**Содержание**

[Введение 5](#_Toc190677557)

[1 Постановка задачи и обзор аналогичных решений 6](#_Toc190677558)

[1.1 Постановка задачи 6](#_Toc190677559)

[1.2 Обзор аналогичных решений 6](#_Toc190677560)

[1.1.1 Web-приложение «Eventbrite» 6](#_Toc190677561)

[1.1.2 Web-приложение «Meetup» 7](#_Toc190677562)

[1.1.3 Web-приложение «Eventful» 8](#_Toc190677563)

[1.3 Выводы по разделу 9](#_Toc190677564)

[2 Проектирование web-приложения 10](#_Toc190677565)

[2.1 Функциональность web-приложения 10](#_Toc190677566)

[2.2 Проектирование базы данных 12](#_Toc190677567)

[2.3 Архитектура web-приложения 14](#_Toc190677568)

[2.4 Выводы по разделу 15](#_Toc190677569)

[3 Реализация web-приложения 16](#_Toc190677570)

[3.1 Обоснование выбора программной платформы 16](#_Toc190677571)

[3.2 Система управления базами данных PostgreSQL 16](#_Toc190677572)

[3.3 Object-Relational Mapping (ORM) Django ORM 16](#_Toc190677573)

[3.4 Программные библиотеки 19](#_Toc190677574)

[3.5 Структура серверной части 22](#_Toc190677575)

[3.6 Реализация функционала для пользователя с ролью «Гость» 23](#_Toc190677576)

[3.6.1 Регистрация 23](#_Toc190677577)

[3.6.2 Аутентификация 23](#_Toc190677578)

[3.7 Реализация функционала для пользователя с ролью «Участник» 24](#_Toc190677579)

[3.7.1 Редактирование пользовательских данных 24](#_Toc190677580)

[3.7.2 Просмотр мероприятий 24](#_Toc190677581)

[3.7.3 Подача заявки на участие 25](#_Toc190677582)

[3.7.4 Отмена заявки 25](#_Toc190677583)

[3.7.5 Добавление мероприятия в Google Календарь 25](#_Toc190677584)

[3.7.6 Создание отзыва 26](#_Toc190677585)

[3.7.7 Просмотр профиля пользователя 27](#_Toc190677586)

[3.7.8 Поиск и фильтрация мероприятий 27](#_Toc190677587)

[3.7.9 Создание мероприятия 28](#_Toc190677588)

[3.8 Реализация функционала для пользователя с ролью «Создатель» 28](#_Toc190677589)

[3.8.1 Редактирование мероприятия 28](#_Toc190677590)

[3.8.2 Просмотр заявок 29](#_Toc190677591)

[3.8.3 Принятие заявок 29](#_Toc190677592)

[3.8.4 Отклонение заявок 30](#_Toc190677593)

[3.9 Реализация функционала для пользователя с ролью «Администратор» 31](#_Toc190677594)

[3.9.1 Управление пользователями 31](#_Toc190677595)

[3.9.2 Управление мероприятиями 31](#_Toc190677596)

[3.10 Структура клиентской части 32](#_Toc190677597)

[3.11 Выводы по разделу 34](#_Toc190677598)

[4 Тестирование web-приложения 35](#_Toc190677599)

[4.1 Функциональное тестирование 35](#_Toc190677600)

[4.2 Тестирование функции добавления мероприятий в Google Календарь 37](#_Toc190677601)

[4.3 Результаты тестирования функций администратора 38](#_Toc190677602)

[4.4 Выводы по разделу 39](#_Toc190677603)

[5 Руководство пользователя 40](#_Toc190677604)

[5.1 Руководство для пользователя с ролью «Гость» 40](#_Toc190677605)

[5.1.1 Вход в аккаунт 40](#_Toc190677606)

[5.1.2 Регистрация 40](#_Toc190677607)

[5.2 Руководство для пользователя с ролью «Участник» 41](#_Toc190677608)

[5.2.1 Создание и редактирование профиля 41](#_Toc190677609)

[5.2.2 Просмотр мероприятий 42](#_Toc190677610)

[5.2.3 Подача и отмена заявок на присоединение к мероприятию 43](#_Toc190677611)

[5.2.4 Добавление мероприятия в Google Календарь 44](#_Toc190677612)

[5.2.5 Написание отзывов 44](#_Toc190677613)

[5.2.6 Просмотр профилей других пользователей 45](#_Toc190677614)

[5.2.7 Поиск и фильтрация мероприятий 46](#_Toc190677615)

[5.2.8 Создание мероприятия 46](#_Toc190677616)

[5.3 Руководство пользователя с ролью «Создатель» 47](#_Toc190677617)

[5.3.1 Редактирование мероприятия 47](#_Toc190677618)

[5.3.2 Просмотр заявок на участие в созданных мероприятиях 48](#_Toc190677619)

[5.3.3 Редактирование списка участников созданных мероприятий 49](#_Toc190677620)

[5.4 Руководство пользователя с ролью «Администратор» 50](#_Toc190677621)

[5.4.1 Редактирование списка пользователей 50](#_Toc190677622)

[5.4.2 Редактирование списка всех мероприятий 51](#_Toc190677623)

[5.5 Выводы по разделу 51](#_Toc190677624)

[Заключение 52](#_Toc190677625)

[Список используемых источников 53](#_Toc190677626)

[Приложение А. Логическая схема базы данных 54](#_Toc190677627)

[Приложение Б. Архитектура web-приложения 55](#_Toc190677628)

[Приложение В. Создание базы данных 56](#_Toc190677629)

# **Введение**

Web-приложения для организации и участия в мероприятиях позволяют пользователям организовывать и находить мероприятия, устанавливать новые социальные связи и координировать участие. Они предоставляют функции для создания мероприятий и участия в них, просмотра информации о мероприятиях и взаимодействия с другими участниками на основе различных предпочтений.

Web-приложение – это клиент-серверное приложение, в котором клиент взаимодействует с сервером по протоколу HTTP [1].

Цель курсового проекта – автоматизировать процесс организации и участия в мероприятиях, включая создание и управление мероприятиями, поиск подходящих мероприятий, взаимодействие участников и обмен сообщениями.

Web-приложение должно поддерживать несколько ролей: гость, участник, создатель, администратор.

Для достижения цели были поставлены следующие задачи:

1. Провести анализ существующих приложений для организации и участия в мероприятиях и определить ключевые требования к разрабатываемому web-приложению (раздел 1);
2. Разработать архитектуру web-приложения (раздел 2);
3. Реализовать функционал web-приложения с учетом поставленных требований (раздел 3);
4. Провести тестирование для выявления и устранения ошибок, а также для проверки соответствия требованиям (раздел 4);
5. Разработать руководство пользователя web-приложения (раздел 5)

Целевая аудитория web-приложения – люди от 18 лет, заинтересованные в организации и участии в различных мероприятиях, включая культурные, образовательные, развлекательные и общественные мероприятия.

В качестве программной платформы было решено использовать Django Rest Framework [2].

* 1. **Постановка задачи и обзор аналогичных решений**
     1. **Постановка задачи**

Веб-приложение «EventFlow» для организации и участия в мероприятиях должно автоматизировать процесс планирования мероприятий, включая управление мероприятиями, просмотр мероприятий, создание новых мероприятий на карте, а также присоединение к уже имеющимся.

Функционально web-приложение должно поддерживать:

Функционал для пользователя с ролью “Гость”:

− регистрации и аутентификации.

Функционал для пользователя с ролью “Участник”:

− редактировать пользовательских данных;

− просматривать мероприятия;

− подавать и отменять заявку на присоединение к мероприятию;

– добавлять мероприятия в Google Календарь;

– оставлять отзывы о прошедших мероприятиях;

– просматривать профили других пользователей;

– просматривать информацию о мероприятии;

– просматривать профили других пользователей;

– искать мероприятия по ключевым словам и фильтровать по категориям;

– создавать мероприятия.

Функционал для пользователя с ролью “Создатель”:

– редактировать информацию о созданных мероприятиях;

– просматривать заявки на участие в созданных мероприятиях;

– редактировать список участников созданных мероприятий;

Функционал для пользователя с ролью “Администратор”:

− редактировать список всех пользователей;

– редактировать список всех мероприятий.

**1.2 Обзор аналогичных решений**

* + 1. **Web-приложение «Eventbrite»**

Приложение «Eventbrite» [3] для создания мероприяий и продвижения мероприятий. В нем можно просматривать мероприятия по местоположению с отображением на карте.

Страница приложения «Eventbrite» представлена на рисунке 1.1.

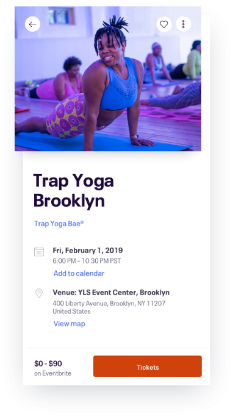


Рисунок 1.1 – Страница приложения «Eventbrite»

Достоинства:

* интеграция с картами, позволяющая легко находить мероприятия по географическому расположению и отслеживать их на карте;
* гибкие настройки мероприятий, включая возможность добавления описаний, изображений и видео, что помогает привлечь внимание участников.

Недостатки:

* отсутствие возможности полноценно участвовать в мероприятиях через приложение, как, например, в социальных сетях, – в первую очередь ориентировано на организацию и управление мероприятиями, а не на взаимодействие участников;
* дополнительные комиссии за использование сервиса, которые могут быть непривлекательными для организаторов с ограниченным бюджетом.

**1.1.2 Web-приложение «Meetup»**

Приложение «Meetup» [4] для поиска и организации мероприятий и встреч по интересам. Мероприятия отображаются на карте, позволяя пользователям легко находить мероприятия поблизости.

Страница приложения «Meetup» представлена на рисунке 1.2.

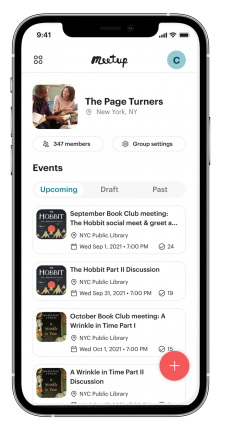


Рисунок 1.2 – Страница приложения «Meetup»

Достоинства:

* возможность общения между участниками через чаты и форумы, что способствует созданию долгосрочных социальных связей;
* геолокация, которая позволяет находить мероприятия вблизи, а также фильтровать мероприятия по местоположению и времени;
* широкий выбор мероприятий по интересам, что помогает людям найти сообщества и мероприятия, соответствующие их увлечениям.

Недостатки:

* нет полноценной системы рекомендаций, несмотря на фильтрацию по интересам, алгоритмы предложения мероприятий могут быть не всегда точными.

**1.1.3 Web-приложение «Eventful»**

Web-приложение «Eventful» [5] для поиска и организации мероприятий, включая концерты, фестивали, выставки и спортивные мероприятия. Позволяет пользователям находить мероприятия по интересам и местоположению, а организаторам – продвигать свои мероприятия.

Страница приложения «Eventful» представлена на рисунке 1.3.

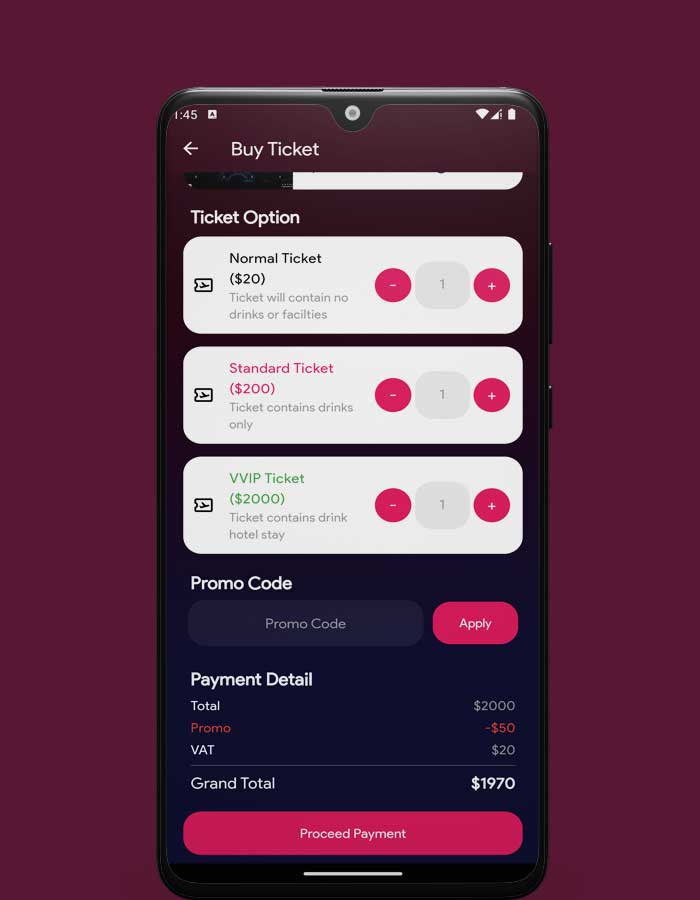


Рисунок 1.3 – Страница приложения «Eventful»

Достоинства:

* интеграция с картой, позволяющая легко находить мероприятия по местоположению и отслеживать их, полезно для тех, кто ищет мероприятия в своем регионе;
* уведомления о новых мероприятиях, позволяя пользователям подписываться на интересующие их типы мероприятий и получать уведомления о новых мероприятиях.

Недостатки:

* проблемы с географической точностью, так как иногда мероприятия могут отображаться с неточными местоположениями, что может сбить с толку пользователей;
* ограниченная функция поиска, так как иногда поиск мероприятий может быть не таким точным из-за недостаточной фильтрации или специфичности критериев.

**1.3 Выводы по разделу**

1. Анализ аналогичных решений показал, что приложения для организации и участия в мероприятиях предлагают базовый функционал, включая создание и управление мероприятиями, поиск мероприятий по интересам, регистрацию участников и интеграцию с картами для удобного поиска. Однако существуют и недостатки, такие как ограниченные возможности в бесплатных версиях, а также отсутствие функционала для точной сортировки мероприятий в некоторых приложениях.

2. Поставленные задачи требуют разработки web-приложения с поддержкой четырех ролей: гостя, участника, создателя и администратора, каждая из которых будет обладать своим перечнем функциональных возможностей, рассмотренных в пункте 1.1.

В результате разработки web-приложения «EventFlow» для организации и участия в мероприятиях будет создано решение, которое автоматизирует ключевые процессы, включая создание и управление мероприятиями, поиск мероприятий, участие в них и взаимодействие между организаторами и участниками.

1. **Проектирование web-приложения**

**2.1 Функциональность web-приложения**

Функциональные возможности web-приложения представлены в диаграмме вариантов использования, представленной на рисунке 2.1.

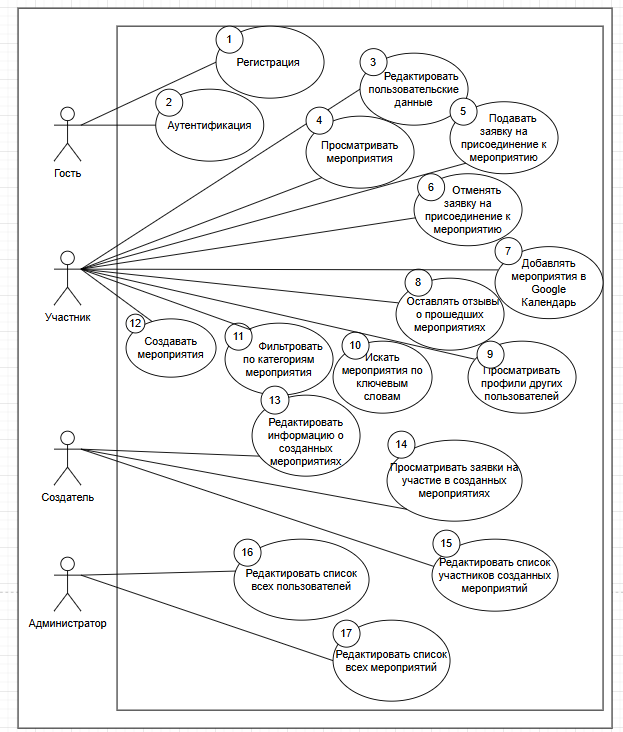


Рисунок 2.1 – Диаграмма вариантов использования web-приложения

Описание ролей пользователей web-приложения представлены в таблице 2.1.

Таблица 2.1 – Описание ролей пользователей web-приложения

|  |  |
| --- | --- |
| Роль | Описание |
| Гость | Пользователь, не прошедший аутентификацию. Может зарегистрироваться или войти в систему. |

Продолжение таблицы 2.1

|  |  |
| --- | --- |
| Роль | Описание |
| Участник | Авторизованный пользователь. Может просматривать мероприятия, оставлять отзывы, подавать/отменять заявки, синхронизировать мероприятия с календарем, создавать мероприятия |
| Создатель | Редактировать их информацию, управлять заявками. |
| Администратор | Уполномоченный пользователь. Имеет доступ к редактированию списка всех пользователей и мероприятий. |

Роли в системе разделены таким образом, чтобы разграничить доступ к функционалу приложения и обеспечить безопасное использование платформы.

Функциональные возможности пользователя с ролью «Гость» приведены в таблице 2.2.

Таблица 2.2 – Функциональные возможности пользователя с ролью «Гость»

|  |  |  |
| --- | --- | --- |
| № | Вариант использования | Пояснение |
| 1 | Регистрация | Возможность создания новой учетной записи в системе. |
| 2 | Аутентификация | Вход в систему с использованием логина и пароля. |

Функциональные возможности пользователя с ролью «Участник» приведены в таблице 2.3.

