0:8000. Том: ./code. Порт: 8000:8000. Зависит от db.
- Тома: postgres_data для хранения данных PostgreSQL.
- Запуск: Команда docker-compose up --build собирает и запускает все сервисы, связывая их в одну сеть.

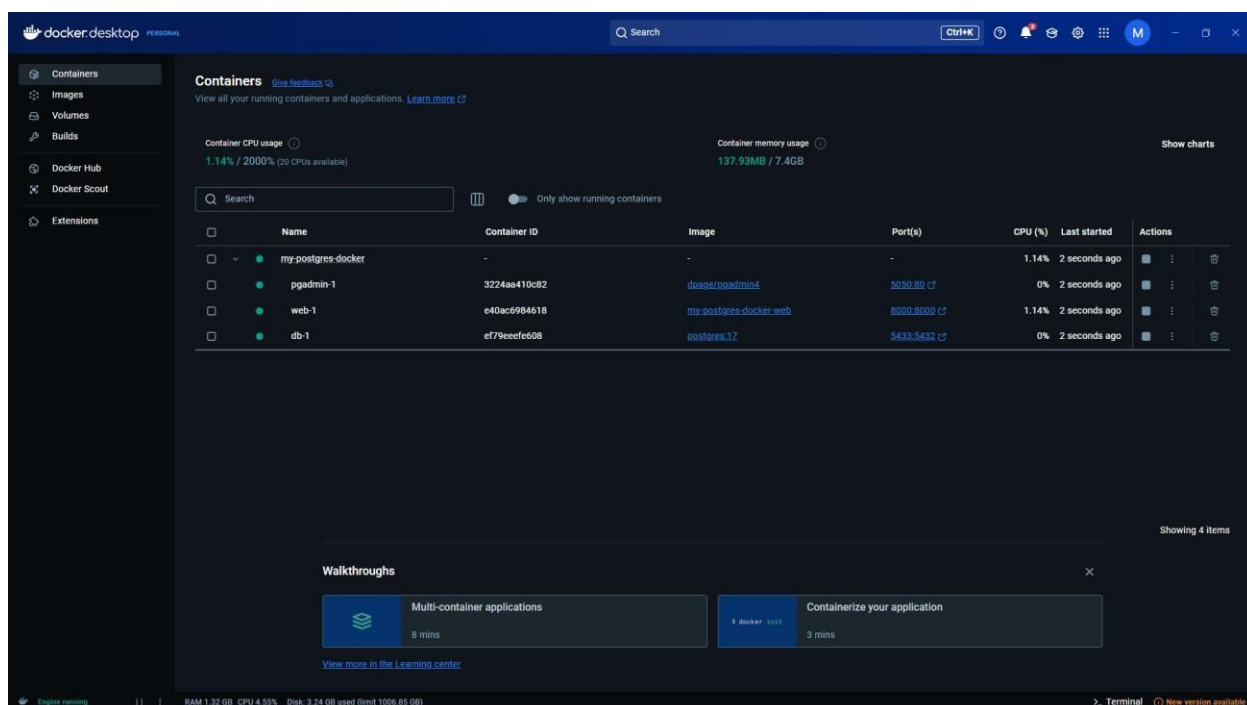
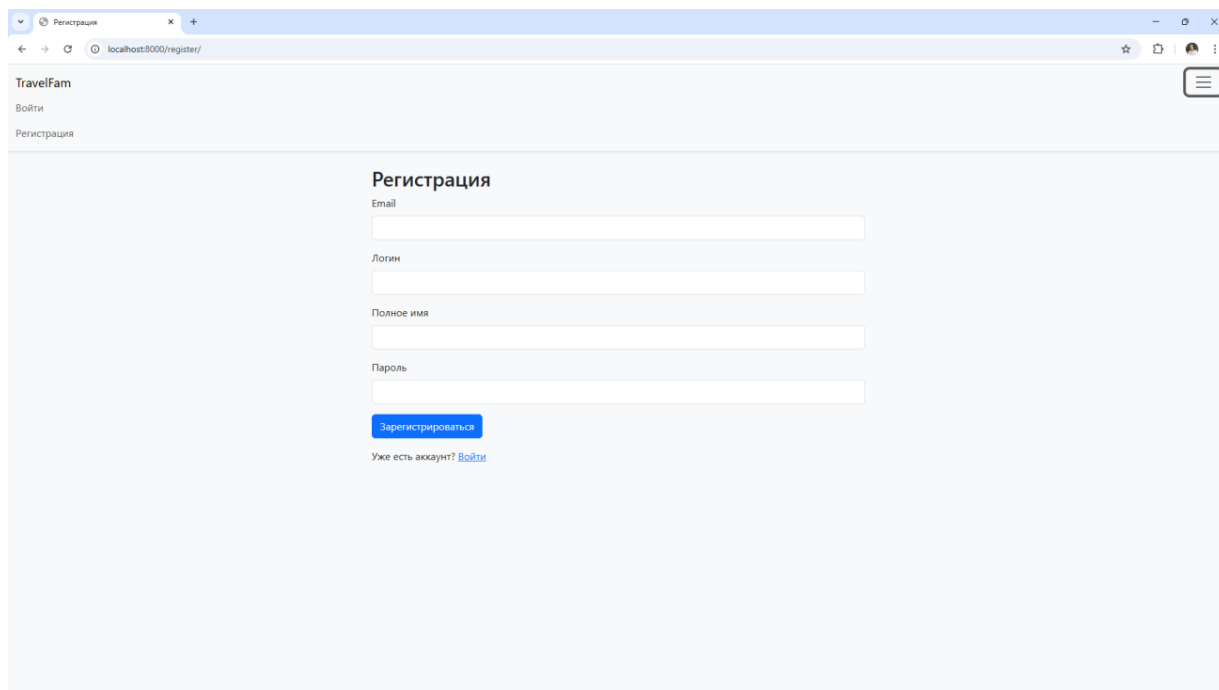


