

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Автоматизированная система для организации мероприятий и
встреч в сообществе**

Студент _____ Резепов Александр Александрович
(подпись, дата) (ФИО полностью)

Руководитель _____ Афонин А.Ю.
(подпись, дата) (фамилия, инициалы)

Нормоконтролер _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой _____ Козлов А.Ю.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК _____ Попова Н.А.
(подпись) (фамилия, инициалы)

Пенза, 2025

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов
«20» декабря 2024 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Резепов Александр Александрович гр. 21ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>
2. Тема работы Автоматизированная система для организации мероприятий и встреч в сообществе
- Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____
3. Руководитель работы к.т.н., доцент кафедры «МО и ПЭВМ» Афонин А.Ю.
4. Задание на работу (назначение разработки, исходные данные и т.п.):
Назначение: разработка автоматизированной системы для организации мероприятий и встреч в сообществе
- Основные функции:
- 1) Создание мероприятий;
 - 2) Поиск мероприятий;
 - 3) Система подписок и подписчиков;
 - 4) Управление мероприятиями;
 - 5) Управление профилем;
 - 6) Комментирование мероприятий.
- Технология разработки: FastAPI.
- Язык программирования: Python.
- Среда программирования: PyCharm CE 2024.

5. Перечень подлежащих разработке вопросов:

- 1) Анализ предметной области и постановка задачи;
 - 2) Проектирование программных средств;
 - 3) Реализация программных средств;
 - 4) Тестирование программных средств;
 - 5) Оформление ВКР.
- _____
- _____
- _____
- _____
- _____
- _____
- _____
- _____
- _____
- _____
- _____
- _____
- _____
- _____
- _____

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2025 – 14.04.2025	
2	Проектирование программных средств	20%	15.04.2025 – 25.04.2025	
3	Реализация программных средств	30%	26.04.2025 – 12.05.2025	
4	Тестирование программных средств	20%	13.05.2025 – 22.05.2025	
5	Оформление пояснительной записки	20%	23.05.2025 – 03.06.2025	

Дата выдачи задания «20» декабря 2024г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 111 листов, 41 рисунок, 6 таблиц, 25 использованных источников и 2 приложения.

ПРОГРАММИРОВАНИЕ, АВТОМАТИЗИРОВАННЫЕ СИСТЕМЫ, ПРОЕКТИРОВАНИЕ, ВЕБ-СИСТЕМА, ОРГАНИЗАЦИЯ МЕРОПРИЯТИЙ, UML, PYTHON, FASTAPI, NEXT.JS, TELEGRAM-БОТ, DOCKER, SWAGGER.

Предмет работы: разработка автоматизированной системы для организации мероприятий и встреч в сообществе.

Объект работы: автоматизированная система для организации мероприятий и встреч в сообществе.

Цель работы: разработать автоматизированную систему для организации мероприятий и встреч в сообществе.

Технология разработки: FastAPI.

Результаты работы: спроектирована, разработана и протестирована автоматизированная система для организации мероприятий и встреч в сообществе.

Область применения: система может использоваться для организации социальных и культурных мероприятий, корпоративных и бизнес встреч.

Разработка автоматизированной системы проводилась на языке программирования Python в среде разработки PyCharm Community Edition 2024.2.4. Разработка производилась в операционной системе Windows 10.

					ПГУ 09.03.04 – 08БР211.19 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Автоматизированная система для организации мероприятий и встреч в сообществе	Лит.	Лист	Листов
Разраб.		Резепов А.А.						
Провер.		Афонин А.Ю.					4	111
Реценз.						группа 21ВП1		
Н. Контр.		Такташкин Д.В.						
Утверд.		Козлов А.Ю.						

Содержание

Введение	6
1 Аналитическая часть	8
1.1 Анализ предметной области	8
1.2 Анализ продуктов–аналогов	9
1.3 Постановка цели и задач	12
1.4 Анализ требований к программным средствам	12
1.4.1 Анализ функциональных требований	13
1.4.2 Анализ нефункциональных требований	17
2 Проектирование автоматизированной системы для организации мероприятий и встреч в сообществе	18
2.1 Выбор технологий разработки	18
2.2 Проектирование архитектуры приложения	20
2.3 Проектирование пользовательского интерфейса	23
2.4 Оценка бюджета разработки системы	31
3 Реализация автоматизированной системы для организации мероприятий и встреч в сообществе	36
3.1 Реализация серверной части системы	36
3.2 Реализация пользовательского интерфейса системы	46
3.3 Реализация взаимодействия между компонентами системы	51
4 Тестирование и оценка качества кода	57
4.1 Тестирование функционала регистрации методом комбинаторики и попарного тестирования	57
4.2 Анализ сложности кода модуля регистрации	60
4.3 Тестирование совместимости системы	63
Заключение	66
Список использованных источников	67
Приложение А (обязательное) Исходный код программного обеспечения .	70
Приложение Б (рекомендуемое) Тестирование функционала регистрации	101

Введение

В современном обществе обмен информацией и коммуникация играют ключевую роль в организации жизнедеятельности. Благодаря этим процессам люди делятся мыслями, договариваются о совместных действиях, строят профессиональные и личные связи.