Таблица 2.3 – Функциональные возможности пользователя с ролью «Участник»

|  |  |  |
| --- | --- | --- |
| № | Вариант использования | Пояснение |
| 3 | Редактирование пользовательских данных | Изменение личной информации в профиле. |
| 4 | Просмотр мероприятий | Поиск и изучение доступных мероприятий. |
| 5 | Подача заявки на присоединение к мероприятию | Отправка запроса на участие в мероприятии. |
| 6 | Отмена заявки | Удаление ранее поданной заявки. |
| 7 | Добавление мероприятия в Google Календарь | Экспорт данных о мероприятии в календарь. |
| 8 | Оставление отзывов о мероприятиях | Публикация комментариев о завершенных мероприятиях. |
| 9 | Просмотр профилей других пользователей | Поиск и изучение профилей участников. |
| 10 | Фильтрация мероприятий по категориям | Сортировка мероприятий по выбранным критериям. |
| 11 | Поиск мероприятий по ключевым словам | Использование текстового запроса для поиска. |
| 12 | Создание мероприятий | Организация новых мероприятий с указанием деталей. |

Функциональные возможности пользователя с ролью «Создатель» представлены в таблице 2.4.

Таблица 2.4 – Функциональные возможности пользователя с ролью «Создатель»

|  |  |  |
| --- | --- | --- |
| № | Вариант использования | Пояснение |
| 13 | Редактирование информации о мероприятиях | Изменение описания, времени, категории. |
| 14 | Просмотр заявок на участие | Управление списком запросов к мероприятию. |
| 15 | Редактирование списка участников | Подтверждение/отклонение заявок. |

Функциональные возможности пользователя с ролью «Администратор» представлены в таблице 2.5.

Таблица 2.5 – Функциональные возможности пользователя с ролью «Администратор»

|  |  |  |
| --- | --- | --- |
| № | Вариант использования | Пояснение |
| 16 | Редактирование списка пользователей | Изменение данных, блокировка аккаунтов. |
| 17 | Редактирование списка мероприятий | Удаление или модификация любых мероприятий. |

Эта система ролей и вариантов использования охватывает весь функционал приложения, обеспечивая разграничение прав доступа и удобство работы с платформой для всех категорий пользователей.

**2.2 Проектирование базы данных**

Согласно диаграмме вариантов использования была разработана база данных. PostgreSQL [6] является современной реляционной базой данных, которая использует табличную модель для хранения данных. Она поддерживает строгую структуру данных, обеспечивая целостность и надежность информации. Логическая схема базы данных приведена в приложении А.

База данных содержит 7 таблиц, назначение каждой таблицы представлено в таблице 2.6.

Таблица 2.6 – Назначение таблиц базы данных

|  |  |
| --- | --- |
| Коллекция | Назначение |
| Tag | Хранение информации о категориях мероприятий |
| Event | Хранение информации о мероприятиях |
| Review | Хранение отзывов пользователей |
| ReviewImage | Хранение изображений, прикрепленных к отзывам |
| EventApplication | Хранение заявок пользователей на участие в мероприятиях |
| UserToken | Хранение токенов пользователей |
| CustomUser | Хранение информации о пользователях |

Таблица Tag служит для хранения информации о категориях, которые могут быть прикреплены к мероприятиям. Структура данной таблицы приведена в таблице 2.7.

Таблица 2.7 – Структура таблицы Tag

|  |  |  |
| --- | --- | --- |
| Название поля\_ | Тип данных | Описание |
| id | Integer | Уникальный идентификатор категории |
| name | String | Название категории |
| created\_at | DateTime | Дата и время создания категории |
| updated\_at | DateTime | Дата и время последнего обновления |

Таблица Event содержит информацию о мероприятиях, организуемых пользователями. Структура данной таблицы приведена в таблице 2.8.

Таблица 2.8 – Структура таблицы Event

|  |  |  |
| --- | --- | --- |
| Название поля | Тип данных | Описание |
| id | Integer | Уникальный идентификатор мероприятия |
| title | String | Название мероприятия |
| participations | Integer | Внешний ключ на таблицу event\_participations |
| description | Text | Описание мероприятия |
| start\_time | DateTime | Дата и время начала мероприятия |
| tags | Integer | Внешний ключ на таблицу tags |
| end\_time | DateTime | Дата и время завершения мероприятия |
| capacity | Integer | Количество участников |
| location | JSON | Место проведения |
| created\_at | DateTime | Дата и время создания мероприятия |
| updated\_at | DateTime | Дата и время последнего обновления |

Таблица Review хранит отзывы пользователей о мероприятиях. Структура данной таблицы приведена в таблице 2.9.

Таблица 2.9 – Структура таблицы Review

|  |  |  |
| --- | --- | --- |
| Название поля | Тип данных | Описание |
| id | Integer | Уникальный идентификатор отзыва |
| event\_id | Integer | Ссылка на мероприятие |
| user\_id | Integer | Ссылка на пользователя, оставившего отзыв |
| text | Text | Текст отзыва |
| images | Integer | Внешний ключ на таблицу ReviewImage |
| created\_at | DateTime | Дата и время создания отзыва |

Таблица ReviewImage используется для хранения изображений, прикрепленных к отзывам пользователей. Структура данной таблицы приведена в таблице 2.10.

Таблица 2.10 – Структура таблицы ReviewImage

|  |  |  |
| --- | --- | --- |
| Название поля | Тип данных | Описание |
| id | Integer | Уникальный идентификатор изображения |
| review\_id | Integer | Ссылка на отзыв |
| image\_url | String | URL изображения |
| created\_at | DateTime | Дата и время загрузки изображения |

Таблица EventApplication отвечает за хранение заявок пользователей на участие в мероприятиях. Структура данной таблицы приведена в таблице 2.11.

Таблица 2.11 – Структура таблицы EventApplication

|  |  |  |
| --- | --- | --- |
| Название поля | Тип данных | Описание |
| id | Integer | Уникальный идентификатор заявки |
| event\_id | Integer | Ссылка на мероприятие |
| user\_id | Integer | Ссылка на пользователя |
| status | String | Статус заявки (ожидание, подтверждено, отклонено) |
| created\_at | DateTime | Дата и время подачи заявки |

Таблица UserToken используется для хранения токенов пользователей. Структура данной таблицы приведена в таблице 2.12.

Таблица 2.12 – Структура таблицы UserToken

|  |  |  |
| --- | --- | --- |
| Название поля | Тип данных | Описание |
| id | Integer | Уникальный идентификатор токена |
| user\_id | Integer | Ссылка на пользователя |
| token | String | Уникальный токен авторизации |

Таблица CustomUser содержит основную информацию о пользователях. Структура данной таблицы приведена в таблице 2.13.

Таблица 2.13 – Структура таблицы CustomUser

|  |  |  |
| --- | --- | --- |
| Название поля | Тип данных | Описание |
| id | Integer | Уникальный идентификатор пользователя |
| username | String | Имя пользователя |
| email | String | Электронная почта пользователя |
| password | String | Хешированный пароль |
| created\_at | DateTime | Дата и время создания пользователя |
| updated\_at | DateTime | Дата и время последнего обновления |
| telegram\_ username | String | Имя аккаунта в телеграм |
| city | String | Город пользователя |
| about | String | Описание |
| photo | String | Фото пользователя |
| birth\_date | DateTime | Дата рождания |

Каждая таблица имеет четко определенные поля, отражающие определенные аспекты работы web-приложения для организации и участия в мероприятиях.

* + 1. **Архитектура web-приложения**

Архитектура web-приложения представлена в приложении Б.

Пояснение назначения каждого элемента web-приложения представлено в таблице 2.14.

Таблица 2.14 – Назначение элементов архитектурной схемы web-приложения

|  |  |
| --- | --- |
| Элемент | Назначение |
| Frontend Server (React Native) [7] | Обрабатывает статические файлы (Expo Bundle)[8] и обеспечивает взаимодействие с клиентскими устройствами через HTTPS. |
| Backend Server (Django Rest Framework) | Выполняет бизнес-логику, обработку API-запросов и управление связью между клиентами и базой данных. |
| Database Server (PostgreSQL) | Служит для хранения структурированных данных, включая информацию о пользователях, предпочтениях, сообщениях, подписках и других объектах. |
| Elasticsearch [9] | Используется для быстрого полнотекстового поиска по данным. |
| LocalStack [10] | Предоставляет локальную эмуляцию AWS-сервисов [11] для разработки и тестирования. |
| Client Browser (Microsoft Edge) | Получает статические файлы для frontend-приложения и взаимодействует с API через HTTPS. [12] |
| Client Mobile Device (Expo Go) [13] | Позволяет пользователям мобильных устройств взаимодействовать с приложением через React Native Expo. |

Описание протоколов, используемых в приложении, представлено в таблице 2.14.

Таблица 2.14 – Описание используемых протоколов

|  |  |
| --- | --- |
| Протокол | Назначение |
| HTTPS | Обеспечивает безопасное соединение между клиентами, сервером API и frontend-сервером. |
| TCP[14] | Используется для надёжной передачи данных между сервером базы данных PostgreSQL и backend-сервером. |

## **Выводы по разделу**

Таким образом, было спроектировано web-приложение, обладающее следующими особенностями:

1. Поддержка четырех ролей с четко разграниченными правами доступа и функциональными возможностями: гость, участник, создатель, администратор, что обеспечивает безопасность и удобство работы с платформой.

2. Спроектирована база данных web-приложения на основе PostgreSQL, которая состоит из таблиц для хранения данных о пользователях, мероприятиях, заявок на мероприятия, токенов уведомлений, а также отзывов.

3. Web-приложение построено на модульной архитектуре с использованием Django для серверной части, Docker Compose используется для запуска многоконтейнерных приложений, что обеспечивает удобство развертывания и масштабируемость системы. Применение Elasticsearch для быстрого полнотекстового поиска и обработки больших объемов данных. Использование LocalStack для эмуляции AWS-сервисов, что позволяет проводить локальное тестирование и разрабатывать систему без необходимости подключения к облачным сервисам.

**3 Реализация web-приложения**

1. **Обоснование выбора программной платформы**

Для серверной части проекта была выбрана платформа Django Rest Framework (DRF), которая представляет собой мощный и гибкий инструмент для разработки RESTful API [15]. DRF работает на базе Django, что обеспечивает высокую интеграцию с его ORM [16], встроенными механизмами аутентификации, администрирования и инструментами для работы с базами данных.

**3.2 Система управления базами данных PostgreSQL**

Для работы веб-приложения используется PostgreSQL – мощная, надежная и масштабируемая система управления базами данных. Она поддерживает сложные SQL-запросы, транзакции с соответствием стандарту ACID, а также широкий спектр типов данных, включая JSON [17]. Скрипт создания базы данных представлен в приложении В.

## **3.3 Object-Relational Mapping (ORM) Django ORM**

В проекте используется ORM Django , которая является встроенным инструментом фреймворка Django и обеспечивает удобный и интуитивно понятный способ работы с базой данных. Django ORM позволяет разработчику работать с базой данных, используя Python-код вместо написания SQL-запросов. Это упрощает создание, чтение, обновление и удаление записей (операции CRUD) и позволяет сосредоточиться на логике приложения, а не на низкоуровневых запросах. Сопоставление моделей, используемых в Django ORM, с их реальными структурами представлено в таблице 3.1.

Таблица 3.1 – Сопоставление моделей, используемых в Django ORM

|  |  |
| --- | --- |
| Название модели | Название таблицы |
| Tag | tag |
| Event | event |
| Review | review |
| ReviewImage | review\_image |
| EventApplication | event\_application |
| UserToken | user\_token |
| CustomUser | custom\_user |

Код, описывающий модель Tag, приведен в листинге 3.1

class Tag(models.Model):

name = models.CharField(max\_length=255, unique=True, help\_text="Название категории (например, 'Спорт')")

created\_at = models.DateTimeField(auto\_now\_add=True)

Листинг 3.1 – Модель Tag

Модель Tag представляет собой таблицу для хранения категорий. Поле name хранит уникальное название категории, а поле created\_at сохраняет дату и время создания.

Код, описывающий модель Event, приведен в листинге 3.2

class Event(models.Model):

title = models.CharField(max\_length=255, help\_text="Название мероприятия (например, 'Игра в футбол')")

description = models.TextField(blank=True, null=True, help\_text="Описание мероприятия")

city = models.CharField(max\_length=254, help\_text="Город проведения мероприятия")

location = models.JSONField(help\_text="Координаты места проведения мероприятия")

start\_time = models.DateTimeField(help\_text="Дата и время начала мероприятия")

end\_time = models.DateTimeField(help\_text="Дата и время окончания мероприятия")

tags = models.ManyToManyField(Tag, related\_name="events", help\_text="Категории мероприятия")

created\_by = models.ForeignKey(

"users.CustomUser", on\_delete=models.CASCADE, related\_name="events", help\_text="Пользователь, создавший мероприятие")

participants = models.ManyToManyField(

"users.CustomUser", related\_name="participated\_events", blank=True, help\_text="Список участников мероприятия"

)

capacity = models.PositiveIntegerField(

default=0, help\_text="Максимальное количество участников для мероприятия"

)

created\_at = models.DateTimeField(auto\_now\_add=True)

Листинг 3.2 – Модель Event

Модель Event описывает мероприятие. Включает поля title, description, city для хранения названия, описания и города. Также используются location для координат, start\_time и end\_time для времени начала и окончания. Взаимосвязь с моделью Tag осуществляется через поле tags, с пользователем – через created\_by, а список участников хранится в поле participants. Поле capacity ограничивает число участников.

Код, описывающий модель Review, приведен в листинге 3.3

class Review(models.Model):

event = models.ForeignKey(

"Event", on\_delete=models.CASCADE, related\_name="reviews", help\_text="Мероприятие, к которому относится отзыв")

user = models.ForeignKey(

User, on\_delete=models.CASCADE, related\_name="reviews", help\_text="Пользователь, оставивший отзыв")

text = models.TextField(help\_text="Текст отзыва")

created\_at = models.DateTimeField(auto\_now\_add=True, help\_text="Дата и время создания отзыва")

Листинг 3.3 – Модель Review

Модель Review описывает отзыв на мероприятие. Поле event связывает отзыв с конкретным мероприятием, поле user – с пользователем, оставившим отзыв, а поле text содержит сам текст отзыва. Поле created\_at фиксирует дату и время создания отзыва.

Код, описывающий модель ReviewImage, приведен в листинге 3.4.

class ReviewImage(models.Model):

review = models.ForeignKey(

Review, on\_delete=models.CASCADE, related\_name="images"

)

image\_url = models.URLField(help\_text="Ссылка на изображение")

created\_at = models.DateTimeField(auto\_now\_add=True)

Листинг 3.4 – Модель ReviewImage

Модель ReviewImage используется для хранения изображений, прикрепленных к отзыву. Поле review связывает изображение с отзывом, а image\_url содержит URL изображения.

Код, описывающий модель EventApplication, приведен в листинге 3.5.

class EventApplication(models.Model):

STATUS\_CHOICES = [

("pending", "Pending"),

("accepted", "Accepted"),

("rejected", "Rejected"),

]

event = models.ForeignKey(

"events.Event", on\_delete=models.CASCADE, related\_name="applications", help\_text="Мероприятие, к которому относится заявка"

)

user = models.ForeignKey(

"users.CustomUser", on\_delete=models.CASCADE, related\_name="applications", help\_text="Пользователь, подавший заявку"

)

status = models.CharField(

max\_length=10, choices=STATUS\_CHOICES, default="pending", help\_text="Статус заявки"

)

created\_at = models.DateTimeField(auto\_now\_add=True, help\_text="Дата и время подачи заявки")

updated\_at = models.DateTimeField(auto\_now=True, help\_text="Дата и время последнего обновления заявки")

Листинг 3.5 – Модель EventApplication

Модель EventApplication описывает заявку пользователя на участие в мероприятии. Поле status хранит статус заявки. Поля created\_at и updated\_at отслеживают дату подачи заявки и дату последнего обновления соответственно.

Код, описывающий модель UserToken, приведен в листинге 3.6.

class UserToken(models.Model):

user\_id = models.IntegerField(unique=True)

token = models.CharField(max\_length=255)

Листинг 3.6 – Модель UserToken

Модель UserToken хранит токен пользователя, связанный с его user\_id. Поле token используется для хранения уникального токена для аутентификации пользователя.

Код, описывающий модель CustomUser, приведен в листинге 3.7.

class CustomUser(AbstractUser):

photo = models.URLField(max\_length=500, blank=True, null=True, help\_text="Ссылка на фото в S3")

city = models.CharField(max\_length=255, blank=True, null=True, help\_text="Город пользователя")

birth\_date = models.DateField(blank=True, null=True, help\_text="Дата рождения пользователя")

tags = models.ManyToManyField(

"events.Tag", related\_name="users\_with\_hobby", blank=True, help\_text="Список увлечений пользователя, совпадающий с категориями"

)

about = models.TextField(blank=True, null=True, help\_text="Описание пользователя")

telegram\_username = models.CharField(

max\_length=255, blank=True, null=True, help\_text="Имя пользователя в Telegram"

)

Листинг 3.7 – Модель CustomUser

Модель CustomUser представляет собой расширенную модель пользователя. Включает поля для хранения информации о фотографии, городе, дате рождения, увлечениях и описании пользователя. Также есть поле для имени пользователя в Telegram.

## **3.4 Программные библиотеки**

В процессе разработки серверной части web-приложения для обеспечения её функциональности и повышения эффективности работы системы были использованы программные библиотеки, представленные в таблице 3.2.

Таблица 3.2 – Программные библиотеки серверной части

|  |  |  |
| --- | --- | --- |
| Библиотека | Версия | Назначение |
| Django | ^5.1.4 | Фреймворк для разработки web-приложений. |
| django-ninja | ^1.3.0 | Инструмент для создания API в Django. |
| psycopg2-binary | ^2.9.10 | Драйвер для работы с базой данных PostgreSQL. |

Продолжение таблицы 3.2.

|  |  |  |
| --- | --- | --- |
| Библиотека | Версия | Назначение |
| djangorestframework-simplejwt | ^5.3.1 | Библиотека для работы с JWT [18] (JSON Web Tokens) для авторизации. |
| django-environ | ^0.11.2 | Библиотека для работы с переменными окружения. |
| djangorestframework | ^3.15.2 | Расширение для создания REST API в Django. |
| boto3 | ^1.35.90 | Клиент AWS для Python. |
| botocore | ^1.35.90 | Библиотека для работы с AWS. |
| elasticsearch-dsl | ^8.17.0 | Высокоуровневый клиент для Elasticsearch. |
| elasticsearch | ^8.17.0 | Библиотека для работы с Elasticsearch. |
| pydantic | ^2.10.4 | Валидация данных с поддержкой email. |
| black | ^24.10.0 | Инструмент для форматирования кода Python. |
| requests | ^2.32.3 | HTTP-клиент для работы с запросами. |

В процессе разработки клиентской части web-приложения были задействованы программные библиотеки, представленные в таблице 3.3.