Рисунок 4 – Скриншот из Docker с запущенным приложением

Настроен .dockerignore для исключения ненужных файлов:

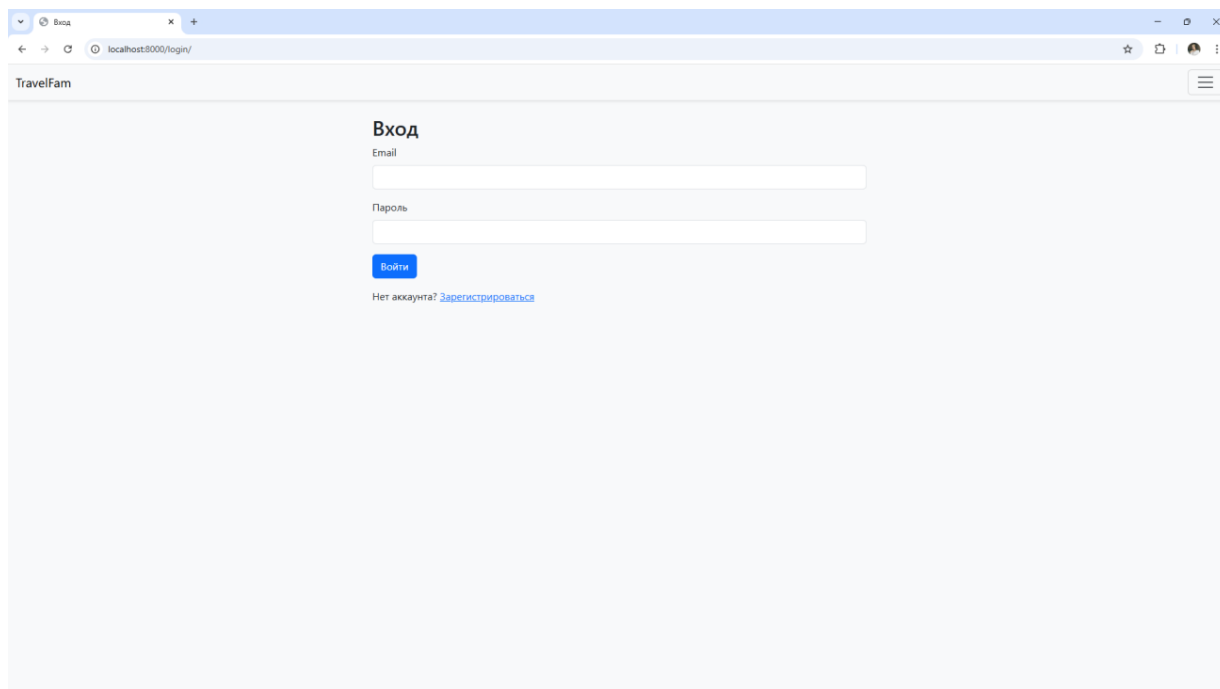
В корне проекта создан файл .dockerignore, в который добавлены каталоги и файлы, не требующиеся для сборки контейнера, что ускоряет и оптимизирует процесс сборки.