Таблица 3.3 – Программные библиотеки клиентской части

|  |  |  |
| --- | --- | --- |
| Библиотека | Версия | Назначение |
| @expo/vector-icons | ^14.0.2 | Набор иконок для использования в приложении. |
| @react-navigation/native | ^7.0.0 | Библиотека для навигации в React Native приложениях. |
| expo | ^52.0.17 | Фреймворк для создания приложений с использованием React Native. |
| expo-constants | ~17.0.3 | Предоставляет системную информацию о приложении и устройстве. |
| expo-font | ~13.0.1 | Подключение пользовательских шрифтов в приложении. |
| expo-image-picker | ~16.0.3 | Инструмент для выбора изображений из галереи или камеры. |
| expo-linking | ~7.0.3 | Библиотека для обработки глубоких ссылок и универсальных ссылок. |
| expo-router | ^4.0.15 | Система маршрутизации для Expo приложений. |
| expo-secure-store | ~14.0.0 | Безопасное хранение данных на устройстве. |
| expo-splash-screen | ~0.29.16 | Управление экраном загрузки в Expo приложениях. |

Продолжение таблицы 3.3.

|  |  |  |
| --- | --- | --- |
| expo-status-bar | ~2.0.0 | Управление статус-баром в приложении. |
| expo-system-ui | ~4.0.5 | Изменение внешнего вида системного интерфейса устройства. |
| expo-web-browser | ~14.0.1 | Открытие веб-страниц в браузере устройства. |
| react | 18.3.1 | Библиотека для построения пользовательских интерфейсов. |
| react-dom | 18.3.1 | Поддержка React для работы в браузере. |
| react-native | 0.76.3 | Фреймворк для создания мобильных приложений. |
| react-native-date-picker | ^5.0.8 | Компонент для выбора даты и времени. |
| react-native-gesture-handler | ~2.20.2 | Поддержка жестов в React Native приложениях. |
| react-native-image-picker | ^7.2.3 | Библиотека для выбора изображений с устройства. |
| react-native-keychain | ^9.2.2 | Хранение данных авторизации в защищённом хранилище устройства. |
| react-native-linear-gradient | ^2.8.3 | Реализация градиентных заливок. |
| react-native-maps | 1.18.0 | Поддержка карт и геолокации в приложении. |
| react-native-modal | ^13.0.1 | Инструмент для создания модальных окон. |
| react-native-modal-datetime-picker | ^18.0.0 | Выбор даты и времени в модальном окне. |
| react-native-paper | ^5.12.5 | UI-компоненты в стиле Material Design. |
| react-native-picker-select | ^9.3.1 | Компонент для создания выпадающих списков. |
| react-native-reanimated | ~3.16.1 | Библиотека для анимации в React Native приложениях. |
| react-native-safe-area-context | 4.12.0 | Управление безопасными областями устройства. |
| react-native-screens | ~4.1.0 | Оптимизация отображения экранов в React Native. |
| react-native-snap-carousel | ^1.3.1 | Компонент для создания каруселей. |
| react-native-swiper | ^1.6.0 | Компонент для создания слайдеров. |
| react-native-web | ^0.19.13 | Поддержка React Native приложений в браузере. |

Программные библиотеки позволяют упростить реализацию web-приложения.

## **3.5 Структура серверной части**

Основные компоненты структуры серверной части включают в себя несколько ключевых элементов, которые обеспечивают эффективную работу приложения:

1. Маршрутизаторы – управляют маршрутами и направляют запросы к соответствующим контроллерам.
2. Контроллеры – обрабатывают запросы от клиента, выполняют бизнес-логику через сервисы и возвращают ответы.
3. Сервисы – предоставляют логику работы, которая повторно используется в контроллерах и других компонентах системы.

**В таблице 3.4 приведён список директорий серверной части проекта.**

Таблица 3.4 – Директории серверной части проекта

|  |  |
| --- | --- |
| Директория | Назначение |
| backend | Содержит настройки проекта (settings.py), маршруты (urls.py), точки запуска (wsgi.py, asgi.py). |
| events | Реализует функционал мероприятий: модели (Event, Review), маршруты, контроллеры, бизнес-логику. |
| users | Управление пользователями: регистрация, аутентификация, профили, JWT-логика. |
| notifications | Реализация уведомлений (регистрация токенов, отправка мероприятий). |
| utils | Вспомогательные модули: JWT-утилиты, интеграция с внешними сервисами. |

**Эта структура организована для обеспечения логики, удобной навигации и масштабируемости серверной части. Таблица соответствия маршрутов контроллерам в исходном коде представлена в таблице 3.5.**

Таблица 3.5 – Маршут и метод контроллера

|  |  |  |
| --- | --- | --- |
| Метод | Маршрут | Метод контроллера |
| POST | api/auth/register/ | register\_user |
| POST | api/auth/login/ | login |
| PATCH | api/auth/profile/ | update\_profile |
| GET | api/events/all/ | get\_all\_events |
| POST | api/events/{event\_id}/apply/ | apply\_to\_event |
| DELETE | api/events/{event\_id}/application/ | delete\_application |
| POST | api/events/{event\_id}/reviews/ | create\_review |
| GET | api/auth/profile/{user\_id}/ | get\_user\_profile |
| GET | api/events/search\_and\_filter/ | search\_and\_filter\_events |
| POST | api/events/create-event/ | create\_event |
| PUT | api/events/{event\_id}/ | update\_event |
| GET | api/events/{event\_id}/applications/ | list\_applications |
| POST | api/events/{event\_id}/applications/{application\_id}/accept/ | accept\_application |
| POST | api/events/{event\_id}/applications/{application\_id}/reject/ | reject\_application |

Данные маршруты используют формат JSON для обмена информацией между клиентом и сервером.

## **3.6 Реализация функционала для пользователя с ролью «Гость»**

### **3.6.1** **Регистрация**

Для регистрации нового пользователя используется маршрут POST api/auth/register/. Реализация контроллера приведена в листинге 3.8.

@router.post("/register/")

def register\_user(request, payload: RegisterSchema):

"""

Регистрация нового пользователя. Проверяет уникальность username и email.

"""

if CustomUser.objects.filter(username=payload.username).exists():

return JsonResponse({"error": "Username уже занят"}, status=400)

if CustomUser.objects.filter(email=payload.email).exists():

return JsonResponse({"error": "Email уже зарегистрирован"}, status=400)

user = CustomUser.objects.create(

username=payload.username,

password=make\_password(payload.password),

email=payload.email,

city=payload.city,

birth\_date=payload.birth\_date,)

return JsonResponse({"user\_id": user.id}, status=201)

Листинг 3.8 – Реализация контроллера register\_user

Контроллер принимает данные: username, password, email, city, birth\_date. Сначала выполняется проверка существования пользователя с таким же username или email через CustomUser.objects.filter(). Если дубликаты найдены, возвращается ошибка с кодом 400. Если проверка пройдена, пароль хешируется с помощью make\_password(), и создается новая запись в базе данных. Успешный ответ содержит ID пользователя и статус 201.

### **3.6.2** **Аутентификация**

Аутентификация реализована через маршрут POST api/auth/login/ с использованием библиотеки SimpleJWT. Настройки JWT добавлены в settings.py. Конфигурация SimpleJWT представлена в листинге 3.9.

SIMPLE\_JWT = {

"ACCESS\_TOKEN\_LIFETIME": timedelta(days=1),

"REFRESH\_TOKEN\_LIFETIME": timedelta(days=14),

"ALGORITHM": "HS256",

"SIGNING\_KEY": SECRET\_KEY,

"AUTH\_HEADER\_TYPES": ("Bearer",),}

Листинг 3.9 – Конфигурация SimpleJWT

При успешной аутентификации возвращаются два токена: access (для доступа к API) и refresh (для обновления сессии). Токены включают идентификатор пользователя (user\_id). Если учетные данные неверны, возвращается ошибка 401.

## **3.7 Реализация функционала для пользователя с ролью «Участник»**

### **3.7.1 Редактирование пользовательских данных**

Маршрут PATCH api/auth/profile/ позволяет обновить профиль. Реализация контроллера update\_profile приведена в листинге 3.10.

@router.patch("/profile/", auth=JWTAuth())

def update\_profile(request, payload: UpdateProfileSchema):

user = request.auth

if payload.email:

user.email = payload.email

if payload.city:

user.city = payload.city

if payload.birth\_date:

user.birth\_date = datetime.strptime(payload.birth\_date, "%Y-%m-%d").date()

user.save()

return {"message": "Профиль обновлен"}

Листинг 3.10 – Реализация контроллера update\_profile

Метод обновляет только переданные поля. Дата преобразуется из строки в объект date. При отсутствии прав доступа возвращается 403.

### **3.7.2 Просмотр мероприятий**

Маршрут GET api/events/all/ возвращает список мероприятий. Реализация контроллера get\_all\_events приведена в листинге 3.11.

@router.get("/all/")

def get\_all\_events(request, is\_archived: Optional[bool] = None):

now = datetime.now(timezone.utc)

events = Event.objects.prefetch\_related("tags", "participants")

if is\_archived is None:

events = events.filter(end\_time\_\_gte=now)

elif is\_archived:

events = events.filter(end\_time\_\_lt=now)

return {

"events": [

{

"id": event.id,

"title": event.title,

"start\_time": event.start\_time,}

for event in events]}

Листинг 3.11 – Реализация контроллера get\_all\_events

Параметр is\_archived фильтрует мероприятия по статусу (активные/архивные). Если параметр не указан, возвращаются только активные мероприятия.

### **3.7.3 Подача заявки на участие**

Для подачи заявки используется маршрут POST api/events/{event\_id}/apply/. Реализация контроллера apply\_to\_event приведена в листинге 3.12.

@router.post("/{event\_id}/apply/", auth=JWTAuth())

def apply\_to\_event(request, event\_id: int):

event = get\_object\_or\_404(Event, id=event\_id)

if event.participants.filter(id=request.auth.id).exists():

return {"error": "Вы уже участник"}, 400

EventApplication.objects.create(user=request.auth, event=event, status="pending")

return {"message": "Заявка подана"}

Листинг 3.12 – Реализация контроллера apply\_to\_event

Перед созданием заявки проверяется, не является ли пользователь уже участником. Статус заявки по умолчанию – pending.

### **3.7.4 Отмена заявки**

Маршрут DELETE api/events/{event\_id}/application/ удаляет заявку (листинг 3.13). Реализация контроллера delete\_application приведена в листинге 3.13.

@router.delete("/{event\_id}/application/", auth=JWTAuth())

def delete\_application(request, event\_id: int):

application = get\_object\_or\_404(EventApplication, event\_id=event\_id, user=request.auth)

application.delete()

return {"message": "Заявка удалена"}

Листинг 3.13 – Реализация контроллера delete\_application

Если заявка не найдена, возвращается ошибка 404.

### **3.7.5 Добавление мероприятия в Google Календарь**

Добавление мероприятия в Google Календарь реализованно на клиентской части с помощью функции handleAddToGoogleCalendar. Реализация функции delete\_application приведена в листинге 3.14.

const handleAddToGoogleCalendar = () => {

const { title, description, start\_time, end\_time, location } = selectedEvent;

const startTime = new Date(start\_time).toISOString().replace(/-|:|\.\d+/g, "");

Продолжение листинга 3.14

const endTime = new Date(end\_time).toISOString().replace(/-|:|\.\d+/g, "");

const googleCalendarUrl = `https://www.google.com/calendar/render?action=TEMPLATE&text=${encodeURIComponent(

title

)}&details=${encodeURIComponent(description)}&location=${encodeURIComponent(

location?.lat + ", " + location?.lng

)}&dates=${startTime}/${endTime}`;

Linking.openURL(googleCalendarUrl);

};

Листинг 3.14 – Реализация функции delete\_application

Функция handleAddToGoogleCalendar предназначена для создания и открытия ссылки, которая добавляет мероприятие в Google Calendar на основе данных выбранного мероприятия. Она извлекает из объекта selectedEvent название (title), описание (description), время начала (start\_time), время окончания (end\_time) и координаты локации (location). Даты преобразуются в формат, совместимый с Google Календарь.

### **3.7.6 Создание отзыва**

Маршрут POST api/events/{event\_id}/reviews/ позволяет оставить отзыв. Реализация контроллера delete\_application приведена в листинге 3.15.

@router.post("/{event\_id}/reviews/", auth=JWTAuth())

def create\_review(request, event\_id: int, text: str = Form(...),

images: List[UploadedFile] = File(None)):

event = get\_object\_or\_404(Event, id=event\_id)

if not event.participants.filter(id=request.auth.id).exists():

return {"error": "Вы не участвовали в мероприятии"}, 400

review = Review.objects.create(event=event, user=request.auth, text=text)

if images:

for image in images:

ReviewImage.objects.create(review=review, image=image)

return {"review\_id": review.id}

Листинг 3.15 – Реализация контроллера delete\_application

Отзыв могут оставлять только участники мероприятия. Изображения сохраняются в связанной модели ReviewImage.

### **3.7.7 Просмотр профиля пользователя**

Маршрут GET api/auth/profile/{user\_id}/ возвращает публичные данные пользователя. Реализация контроллера get\_user\_profile приведена в листинге 3.16.

@router.get("/profile/{user\_id}/")

def get\_user\_profile(request, user\_id: int):

user = get\_object\_or\_404(CustomUser, id=user\_id)

return {

"username": user.username,

"city": user.city,

"birth\_date": user.birth\_date,

}

Листинг 3.16 – Реализация контроллера get\_user\_profile

Контроллер get\_user\_profile предназначен для получения информации о профиле пользователя по его уникальному идентификатору. При обработке GET-запроса по маршруту /profile/{user\_id}/ (где user\_id – целочисленный идентификатор пользователя) система выполняет поиск пользователя в базе данных с использованием модели CustomUser. Если пользователь с указанным ID не существует, автоматически возвращается HTTP-ответ с кодом 404 (Not Found).

### **3.7.8 Поиск и фильтрация мероприятий**

Маршрут GET api/events/search\_and\_filter/ реализует поиск. Реализация контроллера search\_and\_filter\_events приведена в листинге 3.17.

@router.get("/search\_and\_filter/")

def search\_and\_filter\_events(request, query: Optional[str] = None, tags: Optional[str] = None):

events = Event.objects.all()

if query:

events = events.filter(Q(title\_\_icontains=query) | Q(description\_\_icontains=query))

if tags:

tag\_list = tags.split(",")

events = events.filter(tags\_\_name\_\_in=tag\_list)

return {"events": events.values("id", "title")}

Листинг 3.17 – Реализация контроллера search\_and\_filter\_events

Контроллер search\_and\_filter\_events предназначен для поиска и фильтрации мероприятий на основе текстового запроса и категорий. При обработке GET-запроса по маршруту /search\_and\_filter/ система принимает два необязательных параметра: query (текстовый поиск) и tags (список категорий через запятую). Если указан параметр query, выполняется поиск по частичному совпадению текста в полях title (название мероприятия) и description (описание) с использованием логического оператора ИЛИ. При наличии параметра tags мероприятия фильтруются по принадлежности к одному из указанных категорий. Результат возвращается в формате JSON, содержащем массив мероприятий с полями id и title. Если параметры не переданы, маршрут возвращает полный список мероприятий.

### **3.7.9 Создание мероприятия**

Маршрут POST api/events/create-event/ создает новое мероприятие. Реализация контроллера create\_event приведена в листинге 3.18.

@admin.register(models.User)

class UsersAdmin(UserAdmin):

fieldsets = (

(None, {

"fields": ("email", "password")

}),

("Персональная информация", {

"fields": ("is\_hidden", "preferences")

}),

)

list\_display = ("email", "is\_active")

Листинг 3.18 – Реализация контроллера create\_event

Контроллер create\_event предназначен для создания нового мероприятия авторизованным пользователем. При обработке POST-запроса по маршруту /create-event/ система проверяет валидность JWT-токена в заголовке запроса (аутентификация через JWTAuth()). После успешной проверки создается объект мероприятия Event в базе данных с использованием данных из тела запроса.

## **3.8 Реализация функционала для пользователя с ролью «Создатель»**

### **3.8.1 Редактирование мероприятия**

Маршрут PUT api/events/{event\_id}/ обновляет данные мероприятия. Реализация контроллера update\_event приведена в листинге 3.19.

@router.put("/{event\_id}/", auth=JWTAuth())

def update\_event(request, event\_id: int, payload: EventUpdateSchema):

event = get\_object\_or\_404(Event, id=event\_id, created\_by=request.auth)

for field, value in payload.dict().items():

setattr(event, field, value)

event.save()

return {"message": "Мероприятие обновлено"}

Листинг 3.19 – Реализация контроллера update\_event

Контроллер PUT update\_event позволяет авторизованному пользователю обновлять данные созданного им мероприятия.

При обработке PUT-запроса по маршруту /{event\_id}/ система проверяет JWT-токен для аутентификации пользователя, после чего выполняет поиск мероприятия по event\_id с дополнительным условием: мероприятие должно принадлежать текущему пользователю (поле created\_by). Если мероприятие не найдено или пользователь не является его создателем, возвращается HTTP-статус 404.

### **3.8.2 Просмотр заявок**

Маршрут GET api/events/{event\_id}/applications/ возвращает список заявок. Реализация контроллера list\_applications приведена в листинге 3.20.

@router.get("/{event\_id}/applications/", auth=JWTAuth())

def list\_applications(request, event\_id: int):

event = get\_object\_or\_404(Event, id=event\_id, created\_by=request.auth)

applications = event.applications.all()

return {"applications": [app.user.username for app in applications]}

Листинг 3.20 – Реализация контроллера list\_applications

Контроллер list\_applications предоставляет возможность авторизованному пользователю получить список заявок на участие в конкретном мероприятии, созданном им самим. При обработке GET-запроса по маршруту /{event\_id}/applications/ система выполняет следующие действия.

### **3.8.3 Принятие заявок**

Маршрут POST api/events/{event\_id}/applications/{application\_id}/accept/ меняет статус заявки. Реализация контроллера accept\_application приведена в листинге 3.21.

@router.post("/{event\_id}/applications/{application\_id}/accept/", auth=JWTAuth())

def accept\_application(request, event\_id: int, application\_id: int):

application = get\_object\_or\_404(EventApplication, id=application\_id, event\_id=event\_id)

application.status = "accepted"

application.save()

return {"message": "Заявка принята"}

Листинг 3.21 – Реализация контроллера accept\_application

Функция accept\_application позволяет авторизованным пользователям принимать заявки на участие в мероприятии путем отправки POST-запроса по маршруту /{event\_id}/applications/{application\_id}/accept/.

После проверки валидности JWT-токена система ищет заявку (EventApplication) по связке event\_id и application\_id, и если заявка не найдена, возвращает статус 404 Not Found.

Для найденной заявки статус изменяется на “accepted”, и изменения сохраняются в базе данных. В ответе возвращается сообщение “Заявка принята”.

### **3.8.4 Отклонение заявок**

Маршут POST api/events/{event\_id}/applications/{application\_id}/reject/ обновляет статус заявки на “rejected” с дополнительными проверками безопасности. Реализация контроллера reject\_application приведена в листинге 3.22.

@router.post(

"/{event\_id}/applications/{application\_id}/reject/",

response=EventApplicationResponseSchema,

auth=JWTAuth(),

)

def reject\_application(request, event\_id: int, application\_id: int):

user = request.auth

try:

event = Event.objects.get(id=event\_id, created\_by=user)

application = EventApplication.objects.get(

id=application\_id, event=event, status="pending"

)

application.status = "rejected"

application.save()

return EventApplicationResponseSchema(

id=application.id,

event\_id=application.event.id,

user\_id=application.user.id,

username=application.user.username,

photo=application.user.photo if application.user.photo else None,

status=application.status,

created\_at=application.created\_at.isoformat(),

updated\_at=application.updated\_at.isoformat(),

)

except Event.DoesNotExist:

return 403, {"detail": "You are not authorized to manage this event."}

except EventApplication.DoesNotExist:

return 404, {"detail": "Application not found or already processed."}

Листинг 3.22 – Реализация контроллера reject\_application

Перед изменением статуса выполняется проверка: пользователь должен быть создателем мероприятия (created\_by=user), а заявка – находиться в статусе “pending”. При успешном обновлении возвращается полный объект заявки в формате EventApplicationResponseSchema. Ошибки обрабатываются аналогично reject\_application: код 403 возвращается при отсутствии прав на управление мероприятием, 404 – при несуществующей или уже обработанной заявке.

## **3.9 Реализация функционала для пользователя с ролью «Администратор»**

### **3.9.1 Управление пользователями**

Пользователь с ролью «Администратор» получает доступ к редактированию списка всех пользователей и мероприятий через административную панель Django. Для этого используются кастомизированные классы ModelAdmin, которые расширяют стандартный интерфейс управления данными, добавляя фильтрацию, поиск и автозаполнение связанных полей.

Класс CustomUserAdmin настраивает интерфейс для работы с моделью пользователя CustomUser. В списке пользователей отображаются основные данные: имя пользователя (username), электронная почта (email), город (city), дата рождения (birth\_date) и Telegram-username. Фильтрация по городу позволяет группировать пользователей по географическому признаку, а поиск работает по полям username, email и telegram\_username, что упрощает поиск конкретных записей. Администратор может редактировать, просматривать и удалять пользователей напрямую через административную панель. Реализация класса представлена в листинге 3.23.

@admin.register(CustomUser)

class CustomUserAdmin(admin.ModelAdmin):

list\_display = ("username", "email", "city", "birth\_date", "telegram\_username")

list\_filter = ("city",)

search\_fields = ("username", "email", "telegram\_username")

Листинг 3.23 – Реализация класса CustomUserAdmin

В данном коде класс CustomUsersAdmin кастомизирует стандартный интерфейс для управления пользователями. Поле fieldsets определяет, какие группы полей отображаются в форме редактирования, включая обязательные поля, такие как электронная почта и пароль. Поле list\_display задаёт, какие данные отображаются в списке пользователей, чтобы администратор мог быстро видеть ключевую информацию о пользователях. Такой подход упрощает управление пользователями через административную панель и делает интерфейс более удобным для работы.

### **3.9.2 Управление мероприятиями**

Для управления мероприятиями используется класс EventAdmin, связанный с моделью Event. В списке мероприятий отображаются заголовок (title), город (city), время начала (start\_time), время окончания (end\_time), вместимость (capacity) и создатель (created\_by). Фильтрация по городу и категориям (tags) позволяет администратору быстро находить мероприятия в определенных локациях или с конкретными метками.

Поиск работает по заголовку, городу и описанию (description), что помогает локализовать нужные мероприятия даже в большом списке. Поля tags, created\_by и participants используют автозаполнение: при редактировании мероприятия администратор может начать вводить название категории, имя пользователя или участника, и система предложит подходящие варианты. Это ускоряет процесс редактирования и снижает вероятность ошибок. Реализация класса EventAdmin представлена в листинге 3.24.

@admin.register(Event)

class EventAdmin(admin.ModelAdmin):

list\_display = ("title", "city", "start\_time", "end\_time", "capacity", "created\_by")

list\_filter = ("city", "tags")

search\_fields = ("title", "city", "description")

autocomplete\_fields = ("tags", "created\_by", "participants")

Листинг 3.24 – Реализация класса EventAdmin

Кастомизация интерфейса через list\_display обеспечивает четкое отображение ключевых данных. Автозаполнение (autocomplete\_fields) в EventAdmin связано с другими моделями. Например, при выборе категории для мероприятия администратор может начать вводить его название, и система автоматически предложит существующие варианты. Это особенно удобно при работе с большим количеством категорий.

Таким образом, администратор получает инструменты для полного управления пользователями и мероприятиями: от поиска и фильтрации до массового редактирования и удаления. Интеграция автозаполнения и гибких фильтров делает администрирование эффективным даже в условиях больших объемов данных.

## **3.10 Структура клиентской части**

Клиентская часть приложения реализована с использованием компонентного подхода. Основная логика и элементы пользовательского интерфейса размещены в директории app. Директории представлены в таблице 3.6.

Таблица 3.6 – Основные директории проекта и их назначение

|  |  |
| --- | --- |
| Директория | Назначение |
| app/(tabs) | Основные вкладки приложения: главный экран (index.tsx), карта (map.tsx). |
| app/profile | Страницы профиля пользователя: динамический профиль ([userId].tsx), настройки. |
| app | Основные страницы: вход (login.tsx), создание/редактирование мероприятий (CreateEventScreen.tsx, EditEventScreen.tsx). |
| assets | Ресурсы: изображения (images/), шрифты (fonts/), иконки. |
| components | Переиспользуемые компоненты: навигация (TabBarIcon.tsx), UI-элементы (Collapsible.tsx, ThemedText.tsx). |
| constants | Константы: цвета интерфейса (Colors.ts). |

Продолжение таблицы 3.6.

|  |  |
| --- | --- |
| Директория | Назначение |
| hooks | Кастомные хуки: управление темой (useColorScheme.ts), стилями (useThemeColor.ts). |
| screens | Страницы приложения: детали мероприятия (EventDetailsScreen.tsx), фильтры (FilterScreen.tsx). |
| utils | Вспомогательные модули: API-запросы (apiUtils.ts), мок-данные (mockEvents.ts). |

**Таблица соответствия маршрутов и компонентов представлена в таблице 3.7.**

Таблица 3.7 – Маршруты и компоненты страниц

|  |  |  |  |
| --- | --- | --- | --- |
| Компонент страницы | Маршрут | Роль | Назначение компонента |
| login.tsx | /login | Все пользователи | Страница аутентификации пользователя. |
| index.tsx (в tabs) | /tabs | Все пользователи | Главный экран с лентой мероприятий. |
| map.tsx (в tabs) | /tabs/map | Все пользователи | Интерактивная карта с мероприятиями. |
| [userId].tsx | /profile/:userId | Авторизованные пользователи | Просмотр профиля другого пользователя. |
| CreateEventScreen.tsx | /create-event | Авторизованные пользователи | Создание нового мероприятия. |
| EditEventScreen.tsx | /edit-event/:id | Авторизованные пользователи | Редактирование существующего мероприятия. |
| EventDetailsScreen.tsx | /event/:id | Все пользователи | Детализация мероприятия, участники, комментарии. |

Помимо маршрутов и страниц, приложение включает множество компонентов, которые обеспечивают функциональность и удобство использования клиентской части. В таблице 3.8 представлено описание всех остальных компонентов приложения и их назначение.

Таблица 3.8 – Описание компонентов

|  |  |
| --- | --- |
| Компонент | Назначение |
| TabBarIcon | Иконки для навигационной панели (нижнее меню). |
| Collapsible | Сворачиваемый блок для скрытия/отображения контента. |

Продолжение таблицы 3.8.

|  |  |
| --- | --- |
| Компонент | Назначение |
| EventMarker | Маркер мероприятия на карте с обработкой кликов и анимацией. |
| ExternalLink | Безопасное открытие внешних ссылок с валидацией URL. |
| FullscreenGallery | Галерея изображений в полноэкранном режиме. |
| ParallaxScrollView | Скролл с параллакс-эффектом для анимированного контента. |
| ThemedText | Текст, адаптирующийся к теме приложения (темный/светлый режим). |
| ThemedView | Контейнер с поддержкой темы (фоны, границы, тени). |
| NotificationsProvider | Управление push-уведомлениями и их отображением. |
| NavigationButtons | Кнопки навигации (назад/вперед) для перехода между экранами. |

Эти компоненты обеспечивают ключевые функции приложения, включая взаимодействие с мероприятиями (создание, редактирование, просмотр деталей), работу с картой (маркеры мероприятий, фильтрация), управление профилями пользователей, поддержку уведомления, а также навигацию между экранами. Дополнительные функции, такие как анимированная прокрутка и галерея изображений, повышают удобство использования и визуальную привлекательность клиентской части.

## **3.11 Выводы по разделу**

Таким образом, было разработано web-приложение для организации и участия в мероприятиях со следующими особенностями:

1. Использована программная платформа Django Rest Framework для разработки. В качестве системы управления базами данных выбрана реляционная PostgreSQL. Django ORM позволила автоматизировать взаимодействие с БД.

2. Применены технологии для реализации функциональности, а также для аутентификации использовалась библиотека SimpleJWT). Это позволило повысить безопасность и производительность.

3. Клиентская часть разработана с использованием Expo React Native. Были применены современные библиотеки, такие как react-native-maps для работы с картами,expo-image-picker для работы с изображениями. Компонентный подход позволил добиться высокой модульности и переиспользуемости кода. Общее количество реализованных функций составило 17.

# **4 Тестирование web-приложения**

**4.1 Функциональное тестирование**

Для проверки корректности работы всех функций разработанного веб-приложения было проведено ручное тестирование с использованием Postman [12] для отправки запросов на сервер. Описание, ожидаемый результат и итоги тестирования представлены в таблице 4.1.

Таблица 4.1 – Описание ручного тестирования веб-приложения

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Функция | Описание теста | Ожидаемый результат | Результат теста |
| 1 | Регистрация пользователя | Отправить POST-запрос на api/auth/register/ с телом: { "name": "test\_user", "email": "test@example.com", "password": "Test1234!" }. | В таблице users появляется запись с email = "test@example.com", role = "user", name = "test\_user", password, created\_at и updated\_at (текущее время). Сервер возвращает 201, данные пользователя, access token (в теле), refresh token (в cookies). | Совпадает с ожидаемым. |
| 2 | Аутентификация | Отправить POST-запрос на api/auth/login/ с телом: { "email": "test@example.com", "password": "Test1234!" }. | Сервер возвращает 200, access token (в теле), refresh token (в cookies). Проверить валидность токенов через запрос к защищенному маршруту (например, api/auth/profile/). | Совпадает с ожидаемым. |
| 3 | Обновление профиля | Отправить PATCH-запрос на api/auth /profile/ с заголовком Authorization: Bearer <access\_token> и телом: { "name": "new\_name" }. | В таблице users обновляется name = "new\_name", updated\_at изменяется. Сервер возвращает 200 и обновленные данные. | Совпадает с ожидаемым. |
| 4 | Получение всех мероприятий | Отправить GET-запрос на api/events/all/. | Сервер возвращает 200 и массив мероприятий с полями id, title, start\_time, location. | Совпадает с ожидаемым. |
| 5 | Подача заявки на мероприятие | Отправить POST-запрос на api/events /1/apply/ с заголовком Authorization: Bearer <access\_token>. | В таблице applications создается запись с event\_id = 1 и user\_id текущего пользователя. Сервер возвращает 201 и данные заявки. | Совпадает с ожидаемым. |

Продолжение таблицы 4.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Функция | Описание теста | Ожидаемый результат | Результат теста |
| 6 | Удаление заявки | Отправить DELETE-запрос на api/events/1/application/ с заголовком Authorization: Bearer <access\_token>. | Запись в таблице applications удаляется. Сервер возвращает 204. | Совпадает с ожидаемым. |
| 7 | Создание отзыва | Отправить POST-запрос на api/events/1/reviews/ с заголовком Authorization: Bearer <access\_token> и телом: { "text": "Отличное мероприятие!" }. | В таблице reviews создается запись с event\_id = 1 и user\_id текущего пользователя. Сервер возвращает 201 и данные отзыва. | Совпадает с ожидаемым. |
| 8 | Получение профиля | Отправить GET-запрос на api/auth/profile/1/. | Сервер возвращает 200 и данные пользователя с user\_id = 1 (имя, email, мероприятия). | Совпадает с ожидаемым. |
| 9 | Поиск и фильтрация мероприятий | Отправить GET-запрос на api/events/search\_and\_filter/?title=конференция&date=2023-12-01. | Сервер возвращает 200 и массив мероприятий, соответствующих фильтрам. Проверить корректность выборки. | Совпадает с ожидаемым. |
| 10 | Создание мероприятия | Отправить POST-запрос на api/events/create-event/ с заголовком Authorization: Bearer <access\_token> и телом: { "title": "Митап", "start\_time": "2023-12-15T14:00:00Z" }. | В таблице events создается запись. Сервер возвращает 201 и данные мероприятия. | Совпадает с ожидаемым. |
| 11 | Обновление мероприятия | Отправить PUT-запрос на api/events/1/ с заголовком Authorization: Bearer <access\_token> и телом: { "title": "Обновленный митап" }. | В таблице events обновляется title для мероприятия с id = 1. Сервер возвращает 200 и обновленные данные. | Совпадает с ожидаемым. |
| 12 | Получение списка заявок | Отправить GET-запрос на api/events/1/applications/ с заголовком Authorization: Bearer <access\_token>. | Сервер возвращает 200 и массив заявок на мероприятие с event\_id = 1. | Совпадает с ожидаемым. |

Продолжение таблицы 4.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Функция | Описание теста | Ожидаемый результат | Результат теста |
| 13 | Принятие заявки | Отправить POST-запрос на api/events/1/applications/1/accept/ с заголовком Authorization: Bearer <access\_token>. | В таблице applications обновляется status = "accepted". Сервер возвращает 200 и уведомление пользователю. | Совпадает с ожидаемым. |
| 14 | Отклонение заявки | Отправить POST-запрос на api/events/1/applications/1/reject/ с заголовком Authorization: Bearer <access\_token>. | В таблице applications обновляется status = "rejected". Сервер возвращает 200 и уведомление пользователю. | Совпадает с ожидаемым. |

Функциональное тестирование подтвердило корректную работу базовых функций приложения, включая регистрацию, авторизацию и управление профилем. Данные успешно сохраняются в базе PostgreSQL, а пользовательский интерфейс отображает изменения без ошибок. Функциональность чата, подписки и просмотра потока пользователей работает стабильно, обеспечивая удобство взаимодействия.

**4.2 Тестирование функции добавления мероприятий в Google Календарь**

В процессе тестирования функции handleAddToGoogleCalendar была проверена корректность формирования URL-адреса для интеграции с Google Календарь, включая обработку данных мероприятия, кодирование параметров и преобразование временных меток. Тест-кейсы для функции представлены в таблице 4.2.

Таблица 4.2 – Тест-кейсы для функции

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Сценарий | Входные данные | Ожидаемый результат | Результат теста |
| 1 | Корректные данные | selectedEvent:- title: "Митап",- description: "Встреча разработчиков",- start\_time: "2023-12-15T14:00:00Z",- end\_time: "2023-12-15T16:00:00Z",- location: { lat: 55.751244, lng: 37.618423 } | URL: <https://www.google.com/calendar/...&text=Митап&details=Встреча%20разработчиков&location=55.751244%2C%2037.618423&dates=20231215T140000Z/20231215T160000Z>.  Вызов Linking.openURL с этим URL. | Успешно |

Продолжение таблицы 4.2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Сценарий | Входные данные | Ожидаемый результат | Результат теста |
| 2 | Отсутствие локации | selectedEvent без поля location | URL содержит  location=undefined%2C%20undefined.Функция не выбрасывает ошибку. | Успешно |
| 3 | Спецсимволы в названии и описании | title: "Event & Test!",description: "Презентация #1" | Символы &, # и пробелы кодируются:text=Event%20%26%20Test%21&details=Презентация%20%231 | Успешно |
| 4 | Невалидные даты | start\_time: "некорректная дата",end\_time: "2023-12-15T16:00:00Z" | Функция выбрасывает ошибку (если не обработано) или возвращает Invalid Date в URL. | Ошибка¹ |
| 5 | Пустые обязательные поля | title: "",start\_time: отсутствует | URL содержит пустые параметры (text=&dates=undefined).Google Calendar игнорирует пустые поля. | Успешно |

Проведенное тестирование подтвердило стабильную работу функции добавления со бытия в Google Календарь.