The screenshot shows a web browser window with the URL `localhost:8000/register/`. The page title is "TravelFam" and the navigation bar includes "Войти" and "Регистрация". The main content area is titled "Регистрация" and contains the following form elements:

- Email:
- Логин:
- Полное имя:
- Пароль:
- Зарегистрироваться:
- Уже есть аккаунт? [Войти](#)

Рисунок 5 – Страница регистрации



The screenshot shows a web browser window with the URL `localhost:8000/login/`. The page title is "TravelFam" and the navigation bar includes "Войти" and "Регистрация". The main content area is titled "Вход" and contains the following form elements:

- Email:
- Пароль:
- Войти:
- Нет аккаунта? [Зарегистрироваться](#)

Рисунок 6 – Вход

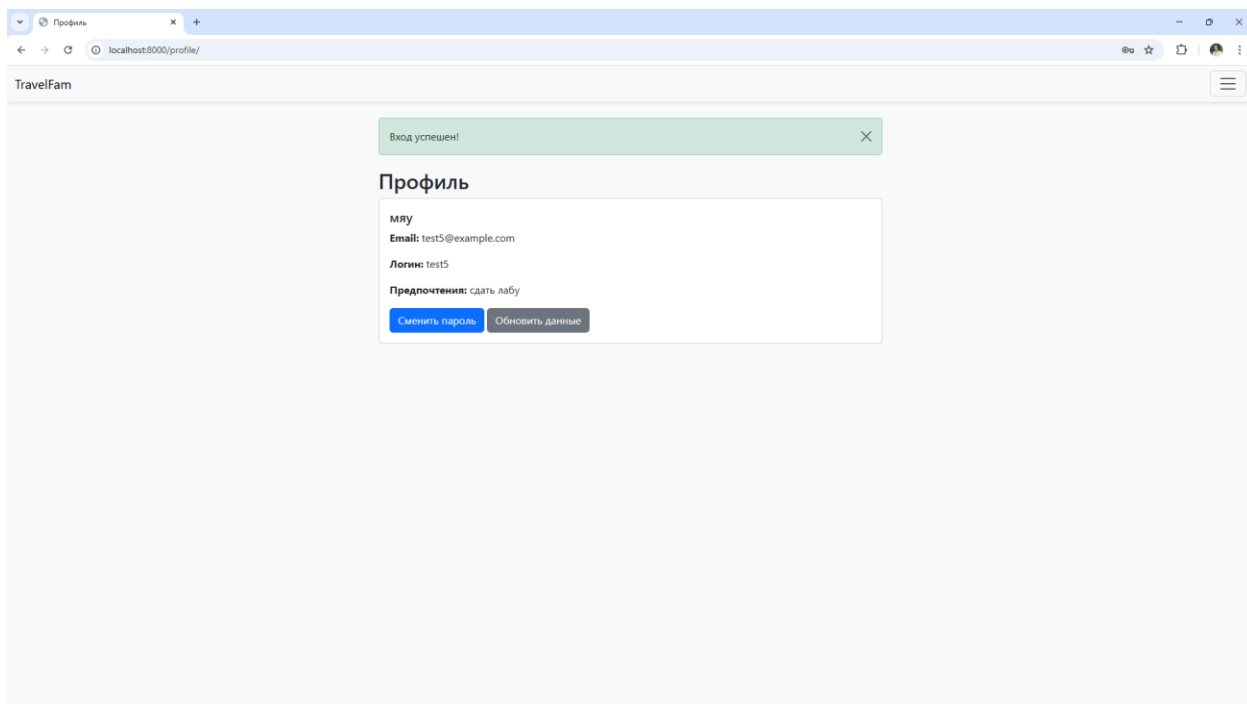


Рисунок 7 – Профиль

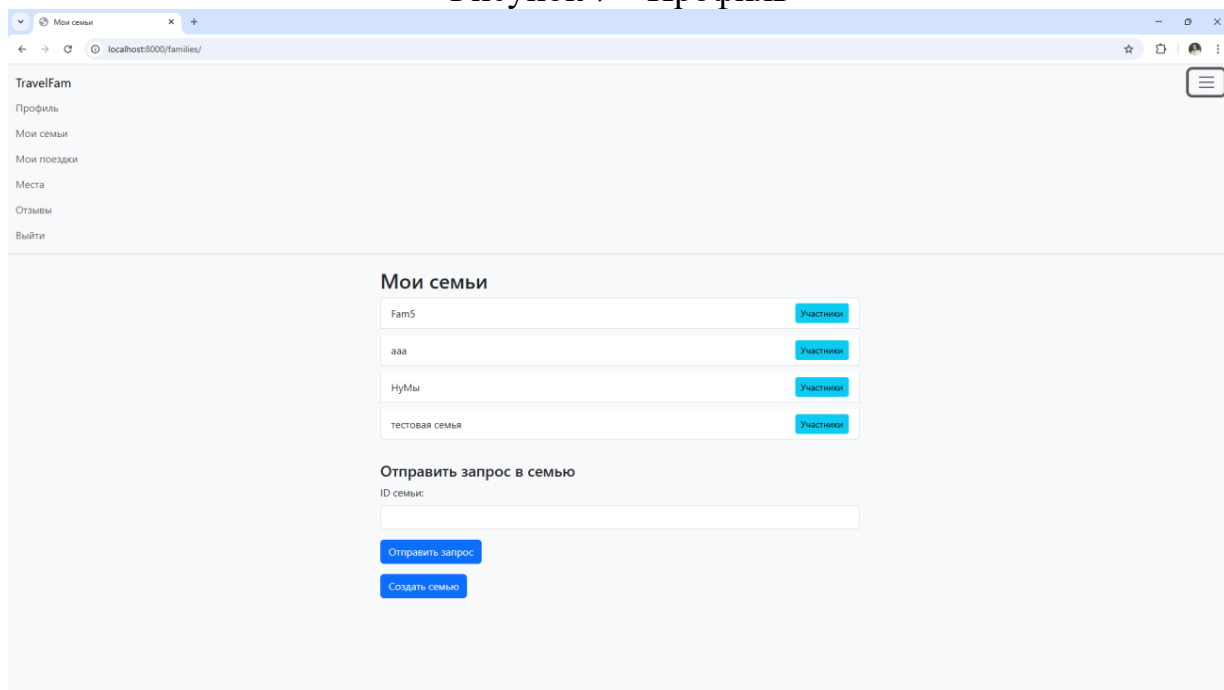


Рисунок 8 – Семьи пользователя

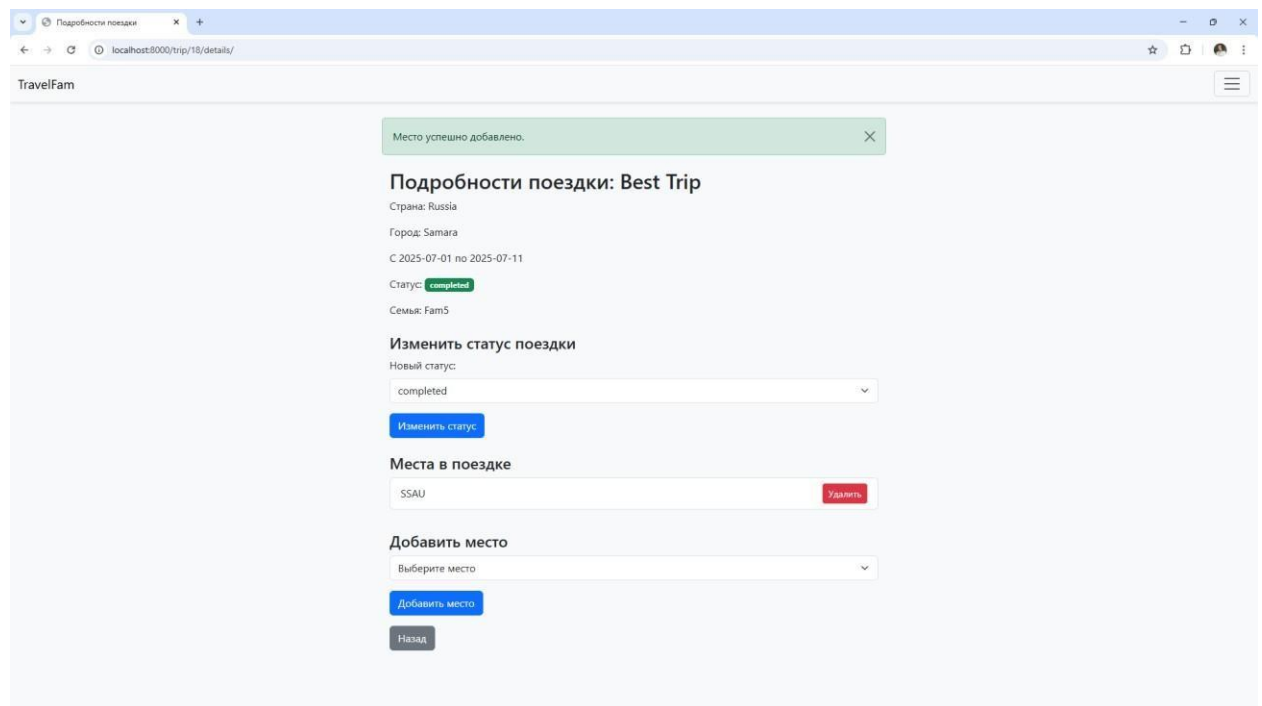


Рисунок 9 – Подробности поездки

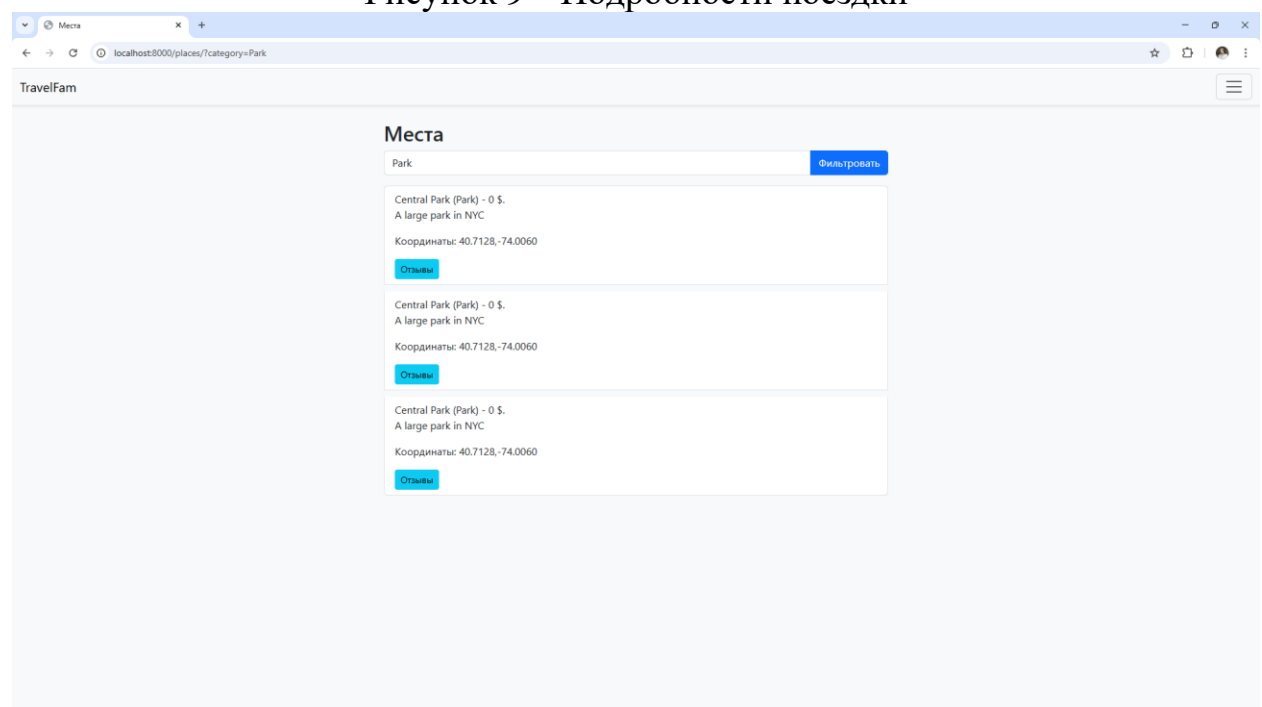


Рисунок 10 – Места

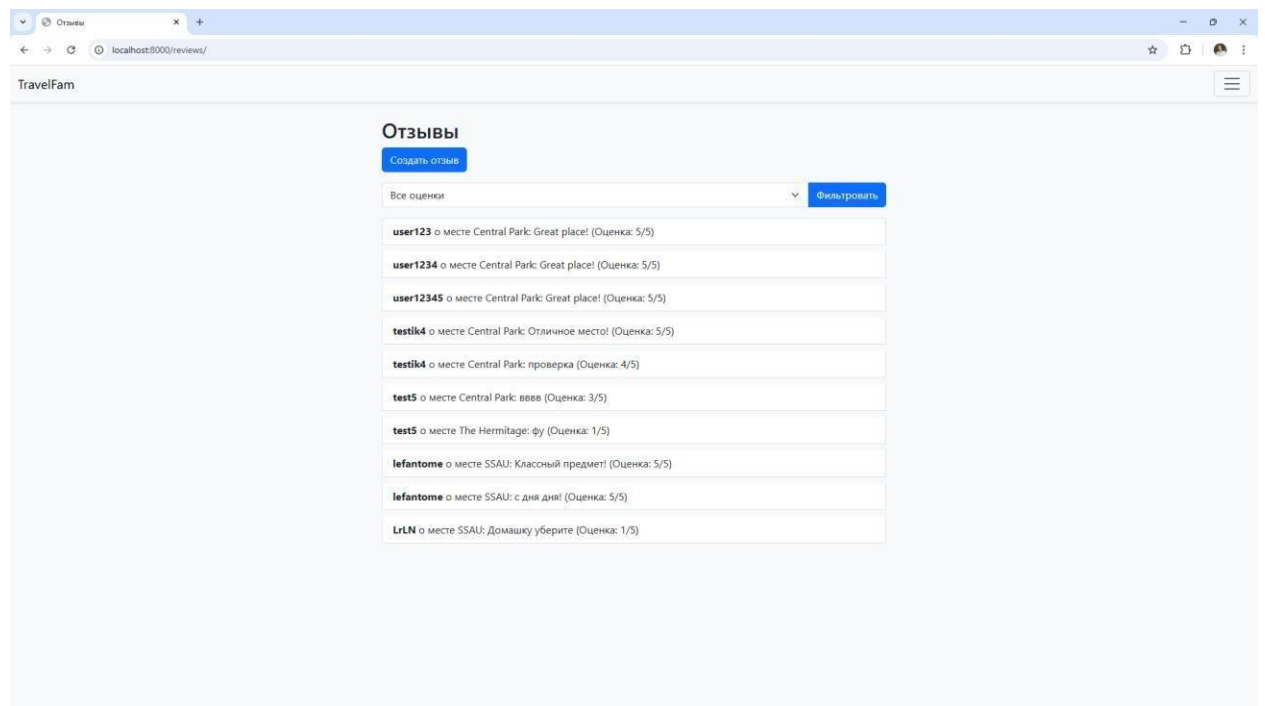


Рисунок 11 – Отзывы

ВЫВОД

Мы разработали веб-приложение TravelFam для планирования семейных путешествий, позволяющее пользователям создавать семьи, планировать поездки, добавлять места и оставлять отзывы. Проект реализован с