**4.3 Результаты тестирования функций администратора**

В процессе тестирования административного функционала были проверены

ключевые операции, доступные администраторам в интерфейсе Django Admin.

Основное внимание уделялось отображению списка пользователей, удалению

и редактированию данных пользователей, созданию, редактированию и просмотру мероприятий. Все тесты были выполнены успешно, и данные обновлялись корректно.

Результаты тестов представлены в таблице 4.3.

Таблица 4.3 – Результаты тестирования функций администратора

|  |  |  |  |
| --- | --- | --- | --- |
| № | Функция | Описание теста | Результат теста |
| 1 | Просмотр списка пользователей | Проверка отображения полей username, email, city, birth\_date, telegram\_user-name.<br>Фильтрация по городу.<br>Поиск по username, email, telegram\_username. | Успешно |
| 2 | Удаление пользователя | Удаление пользователя «TestUser1» через Django Admin.<br>Проверка удаления записи из таблицы CustomUser и каскадного удаления связанных данных. | Успешно |

Продолжение таблицы 4.3

|  |  |  |  |
| --- | --- | --- | --- |
| № | Функция | Описание теста | Результат теста |
| 3 | Редактирование пользователя | Изменение полей email и city у пользователя «TestUser2».<br>Проверка обновления данных в БД и отображения изменений в интерфейсе. | Успешно |
| 4 | Просмотр списка мероприятий | Проверка отображения полей title, city, start\_time, end\_time, capacity, created\_by.<br>Фильтрация по городу и категориям.<br>Поиск по title, city, description. | Успешно |
| 5 | Создание мероприятия | Создание мероприятия с обязательными полями (title, city, start\_time).<br>Проверка записи в БД и отображения в административной панели. | Успешно |
| 6 | Редактирование мероприятия | Изменение title и capacity у мероприятия «Митап по React».<br>Проверка обновления данных в БД и интерфейсе. | Успешно |
| 7 | Удаление мероприятия | Удаление мероприятия «ТехноФест-2023».<br>Проверка отсутствия записи в БД и удаления связанных заявок/отзывов. | Успешно |

Результаты тестирования подтвердили, что административные функции работают корректно. Интерфейс Django Admin позволяет эффективно управлять данными пользователей, включая просмотр, редактирование и удаление информации, а также управлением мероприятиями. Это обеспечивает стабильность и удобство администрирования системы.

## **4.4 Выводы по разделу**

1. Тестирование проводилось ручным методом с использованием Postman. Проведено тестирование 14 основных функций приложения, 5 сценариев интеграции с Google Календарь и 7 административных операций. Все ключевые операции, включая регистрацию, аутентификацию, управление мероприятиями, заявками, отзывами, а также функции администрирования (создание, редактирование, удаление пользователей и мероприятий), работают корректно.

2. Все тесты выполнены успешно, а пользовательский интерфейс обновляется в ответ на изменения состояния данных.

3. Использование ручного тестирования позволило выявить и устранить мелкие недочеты в работе интерфейса и функционала.

4. Количество тестов составило 26.

# **5 Руководство пользователя**

## **5.1 Руководство для пользователя с ролью «Гость»**

При открытии web-приложения пользователь видит страницу входа с двумя полями: username и password. Страница входа представлена на рисунке 5.1.

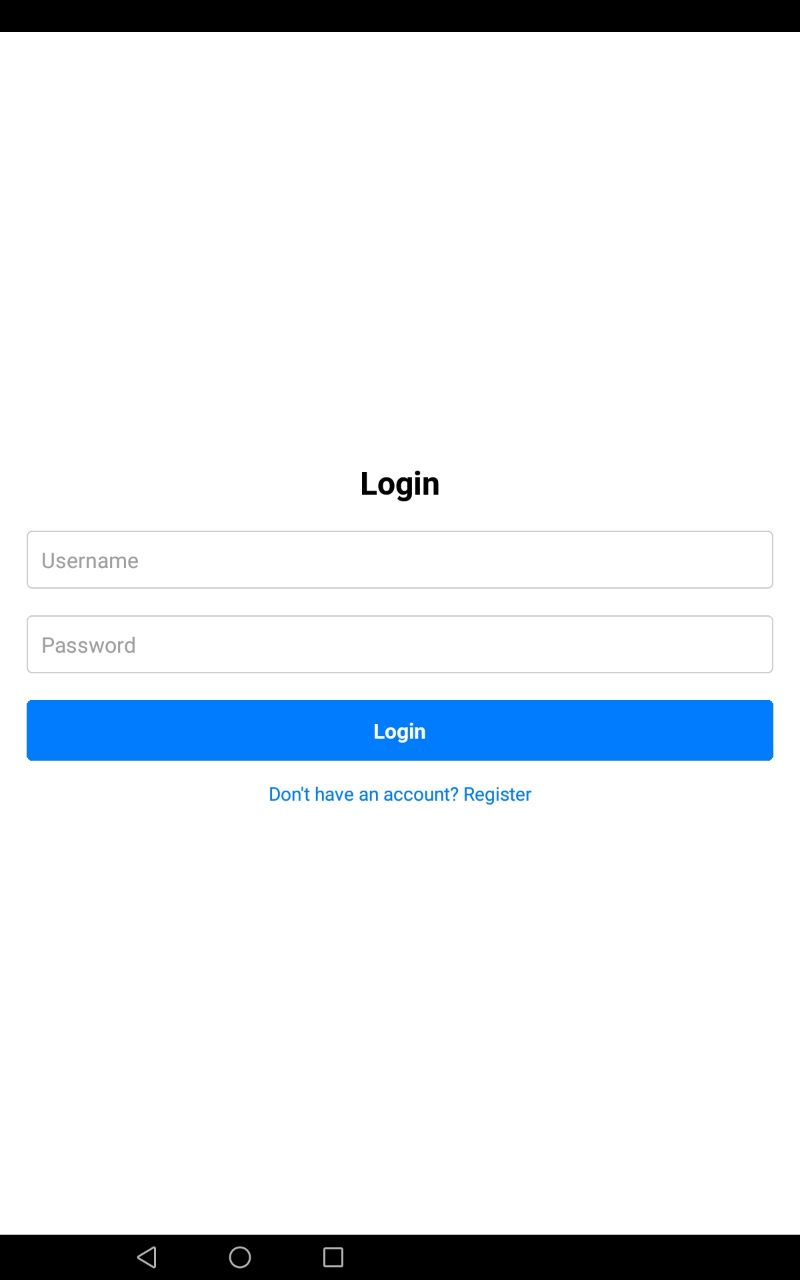


Рисунок 5.1 – Страница входа

Под полями расположены кнопка «Login» для авторизации и «Don’t have an account? Register» для перехода к созданию новой учетной записи.

### **5.1.1 Вход в аккаунт**

Для входа в аккаунт пользователь вводит свои данные в поля «Username» и «Password» на странице входа.

После этого нужно нажать кнопку «Login». Если данные введены корректно, пользователь будет перенаправлен на страницу профиля.

### **5.1.2 Регистрация**

Для регистрации в приложении пользователь нажимает на кнопку «Don’t have an account? Register» на странице входа, после чего попадает на форму регистрации. В этой форме необходимо указать имя, пароль, адрес электронной почты, город и дату рождения. После заполнения всех полей пользователь отправляет данные, нажав кнопку «Register». Успешно зарегистрировавшись, он может войти в аккаунт и начать пользоваться приложением.

Также если пользователь вспомнит, что у него уже есть аккаунт, то на экране есть кнопка «Already have an account? Login».

Страница регистрации представлена на рисунке 5.2.

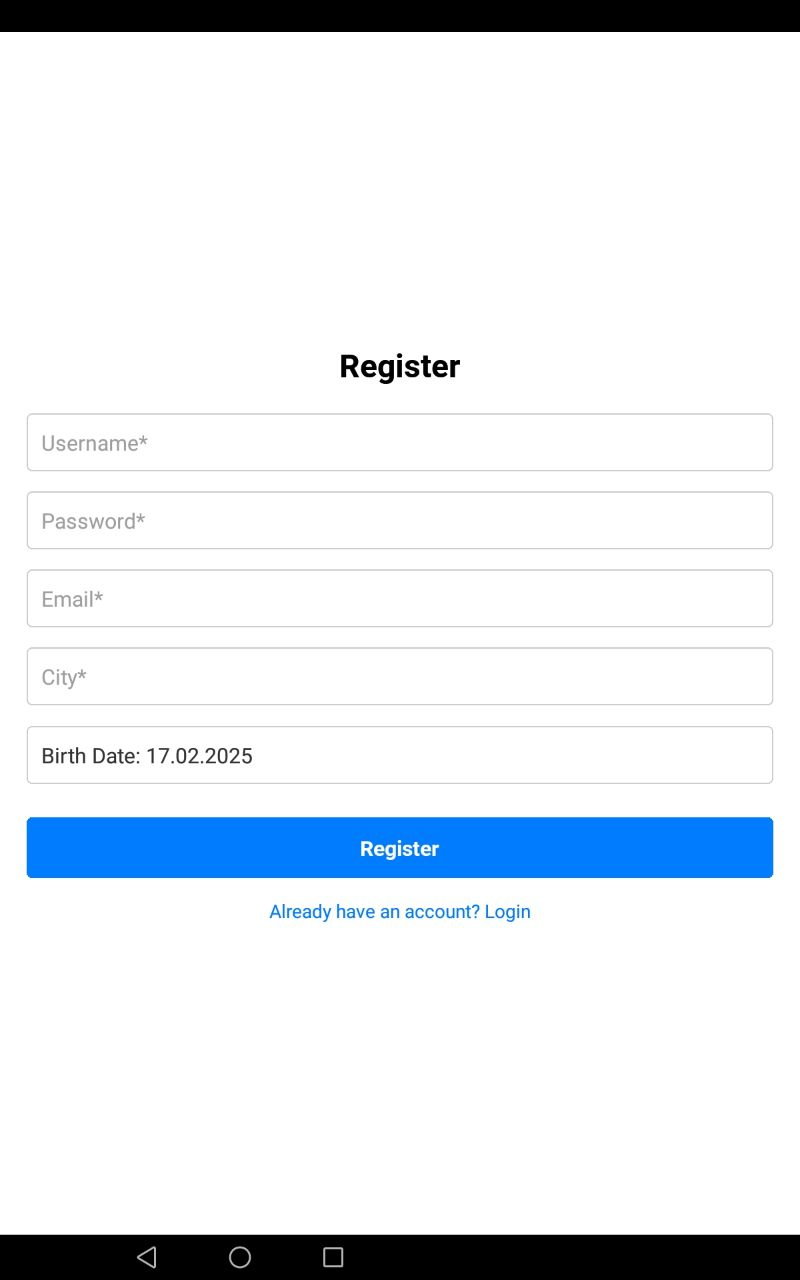


Рисунок 5.2 – Страница регистрации

Далее пользователь переходит на страницу заполнения профиля.

## **5.2 Руководство для пользователя с ролью «Участник»**

### **5.2.1 Создание и редактирование профиля**

В приложении для организации и участия в мероприятиях пользователь при создании профиля указывает имя, электронную почту, город и дату рождения.

После на странице «Profile» пользователь может редактировать эти данные или добавлять новые данные: информацию о себе и имя аккаунта в телеграме. Также можно добавить фото профиля.

Страница собственного профиля представлена на рисунке 5.3.

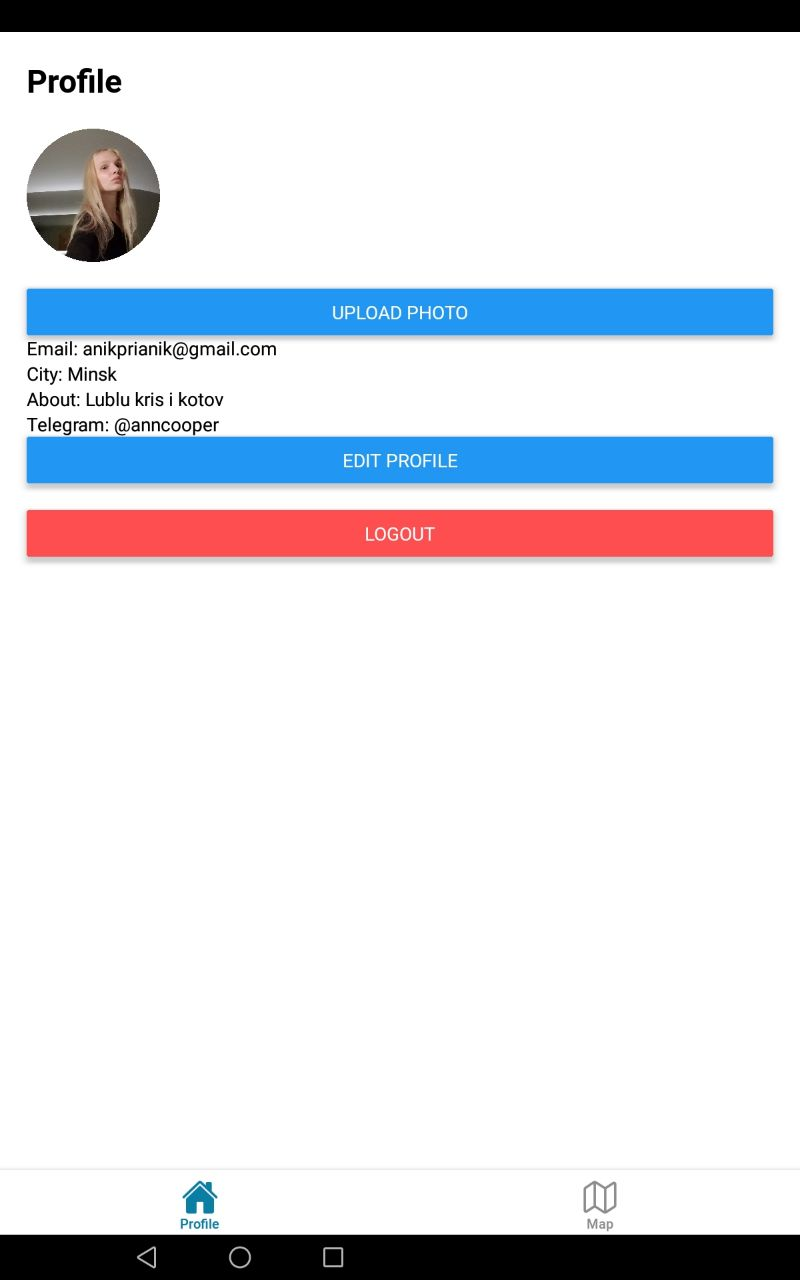


Рисунок 5.3 – Страница профиля

Изменения сохраняются сразу и становится доступным для других пользователей.

### **5.2.2 Просмотр мероприятий**

В приложении на вкладке «Map» находится карта рисунок 5.4.

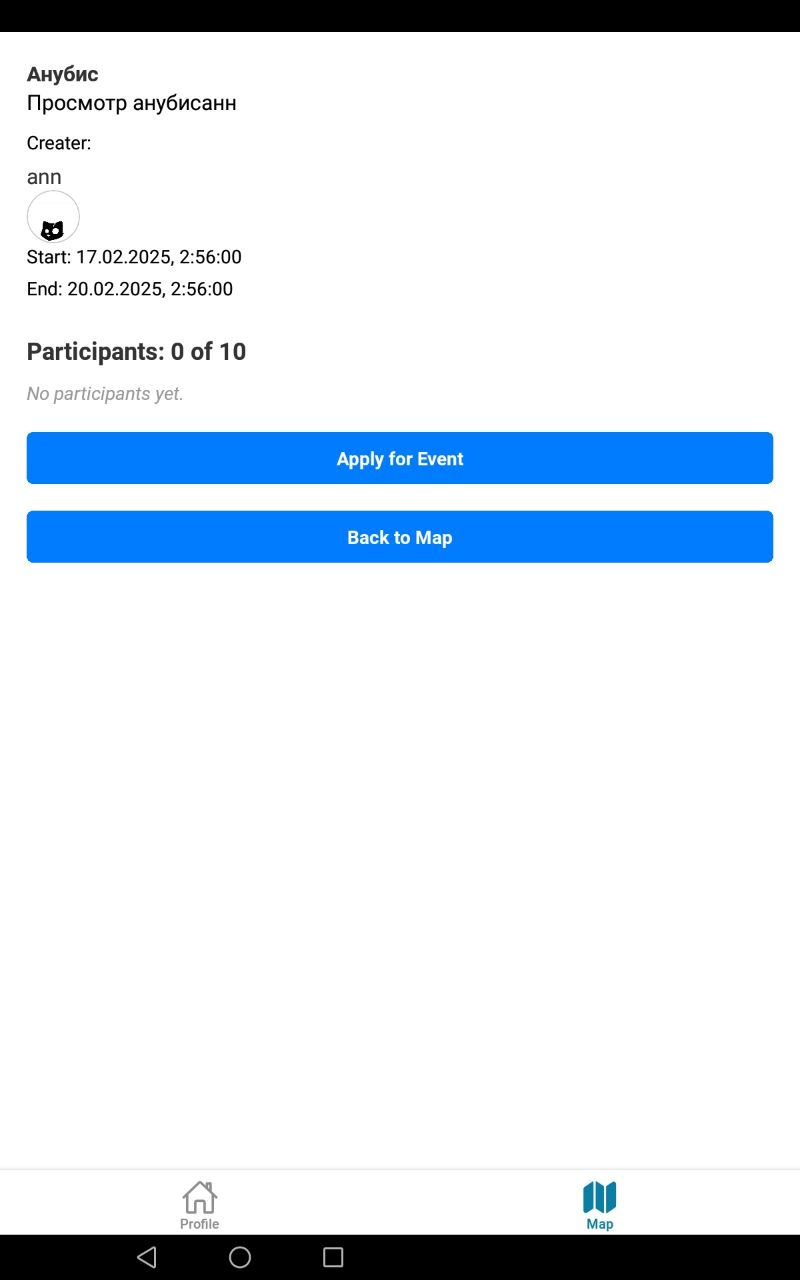


Рисунок 5.4 – Страница потока людей

Красные метки на этой карте являются мероприятиями, которые создали пользователи. Чтобы переключаться между ними можно использовать стрелки в нижней части экрана или кликнуть на саму метку.

### **5.2.3 Подача и отмена заявок на присоединение к мероприятию**

Для того, чтобы подать заявку, надо выбрать заинтересовавшие мероприятие и кликнуть на него, после этого откроется окно с деталями мероприятия. Страница мероприятия представлена на рисунке 5.5.



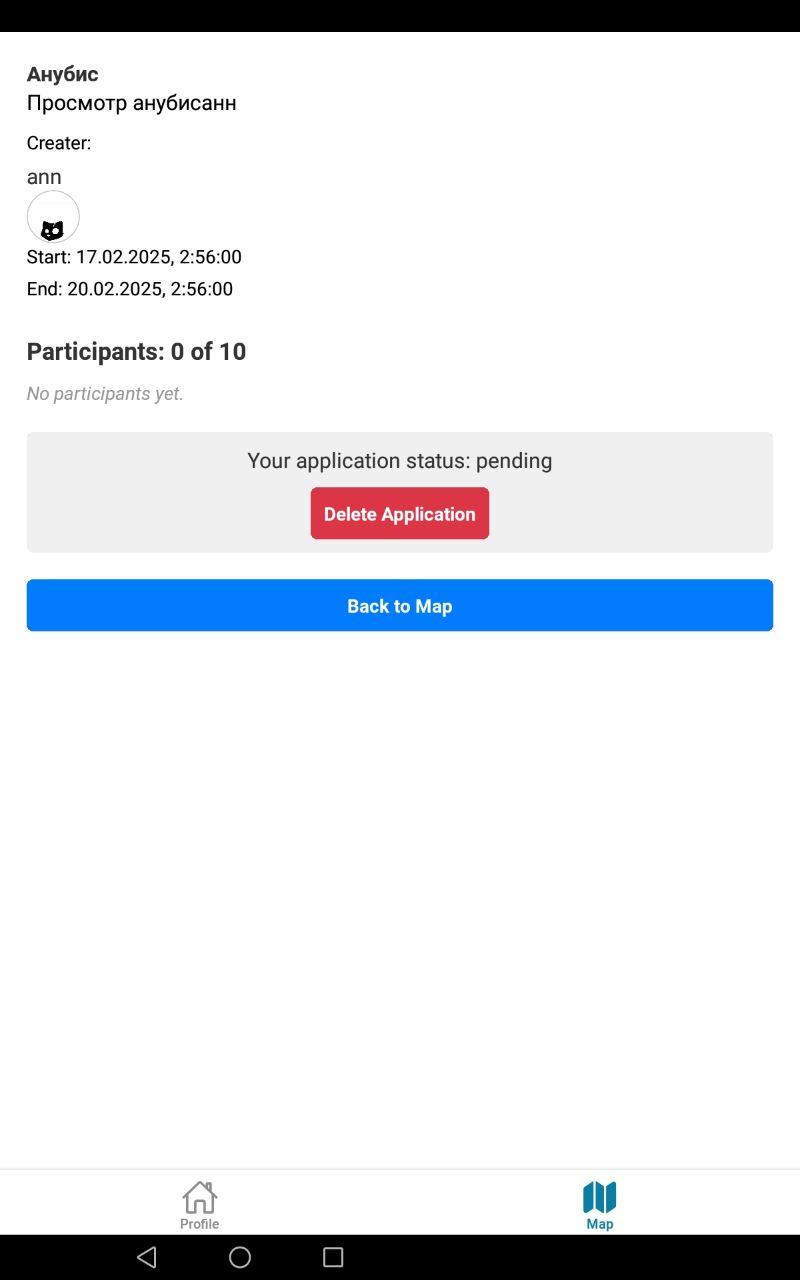
Фильма Анубис

Фильма Анубис

Рисунок 5.5 – Страница мероприятия

Дальше нужно найти кнопку «Apply for event» и нажать на нее.

После этого, если вы передумаете, то вы можете отменить свою заявку, нажав на кнопку «Delete Application». Страница мероприятия с отменой заявки представлена на рисунке 5.5.



Фильма Анубис

Фильма Анубис

Рисунок 5.6 – Страница мероприятия с отменой заявки

После нажатия данной кнопки снова появляется страница мероприятия с возможностью подачи заявки на участие в нём.

### **5.2.4 Добавление мероприятия в Google Календарь**

Чтобы добавить мероприятие в Google Календарь нужно дождаться пока организатор мероприятия рассмотри вашу заявку, и, после этого, в случае успешного удовлетворения вы сможете добавить мероприятие в свой Google Календарь, нажав на кнопку «Add to Google». Страница мероприятия с одобренной заявкой представлена на рисунке 5.7.

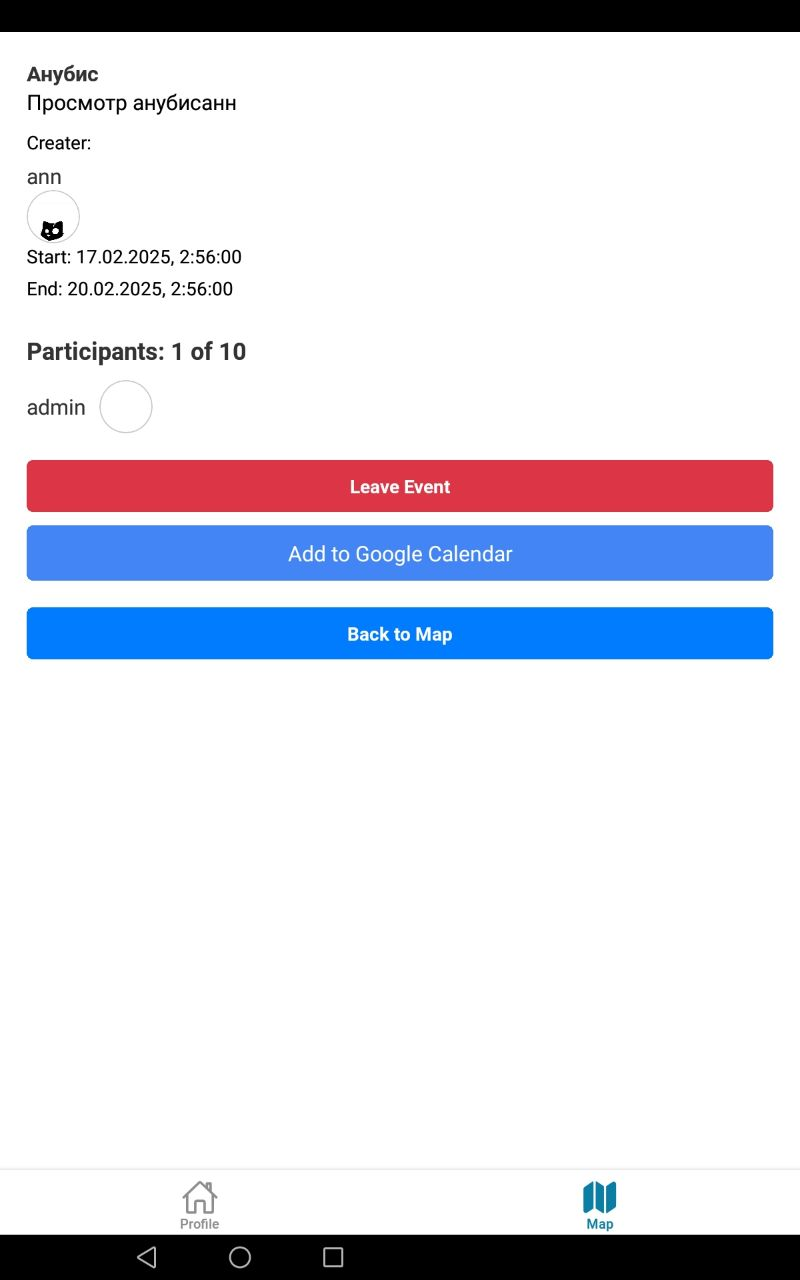


Рисунок 5.7 – Страница мероприятия с одобренной заявкой

После нажатия открывается Google Календарь со всеми заполненными полями.

### **5.2.5 Написание отзывов**

После того, как мероприятие завершится оно автоматически перенесётся в архив, и, если вы являлись его участником, то вы сможете оставить отзыв.

Страница с мероприятием, на которой можно оставить отзыв, представлена на рисунке 5.8.

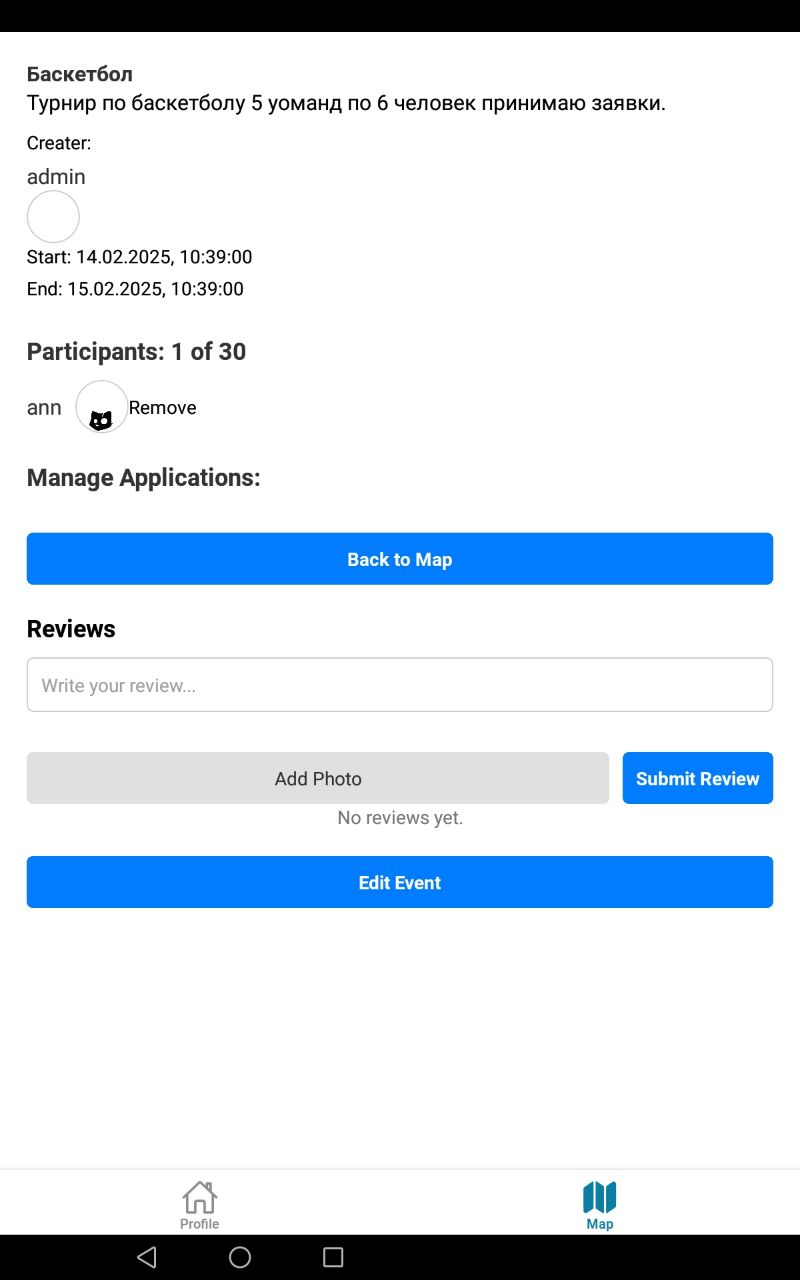


Рисунок 5.8 – Страница с мероприятием, на которой можно оставить отзыв

Далее этот отзыв могу видеть все пользователи, зашедшие на данное мероприятие.

### **5.2.6 Просмотр профилей других пользователей**

Вы можете просмотреть профили других участников мероприятия или профили создателей мероприятий. Для этого надо кликнуть на изображение профиля на странице мероприятия. Страница другого пользователя представлена на рисунке 5.9.

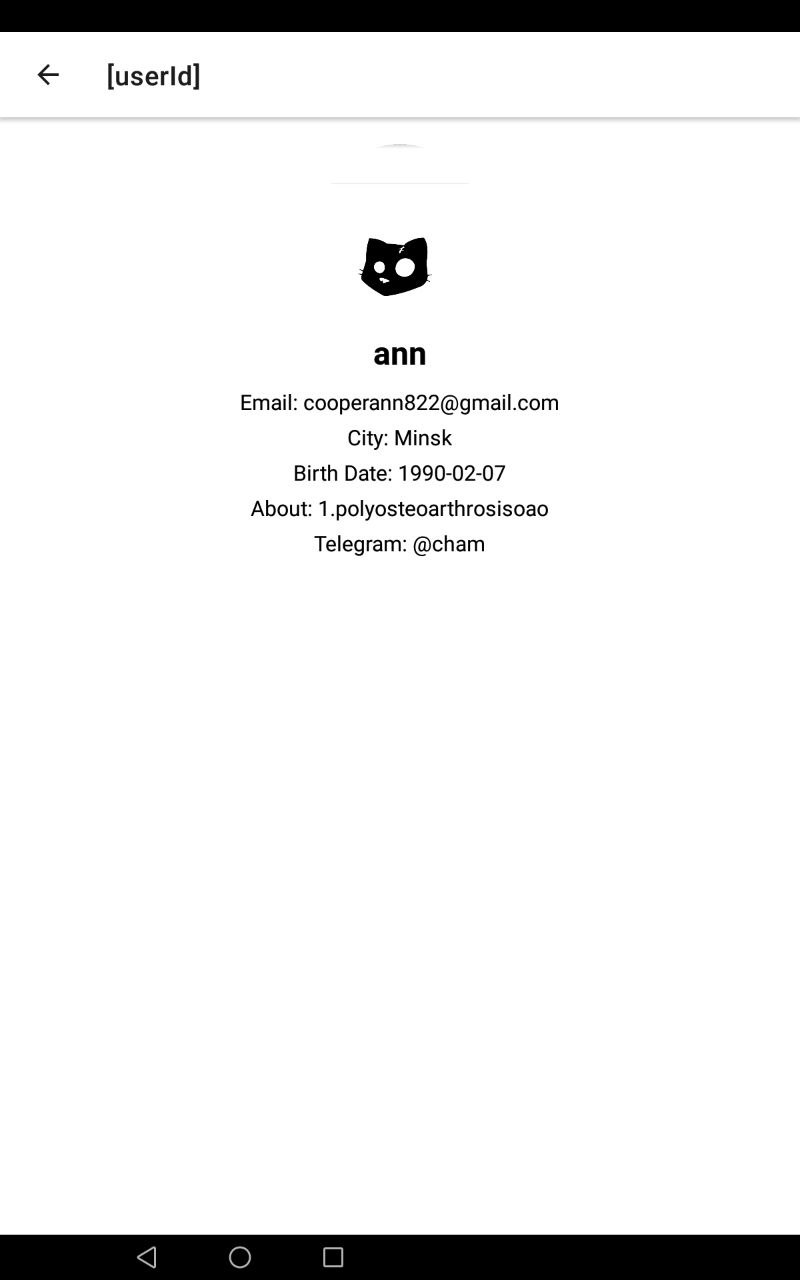


Рисунок 5.9 – Страница другого пользователя

На странице представлена вся информация профиля с фотографией.

### **5.2.7 Поиск и фильтрация мероприятий**

Для того, чтобы воспользоваться поиском, надо на вкладке «Map» нажать на пустое поле в верхней части экрана и написать название мероприятия, которое нужно найти. При нажатии на кнопку «Search» произойдет поиск нужных мероприятий. Чтобы воспользоваться фильтрами, надо нажать на кнопку «Filter», и откроется окно с фильтрами. Страница фильтрации представлена на рисунке 5.10.