использованием Django, Django REST Framework и Jinja2 для серверного рендеринга, а также PostgreSQL для хранения данных. Мы освоили архитектуру клиент-сервер, разработав REST API с JWT- аутентификацией для управления пользователями, семьями и поездками. Научились проектировать базу данных, создав модели с ForeignKey и ManyToMany связями, и настроили миграции. Освоили навыки контейнеризации, упаковав приложение в Docker с помощью Dockerfile и docker-compose.yml. Настроили взаимодействие сервисов (Django, PostgreSQL, PgAdmin) в общей Docker-сети с томами для данных.

Разработали пользовательский интерфейс с помощью Jinja2 и Bootstrap, обеспечив удобное отображение форм и списков. Научились обрабатывать HTTP-запросы через Django-представления, интегрируя их с API. Проверили работу API с помощью Postman, настроив аутентификацию и обновление токенов. В результате мы приобрели опыт командной разработки, проектирования веб-приложений и управления контейнерами.

ПРИЛОЖЕНИЕ А

```
class CustomUserManager(BaseUserManager):
    def create_user(self, email, login, full_name, password, **extra_fields):
        if not email:
            raise ValueError('Email обязателен')
        if not login:
            raise ValueError('Логин обязателен')
        if not full_name:
            raise ValueError('Полное имя обязательно')
        if not password:
            raise ValueError('Пароль обязателен')

        email = self.normalize_email(email)
        user = self.model(email=email, login=login, full_name=full_name,
**extra_fields)
        user.set_password(password)
        user.save(using=self._db)
        return user

    def create_superuser(self, email, login, full_name, password=None,
**extra_fields):
        extra_fields.setdefault('is_staff', True)
        extra_fields.setdefault('is_superuser', True)
        return self.create_user(email, login, full_name, password, **extra_fields)

class User(AbstractBaseUser, PermissionsMixin):
    email = models.EmailField(max_length=50, unique=True)
    login = models.CharField(max_length=50, unique=True)
```

```
full_name = models.CharField(max_length=50)
preferences = models.CharField(max_length=100, null=True, blank=True)
create_date = models.DateField(default=timezone.now)
is_active = models.BooleanField(default=True)
is_staff = models.BooleanField(default=False)
```

```
objects = CustomUserManager()
```

```
USERNAME_FIELD = 'email'
REQUIRED_FIELDS = ['login', 'full_name']
```

```
def __str__(self):
    return self.full_name
```

```
class Family(models.Model):
    name = models.CharField(max_length=50)
    create_date = models.DateField(auto_now_add=True)

    def __str__(self):
        return self.name
```

```
class FamilyMember(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    family = models.ForeignKey(Family, on_delete=models.CASCADE)
    role = models.CharField(max_length=50, default='member')
    def __str__(self):
        return f"{self.user.full_name} in {self.family.name}"
```



```

class FamilyRequests(models.Model):
    PENDING = 'в ожидании'
    ACCEPTED = 'принят'
    DECLINED = 'отклонён'
    STATUS_CHOICES = [
        (PENDING, 'В ожидании'),
        (ACCEPTED, 'Принят'),
        (DECLINED, 'Отклонён'),
    ]
    family = models.ForeignKey(Family, on_delete=models.CASCADE)
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    create_date = models.DateField(default=timezone.now)
    status = models.CharField(
        max_length=10,
        choices=STATUS_CHOICES,
        default=PENDING
    )

    class Meta:
        unique_together = ('family', 'user')

    def __str__(self):
        return f"Request from {self.user.full_name} to {self.family.name} "
        (Status: {self.status})"

class Place(models.Model):
    coordinates = models.CharField(max_length=30)
    name = models.CharField(max_length=50)