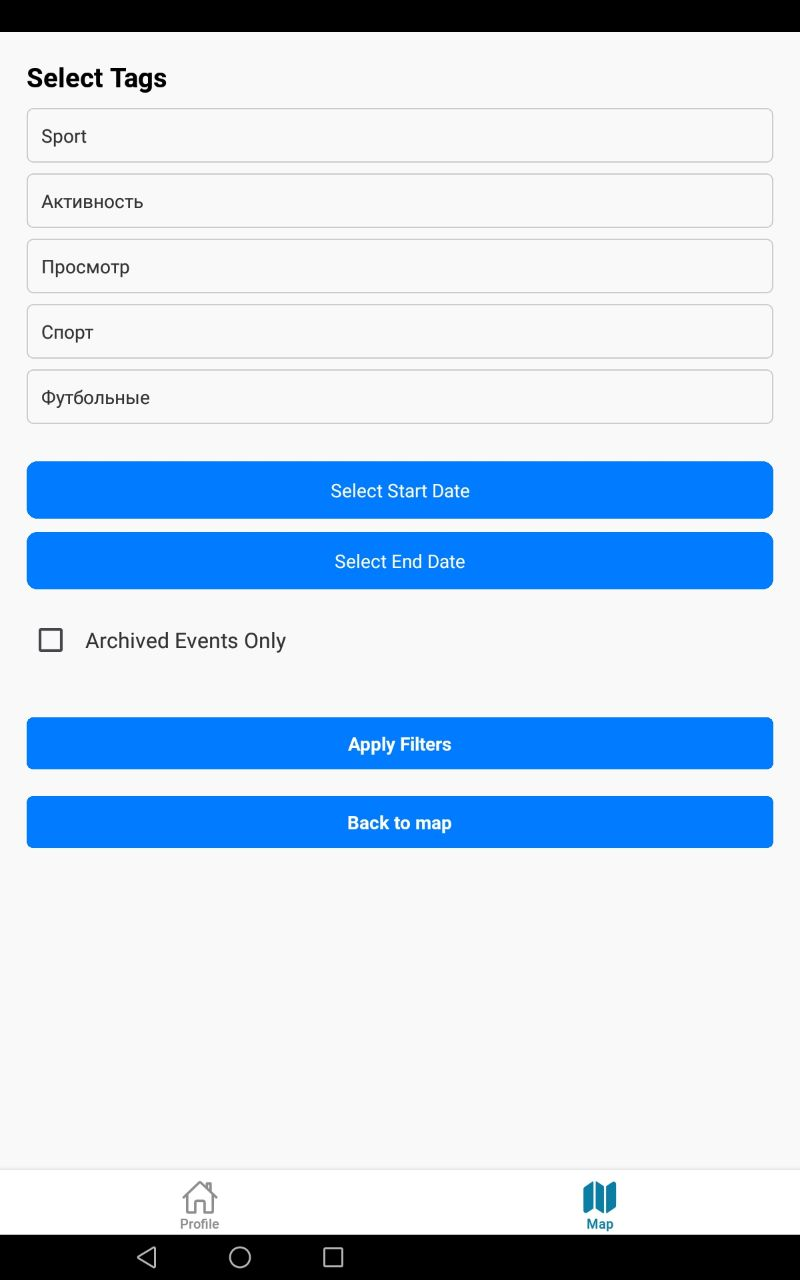


Рисунок 5.10 – Страница фильтрации

После того как вы выберете фильтры вам нужно нажать кнопку «Apply Filters» и фильтры применятся

### **5.2.8 Создание мероприятия**

Чтобы создать мероприятие, нужно нажать на «Create Event» на вкладке «Map». После этого откроется меню с созданием мероприятия. Страница с созданием мероприятия представлена на рисунке 5.11.

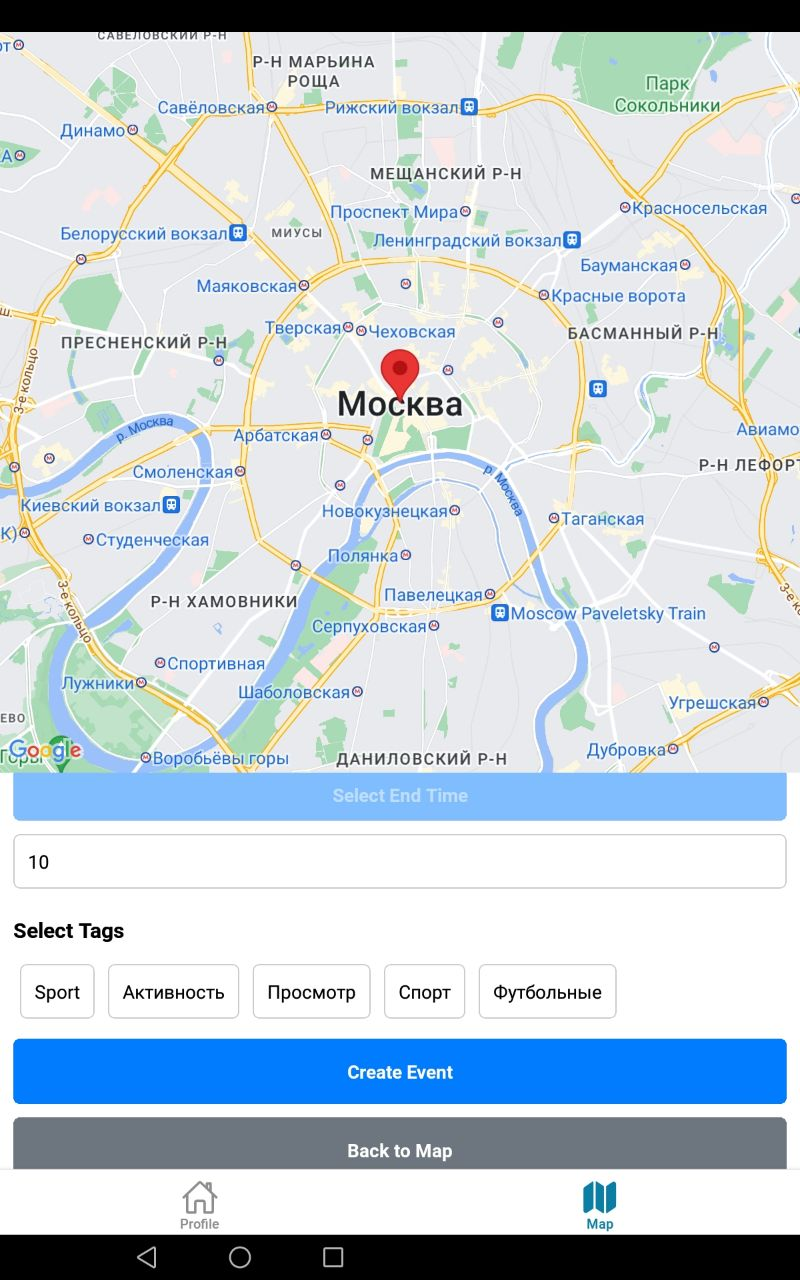


Рисунок 5.11 – Страница фильтрации

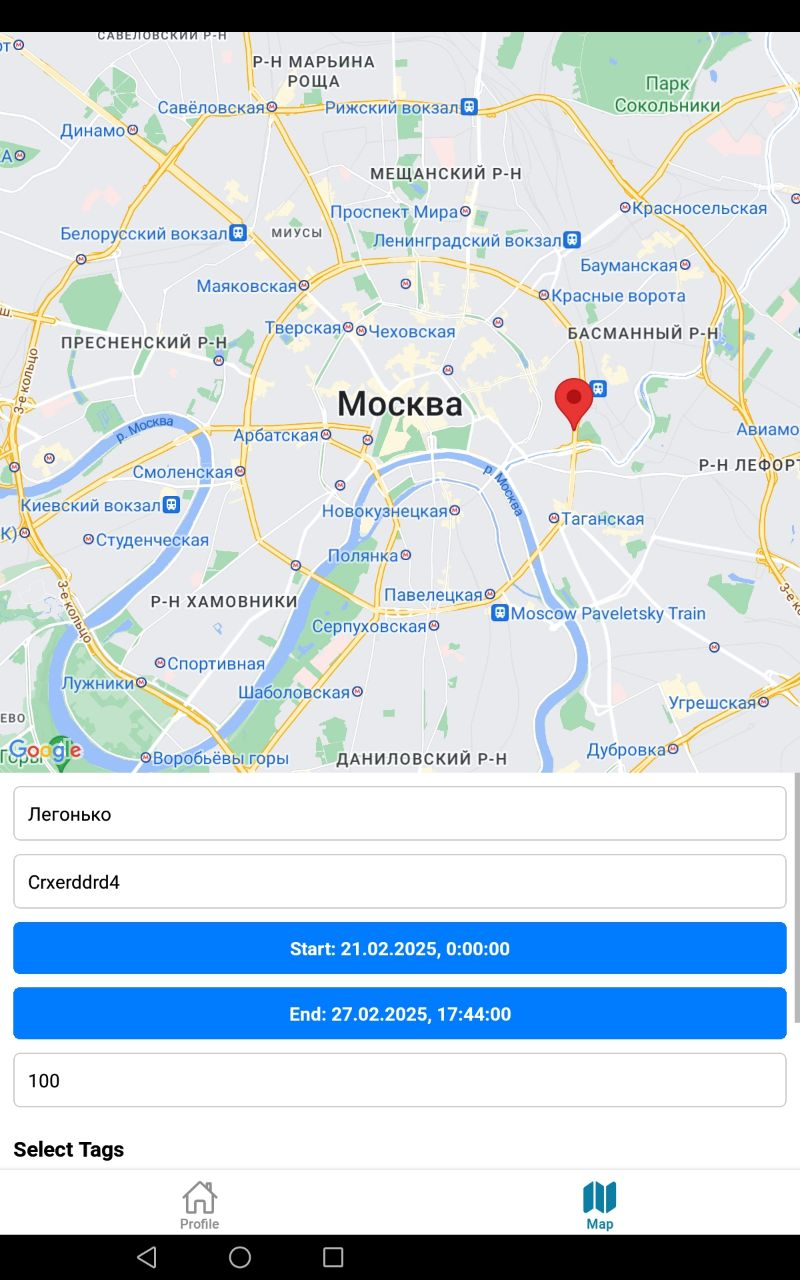
После заполнения нужно нажать на кнопку «Create Event», и данное мероприятие будет создано.

## **5.3 Руководство пользователя с ролью «Создатель»**

### **5.3.1 Редактирование мероприятия**

Чтобы отредактировать информацию о созданном мероприятии надо, найти его на карте и нажать на кнопку «edit Event».

Страница для редактирования мероприятий представлена на рисунке 5.12.



Баскетбол для всех

Рисунок 5.12 – Страница для редактирования мероприятий

Для того, чтобы сохранить изменения, нужно нажать «update event».

## **5.3.2 Просмотр заявок на участие в созданных мероприятиях**

Чтобы просмотреть заявки на созданное мероприятие, надо найти его на карте и нажать на него. После этого можно принимать или отказывать на заявки желающих попасть на созданное мероприятие.

Страница просмотра заявок представлена на рисунке 5.13.

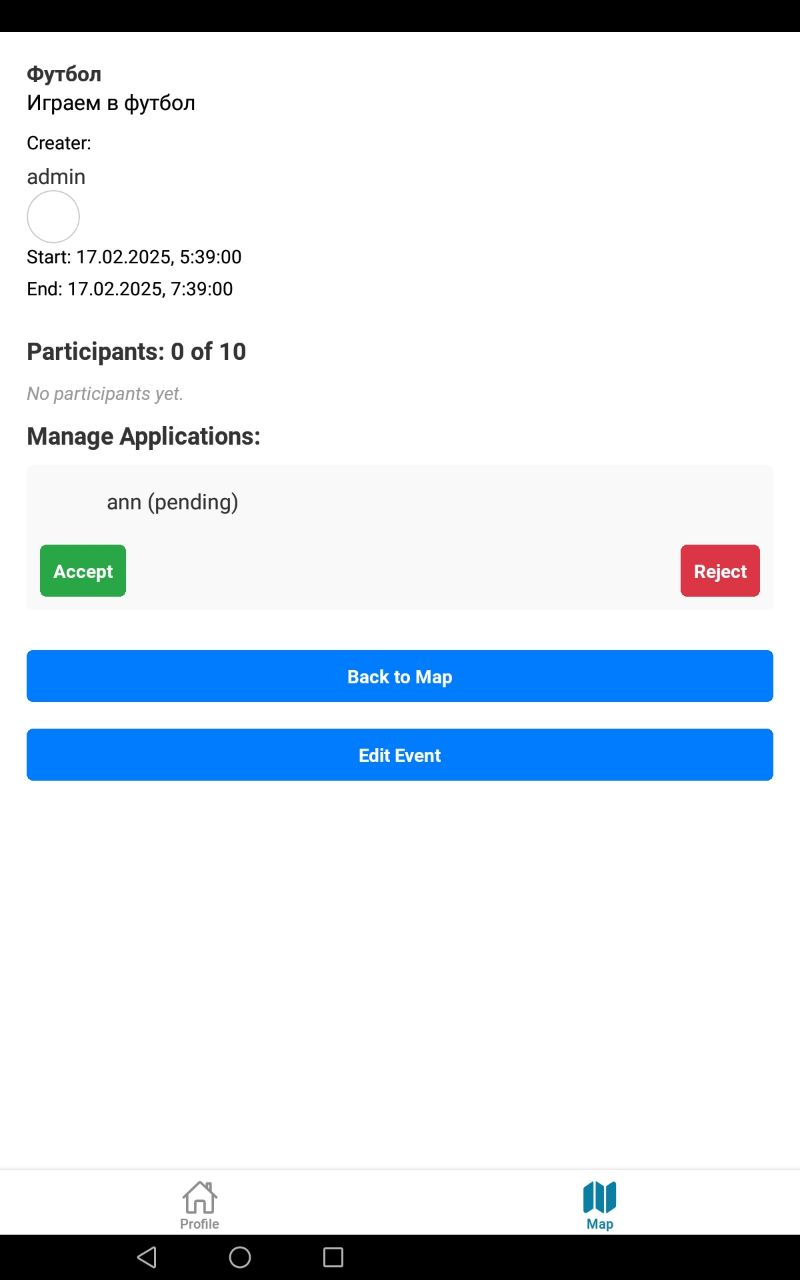


Рисунок 5.13 – Страница просмотра заявок

Чтобы принимать или отказывать на заявки, нужно нажимать на кнопки «Accept» и «Reject».

## **5.3.3 Редактирование списка участников созданных мероприятий**

На странице своего мероприятия, можно удалять участников используя кнопку «remove» напротив участника. Страница редактирование списка участников представлена на рисунке 5.14.

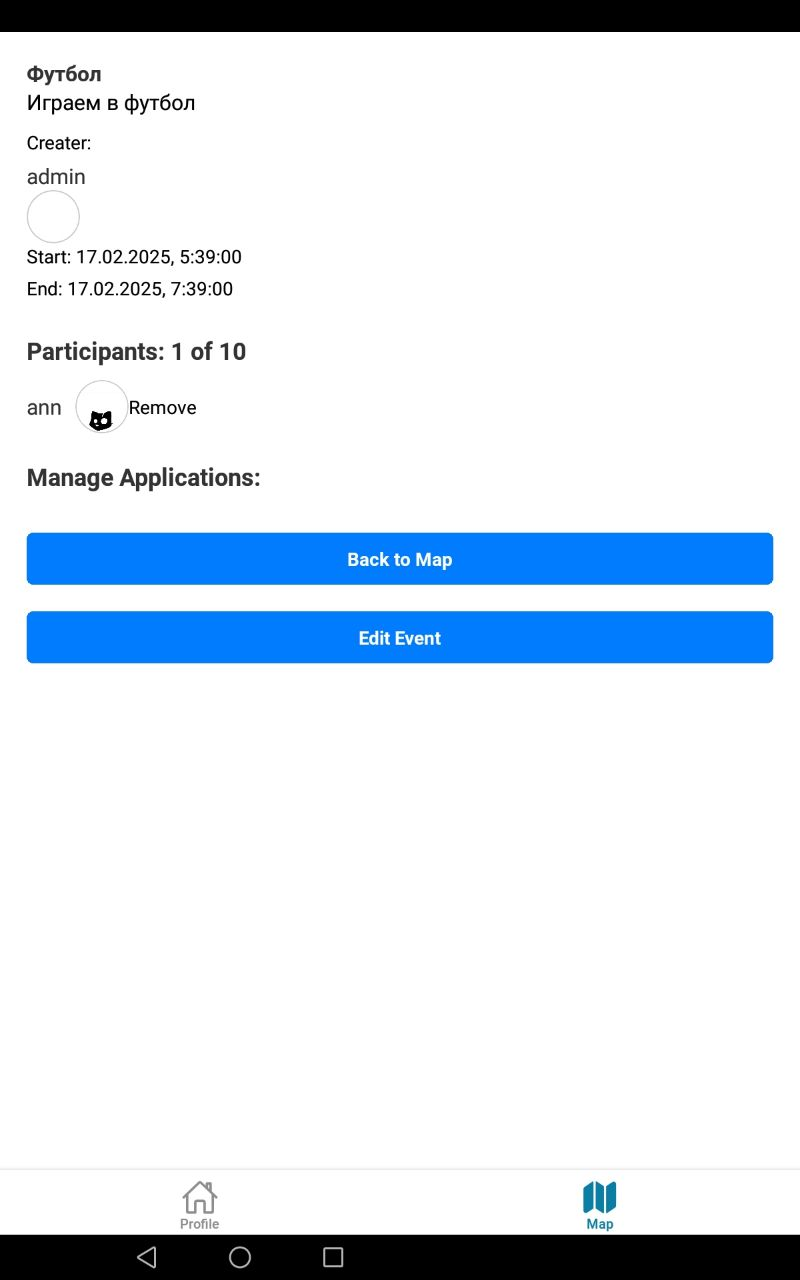


Рисунок 5.14 – Страница редактирование списка участников

## **5.4 Руководство пользователя с ролью «Администратор»**

### **5.4.1 Редактирование списка пользователей**

Администратор может просматривать и редактировать список всех пользователей. Для этого необходимо перейти в панель администратора. Страница панели администратора представлена на рисунке 5.15.



Рисунок 5.15 – Страница панели администратора

В ней будет раздел «СustomUsers». В котором можно перейти на страницу с пользователями. Страница администратора для просмотра пользователей представлена на рисунке 5.16.