```

```

cost = models.IntegerField(null=True, blank=True)
category = models.CharField(max_length=100, null=True, blank=True)
description = models.CharField(max_length=500, null=True, blank=True)

def __str__(self):
    return self.name

class Reviews(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    mark = models.IntegerField()
    text = models.CharField(max_length=500)
    place = models.ForeignKey(Place, on_delete=models.CASCADE)
    id = models.AutoField(primary_key=True)
    def __str__(self):
        return f"Review by {self.user.full_name} for {self.place.name}"

class Trip(models.Model):
    name = models.CharField(max_length=50)
    country = models.CharField(max_length=50)
    city = models.CharField(max_length=50)
    start_date = models.DateField()
    end_date = models.DateField(null=True, blank=True)
    family_member = models.ForeignKey(FamilyMember,
on_delete=models.CASCADE)
    family = models.ForeignKey(Family, on_delete=models.CASCADE)
    status = models.CharField(max_length=50)
    places = models.ManyToManyField(Place, through='TripPlace',
related_name='trips', blank=True)

```

```
def __str__(self):  
    return self.name
```

```
class TripPlace(models.Model):  
    trip = models.ForeignKey(Trip, on_delete=models.CASCADE)  
    place = models.ForeignKey(Place, on_delete=models.CASCADE)  
    def __str__(self):  
        return f"{self.place.name} in {self.trip.name}"
```

ПРИЛОЖЕНИЕ Б

```
class ReviewViewSet(viewsets.ModelViewSet):
    queryset = Reviews.objects.all()
    serializer_class = ReviewSerializer
    permission_classes = [permissions.IsAuthenticated]

    def get_queryset(self):
        if 'place_id' in self.kwargs:
            return Reviews.objects.filter(place_id=self.kwargs['place_id'])
        return super().get_queryset()

    def perform_create(self, serializer):
        if 'place_id' in self.kwargs:
            serializer.save(
                user=self.request.user,
                place_id=self.kwargs['place_id']
            )
        else:
            serializer.save(user=self.request.user)
```

ПРИЛОЖЕНИЕ В

serializer:

```
class FamilyRequestSerializer(serializers.ModelSerializer):  
    class Meta:  
        model = FamilyRequests  
        fields = ['id', 'user', 'status']  
  
    def create(self, validated_data):  
        family_id = self.context['view'].kwargs['family_id']  
        user = validated_data['user']  
        if FamilyMember.objects.filter(user=user, family_id=family_id).exists():  
            raise serializers.ValidationError(  
                {'error': 'Вы уже являетесь членом этой семьи'}  
            )  
        if FamilyRequests.objects.filter(  
            user=user,  
            family_id=family_id,  
            status=FamilyRequests.PENDING  
        ).exists():  
            raise serializers.ValidationError(  
                {'error': 'У вас уже есть ожидающий запрос в эту семью'}  
            )  
        validated_data['family_id'] = family_id  
        return super().create(validated_data)
```

urls:

```
path('api/register/', views.UserRegistrationView.as_view(), name='register'),  
path('api/login/', views.UserLoginView.as_view(), name='login'),
```

```
    path('api/user/<int:user_id>/change-password/',
views.UserViewSet.as_view({'post': 'change_password'}), name='change-
password'),
    path('api/logout/', views.UserLogoutView.as_view(), name='logout'),
    path('api/user/<int:user_id>/', views.UserViewSet.as_view({'get': 'retrieve',
'put': 'update'}), name='user-detail'),
```

ПРИЛОЖЕНИЕ Д

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 %}TravelFam{% endblock %}</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet"                                integrity="sha384-
9ndCyUaIbzAi2FUVXJi0CjmCapSmO7SnpJef0486qhLnuZ2cdeRhO02iuK6FUUV
M" crossorigin="anonymous">
    <link rel="stylesheet" href="{% static 'css/style.css' %}">
</head>
<body>
    <nav class="navbar navbar navbar-light bg-light">
        <div class="container-fluid">
            <a class="navbar-brand" href="/">TravelFam</a>
            <button      class="navbar-toggler"      type="button"      data-bs-
toggle="collapse" data-bs-target="#navbarNav" aria-controls="navbarNav" aria-
expanded="false" aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarNav">
                <ul class="navbar-nav ms-auto">
                    {% if request.session.access_token %}
```

```

<li class="nav-item">
    <a class="nav-link" href="{ % url 'profile'
%}">Профиль</a>
</li>
<li class="nav-item">
    <a class="nav-link" href="{ % url 'families' % }">Мои
семьи</a>
</li>
<li class="nav-item">
    <a class="nav-link" href="{ % url 'trips' % }">Мои
поездки</a>
</li>
<li class="nav-item">
    <a class="nav-link" href="{ % url 'places' % }">Места</a>
</li>
<li class="nav-item">
    <a class="nav-link" href="{ % url 'reviews'
% }">Отзывы</a>
</li>
<li class="nav-item">
    <a class="nav-link" href="{ % url 'logout' % }">Выйти</a>
</li>
{ % else % }
<li class="nav-item">
    <a class="nav-link" href="{ % url 'login' % }">Войти</a>
</li>
<li class="nav-item">

```



```

        <a class="nav-link" href="{ % url 'register'
%}">Регистрация</a>
    </li>
    {% endif %}
</ul>
</div>
</div>
</nav>
<div class="container mt-4">
    {% if messages %}
        {% for message in messages %}
            <div class="alert alert-{ { message.tags } } alert-dismissible fade
show" role="alert">
                { { message } }
                <button type="button" class="btn-close" data-bs-dismiss="alert"
aria-label="Close"></button>
            </div>
        {% endfor %}
    {% endif %}
    {% block content %}
    {% endblock %}
</div>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"
integrity="sha384-
geWF76RCwLtnZ8qwWowPQNguL3RmwHVBC9FhGdlKrxdiJJigb/j/68SIy3Te4B
kz" crossorigin="anonymous"></script>
</body>

```

</html>

login.html:

{ % extends 'base.html' % }

{ % load static % }

{ % block title % }Вход{ % endblock % }

{ % block content % }

<h2>Вход</h2>

{ % if error % }

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

{ % endif % }

<form method="POST" action="{ % url 'login' % }">

{ % csrf_token % }

<div class="mb-3">

<label for="email" class="form-label">Email</label>

<input type="email" class="form-control" id="email" name="email"

required>

</div>

<div class="mb-3">

<label for="password" class="form-label">Пароль</label>

<input type="password" class="form-control" id="password"

name="password" required>

</div>

<button type="submit" class="btn btn-primary">Войти</button>

</form>

```

    <p class="mt-3">Нет аккаунта? <a href="{% url 'register'
%}">Зарегистрироваться</a></p>
{% endblock %}

```

profile.html:

```

{% extends 'base.html' %}
{% load static %}

{% block title %}Профиль{% endblock %}

{% block content %}
<h2>Профиль</h2>
<div class="card">
    <div class="card-body">
        <h5 class="card-title">{{ user.full_name }}</h5>
        <p class="card-text"><strong>Email:</strong> {{ user.email }}</p>
        <p class="card-text"><strong>Логин:</strong> {{ user.login }}</p>
        <p class="card-text"><strong>Предпочтения:</strong> {{
user.preferences|default:"Не указаны" }}</p>
        <a href="{% url 'change-password' %}" class="btn btn-
primary">Сменить пароль</a>
        <a href="{% url 'update-profile' %}" class="btn btn-
secondary">Обновить данные</a>
    </div>
</div>
{% endblock %}

```

families.html:

```

{% extends 'base.html' %}

{% load static %}


{% block title %}Мои семьи{% endblock %}


{% block content %}
<div class="container">
    <h2>Мои семьи</h2>
    {% if families %}
        <ul class="list-group">
            {% for family in families %}
                <li class="list-group-item d-flex justify-content-between align-items-center">
                    <span>{{ family.name }}</span>
                    <a href="{% url 'family-members' family_id=family.id %}"
class="btn btn-sm btn-info">Участники</a>
                </li>
            {% endfor %}
        </ul>
    {% else %}
        <p>Вы не состоите ни в одной семье.</p>
    {% endif %}


    <h4 class="mt-4">Отправить запрос в семью</h4>
    <form method="POST" action="{% url 'families' %}">
        {% csrf_token %}
        <div class="mb-3">
            <label for="family_id" class="form-label">ID семьи:</label>

```

```

        <input type="number" name="family_id" id="family_id" class="form-
control" required>
    </div>
    <button type="submit" class="btn btn-primary">Отправить
запрос</button>
</form>

    <a href="{% url 'create-family' %}" class="btn btn-primary mt-3">Создать
семью</a>
</div>
{% endblock %}

```