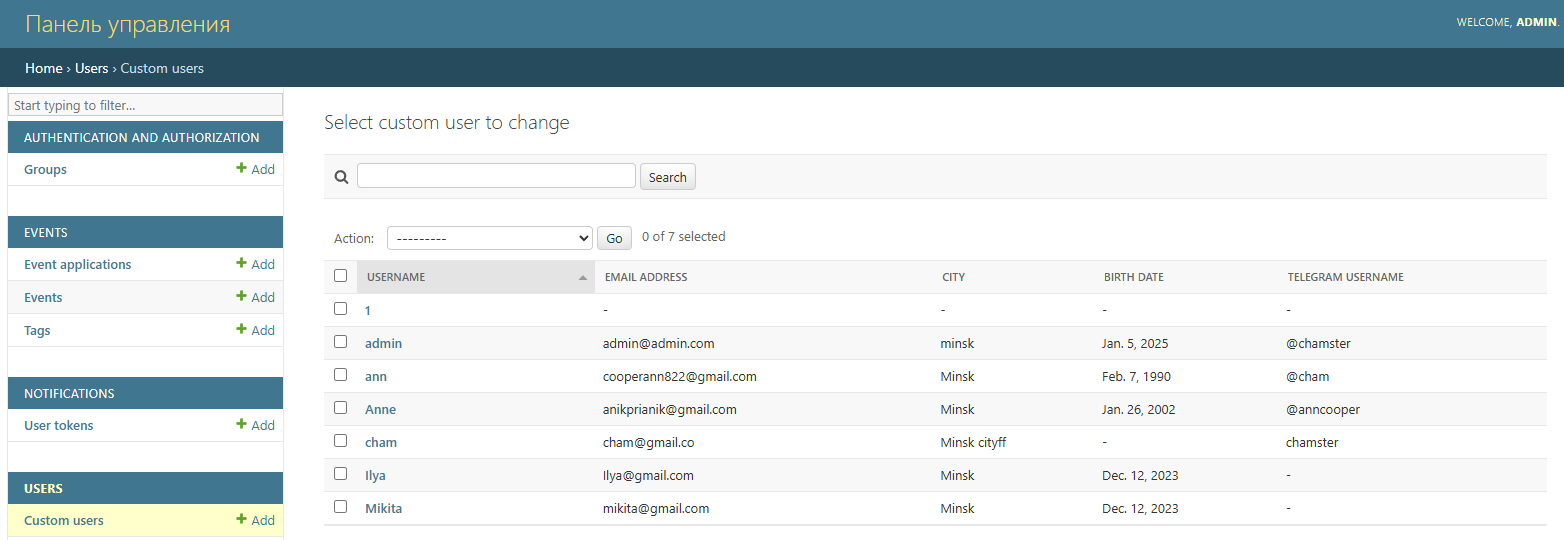


Рисунок 5.16 – Страница администратора для просмотра пользователей

## **5.4.2 Редактирование списка всех мероприятий**

Администратор может просматривать и редактировать список всех мероприятий. Для этого необходимо перейти в панель администратора. В ней будет раздел «Events». В котором можно перейти на страницу с мероприятиями. Страница администратора для просмотра мероприятий представлена на рисунке 5.17.

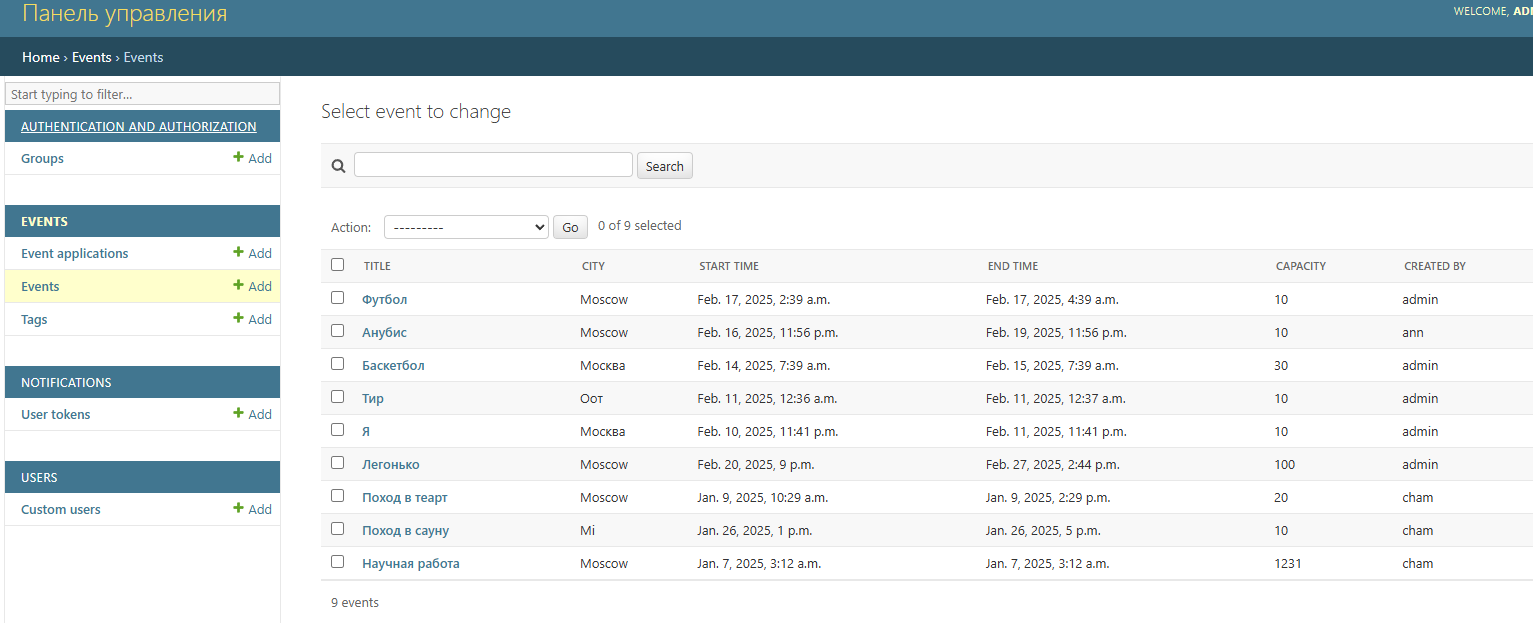


Рисунок 5.17 – Страница администратора для просмотра мероприятий

## **5.5 Выводы по разделу**

1. Разработано руководство, описывающее действия пользователей системы: гостя, участника, создателя и администратора, с учетом уникальных функций каждой роли, описанных в диаграмме вариантов использования.

2. Гости могут зарегистрироваться и аутентифицироваться. Участники могут редактировать пользовательские данные, просматривать мероприятия, подавать и отменять заявку на присоединение к мероприятию; добавлять мероприятия в Google Календарь; оставлять отзывы о прошедших мероприятиях; просматривать профили других пользователей; просматривать информацию о мероприятии; просматривать профили других пользователей; искать мероприятия по ключевым словам и фильтровать по категориям; создавать мероприятия. Создатели могут то же, что и участники и ещё редактировать информацию о созданных мероприятиях, просматривать заявки на участие в созданных мероприятиях и редактировать список участников созданных мероприятий. Администратор может редактировать список всех пользователей и редактировать список всех мероприятий.

**Заключение**

В результате работы над проектом было разработано web-приложение «EventFlow» для организации и участия в мероприятиях, которое соответствует заявленным целям и требованиям:

1. Web-приложение поддерживает четыре ключевые роли: гость, участник, создатель и администратор.

2. Web-приложение использует клиент-серверную архитектуру. Серверная часть реализована на Django Rest Framework, клиентская – Expo React Native.

3. Web-приложение включает 17 ключевых функций, охватывающих весь необходимый функционал: создание профиля, создание мероприятий, подачи заявок на участие в мероприятиях, оставление отзывов.

4. Для хранения данных была создана база данных, содержащая 7 таблиц: tag, event, review, review\_image, event\_applications, custom\_user, user\_token.

5. Общий объем программного кода web-приложения составил 6142 строки авторского кода.

6. Для обеспечения качества продукта было проведено тщательное тестирование. Было проведено 26 тестов для проверки функциональности приложения. В итоге данный проект является примером успешного использования современных технологий и инструментов для разработки мобильных приложений. Реализованные функции, архитектура и подход к разработке обеспечивают приложение высокой производительностью, масштабируемостью и удобством для пользователей.

Список используемых источников

1. RFC 2616 HTTP/1.1 [Электронный ресурс]. – Режим доступа: https://www.rfc-editor.org/rfc/rfc2616 – Дата доступа: 01.01.2025.
2. Django Rest Framework [Электронный ресурс]. – Режим доступа: https://www.django-rest-framework.org/ – Дата доступа: 03.01.2025.
3. Eventbrite [Электронный ресурс]. – Режим доступа: https://www.eventbrite.ie/ – Дата доступа: 03.01.2025.
4. Meetup [Электронный ресурс]. – Режим доступа: https://www.meetup.com/ru-RU/ – Дата доступа: 05.01.2025.
5. Eventful [Электронный ресурс]. – Режим доступа: https://eventful.ie/ – Дата доступа: 05.01.2025.
6. PostgreSQL Электронный ресурс]. – Режим доступа: https://www.postgresql.org/ – Дата доступа: 05.01.2025.
7. React Native [Электронный ресурс]. – Режим доступа: https://reactnative.dev/ – Дата доступа: 16.01.2025.
8. Expo Bundel [Электронный ресурс]. – Режим доступа: https://docs.expo.dev/guides/customizing-metro/ – Дата доступа: 16.01.2025.
9. Elasticsearch [Электронный ресурс]. – Режим доступа: https://www.elastic.co/elasticsearch – Дата доступа: 18.01.2025.
10. Localstack [Электронный ресурс]. – Режим доступа: https://www.localstack.cloud/ – Дата доступа: 18.01.2025.
11. AWS [Электронный ресурс]. – Режим доступа: https://aws.amazon.com/ru/ – Дата доступа: 23.01.2025.
12. HTTPS [Электронный ресурс]. – Режим доступа: ttps://www.rfc-editor.org/rfc/rfc2818 – Дата доступа: 21.01.2025.
13. Expo Go [Электронный ресурс]. – Режим доступа: https://expo.dev/go – Дата доступа: 21.01.2025.
14. TCP [Электронный ресурс]. – Режим доступа: https://www.ietf.org/rfc/rfc9293.html – Дата доступа: 22.01.2025.
15. Restfull api [Электронный ресурс]. – Режим доступа: https://aws.amazon.com/what-is/restful-api/ – Дата доступа: 26.01.2025.
16. Django ORM [Электронный ресурс]. – Режим доступа: https://www.pythontutorial.net/django-tutorial/django-orm/ – Дата доступа: 26.01.2025.
17. JSON [Электронный ресурс]. – Режим доступа: https://www.w3schools.com/whatis/whatis\_json.asp – Дата доступа: 31.01.2025.
18. JWT [Электронный ресурс]. – Режим доступа: https://jwt.io/– Дата доступа: 01.02.2025.

# **Приложение А. Логическая схема базы данных**

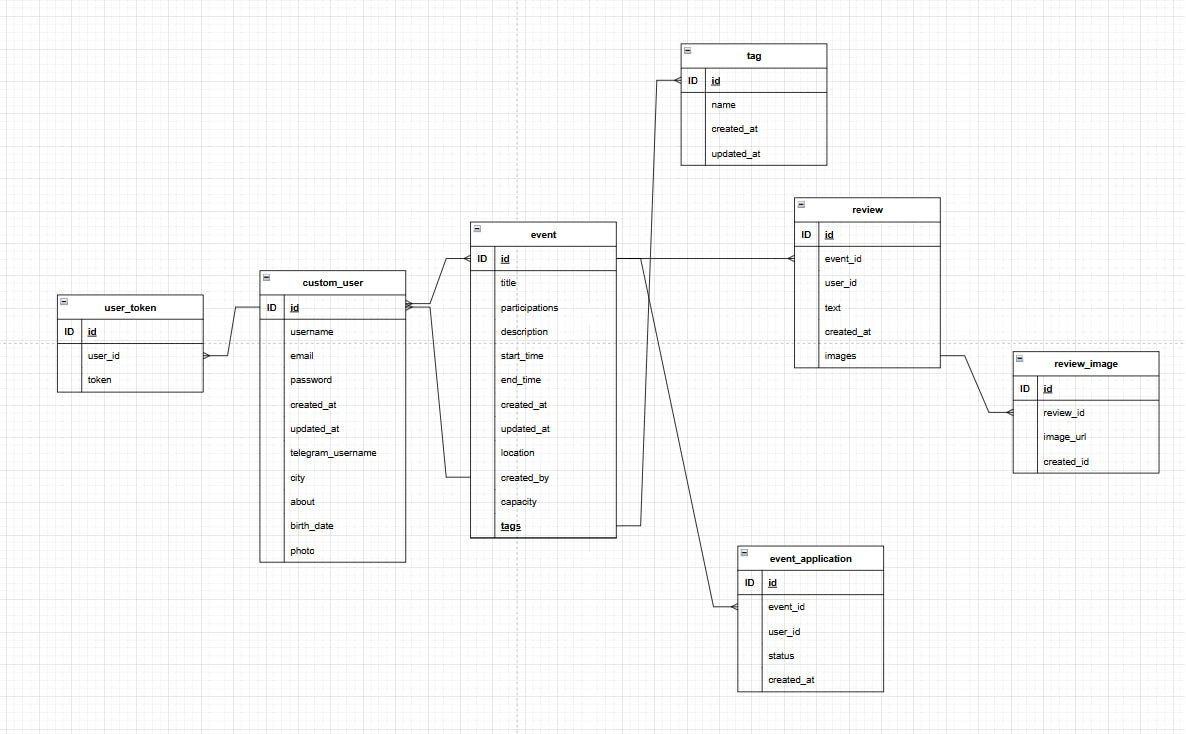


Рисунок – Логическая схема базы данных

# **Приложение Б. Архитектура web-приложения**

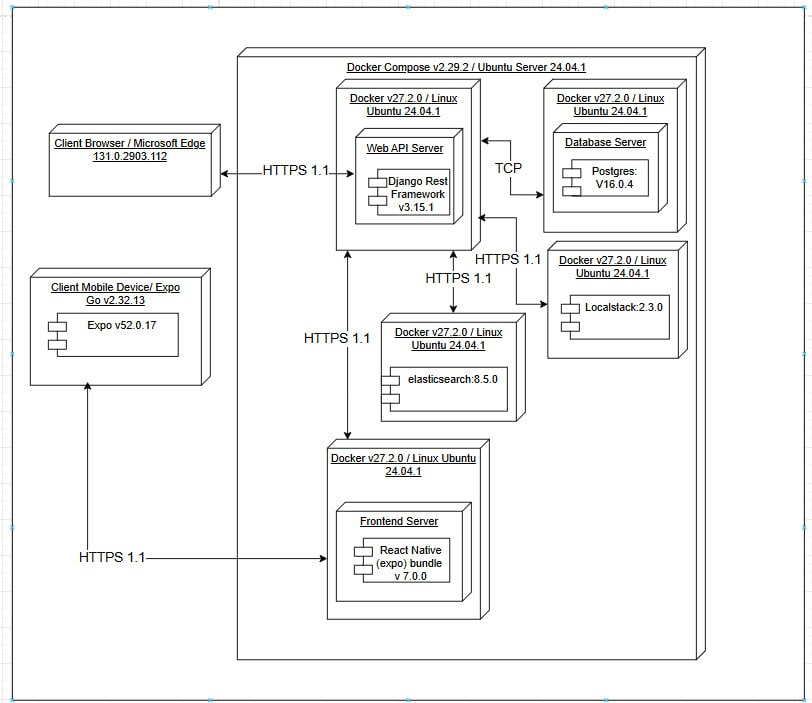


Рисунок – Архитектура web-приложения

# **Приложение В. Создание базы данных**

-- Создание таблицы пользователей

CREATE TABLE users\_customuser (

id SERIAL PRIMARY KEY,

password VARCHAR(128) NOT NULL,

last\_login TIMESTAMPTZ,

is\_superuser BOOLEAN NOT NULL,

username VARCHAR(150) UNIQUE NOT NULL,

first\_name VARCHAR(150),

last\_name VARCHAR(150),

email VARCHAR(254),

is\_staff BOOLEAN NOT NULL,

is\_active BOOLEAN NOT NULL,

date\_joined TIMESTAMPTZ NOT NULL,

photo VARCHAR(500),

city VARCHAR(255),

birth\_date DATE,

about TEXT,

telegram\_username VARCHAR(255)

);

-- Таблица категорий

CREATE TABLE events\_tag (

id SERIAL PRIMARY KEY,

name VARCHAR(255) UNIQUE NOT NULL,

created\_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT\_TIMESTAMP

);

-- Таблица мероприятий

CREATE TABLE events\_event (

id SERIAL PRIMARY KEY,

title VARCHAR(255) NOT NULL,

description TEXT,

city VARCHAR(254) NOT NULL,

location JSONB NOT NULL,

start\_time TIMESTAMPTZ NOT NULL,

end\_time TIMESTAMPTZ NOT NULL,

created\_by\_id INT REFERENCES users\_customuser(id) ON DELETE CASCADE,

capacity INT NOT NULL DEFAULT 0,

created\_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT\_TIMESTAMP

);

-- Связь ManyToMany между мероприятиями и категориями

CREATE TABLE events\_event\_tags (

id SERIAL PRIMARY KEY,

event\_id INT REFERENCES events\_event(id) ON DELETE CASCADE,

tag\_id INT REFERENCES events\_tag(id) ON DELETE CASCADE

);

-- Связь ManyToMany между мероприятиями и участниками

CREATE TABLE events\_event\_participants (

id SERIAL PRIMARY KEY,

event\_id INT REFERENCES events\_event(id) ON DELETE CASCADE,

customuser\_id INT REFERENCES users\_customuser(id) ON DELETE CASCADE

);

-- Таблица отзывов

CREATE TABLE events\_review (

id SERIAL PRIMARY KEY,

text TEXT NOT NULL,

event\_id INT REFERENCES events\_event(id) ON DELETE CASCADE,

user\_id INT REFERENCES users\_customuser(id) ON DELETE CASCADE,

created\_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT\_TIMESTAMP

);

-- Таблица изображений отзывов

CREATE TABLE events\_reviewimage (

id SERIAL PRIMARY KEY,

image\_url VARCHAR(500) NOT NULL,

review\_id INT REFERENCES events\_review(id) ON DELETE CASCADE,

created\_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT\_TIMESTAMP

);

-- Таблица заявок на мероприятия

CREATE TABLE events\_eventapplication (

id SERIAL PRIMARY KEY,

status VARCHAR(10) NOT NULL DEFAULT 'pending',

created\_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT\_TIMESTAMP,

event\_id INT REFERENCES events\_event(id) ON DELETE CASCADE,

user\_id INT REFERENCES users\_customuser(id) ON DELETE CASCADE,

UNIQUE(event\_id, user\_id)

);

-- Таблица токенов для уведомлений

CREATE TABLE notifications\_usertoken (

id SERIAL PRIMARY KEY,

user\_id INT UNIQUE NOT NULL,

token VARCHAR(255) NOT NULL

);

Листинг – Скрипт создания базы данных