С развитием цифровых технологий люди стали больше общаться онлайн, следовательно, возросла потребность в эффективной коммуникации. Обмен информацией необходим не только в профессиональной сфере, но и для поддержания личных связей на расстоянии. Особенно это проявляется при организации различных встреч и мероприятий. Планирование требует значительных усилий: необходимо заниматься согласованием времени, информированием участников и распределением задач.

Автоматизированные инструменты сильно упрощают эти задачи: современные программы предлагают простые и понятные интерфейсы, с помощью которых можно быстро создавать события, менять детали встречи и уведомлять всех участников. Использование таких систем помогает избежать ошибок, уменьшает нагрузку на организаторов мероприятий, и улучшает взаимодействие между участниками.

Актуальность разработки такой системы обусловлена растущей потребностью в эффективных инструментах для организации мероприятий различных типов: от деловых встреч и конференций до культурных, спортивных и образовательных событий. В современном мире, где время – ценный ресурс, автоматизация становится ключевым фактором успеха. Она позволяет существенно сократить временные затраты на планирование, подготовку и проведение мероприятий, а также повысить заинтересованность участников.

Автоматизированные системы предлагают широкий спектр возможностей для оптимизации процессов, таких как регистрация участников, управление списками, распределение ресурсов и координация задач. Они помогают создать удобные условия для обмена информацией, обеспечивая быстрый доступ к

необходимым данным и улучшая коммуникацию между организаторами, участниками и партнерами. В результате автоматизация способствует более эффективному управлению мероприятиями, снижению ошибок и повышению общей удовлетворенности всех сторон.

Однако автоматизация не способна полностью заменить человеческое взаимодействие. Технологические решения выступают как инструменты, облегчающие выполнение повседневных задач и ускоряющие организационные процессы, но они не заменяют живое общение.

Целью данной работы является разработка автоматизированной системы, которая позволит упростить процесс организации мероприятий и будет предоставлять удобный интерфейс для планирования, координации и публикации событий, а также способствовать улучшению взаимодействия между участниками и повышению качества мероприятий.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- проанализировать предметную область;
- провести анализ программных продуктов-аналогов;
- выделить функциональные и нефункциональные требования к разрабатываемой системе;
- разработать необходимые диаграммы в нотации UML;
- спроектировать архитектуру автоматизированной системы;
- смоделировать пользовательский интерфейс;
- провести планирование и оценку бюджета разработки системы;
- осуществить кодирование спроектированной системы;
- провести тестирование разработанной системы и выполнить оценку качества кода.

После выполнения указанных задач необходимо осуществить анализ проделанной работы и сформулировать выводы о достижении поставленной цели.

1 Аналитическая часть

Для разработки автоматизированной системы для организации мероприятий и встреч в сообществе нужно изучить предметную область и программные продукты-аналоги. Также нужно выделить функциональные и нефункциональные требования.

1.1 Анализ предметной области

В наше время управлять информацией о мероприятиях становится все сложнее. Газеты, афиши и посты в соцсетях имеют свои минусы. А главной проблемой является отсутствие одного места со всей нужной информацией о мероприятиях. Из-за этого информация часто повторяется, и организовать мероприятия становится сложнее.

У участников есть проблемы с поиском надежной и актуальной информации. Источники информации разбросаны по разным местам, что делает поиск неудобным, данные могут быть уже неактуальными.

В современном мире, где все стремится к автоматизации, разработка системы для создания и управления мероприятиями становится необходимой. Эта система упростит общение между организаторами и участниками.

Данная система ориентирована на русскоязычных пользователей. Основными ролями системы будут организаторы и участники мероприятий. Организаторы смогут создавать и управлять мероприятиями, общаться с участниками и получать обратную связь. Участники будут искать мероприятия, подписываться и принимать участие в мероприятиях.

Основные задачи системы:

- создание инструмента для хранения и управления мероприятиями;
- обеспечение удобного поиска мероприятий;
- организация коммуникации между организаторами и участниками.

Таким образом, разработка данной системы является актуальной задачей, она поможет улучшить процессы организации мероприятий и коммуникации организаторов и участников.

1.2 Анализ продуктов-аналогов

В ходе разработки данной системы был проведен анализ существующих аналогов. Ниже приведен обзор некоторых продуктов.

Zoho Backstage – это комплексный инструмент для управления мероприятиями, который помогает организаторам планировать, продвигать и проводить различные события [1].

Преимущества и недостатки данного продукта приведены в таблице 1.

Таблица 1 – Преимущества и недостатки Zoho Backstage

Преимущества	Недостатки
Возможность интеграции с другими приложениями Zoho и внешними инструментами	В бесплатной версии функционал ограничен, а дополнительные опции стоят слишком дорого
Удобный интерфейс	Отсутствие поддержки русского языка
Возможность покупки билетов	Сложная регистрация
	Не подходит для создания некоммерческих и небольших мероприятий

Интерфейс данного программного продукта представлен на рисунке 1.

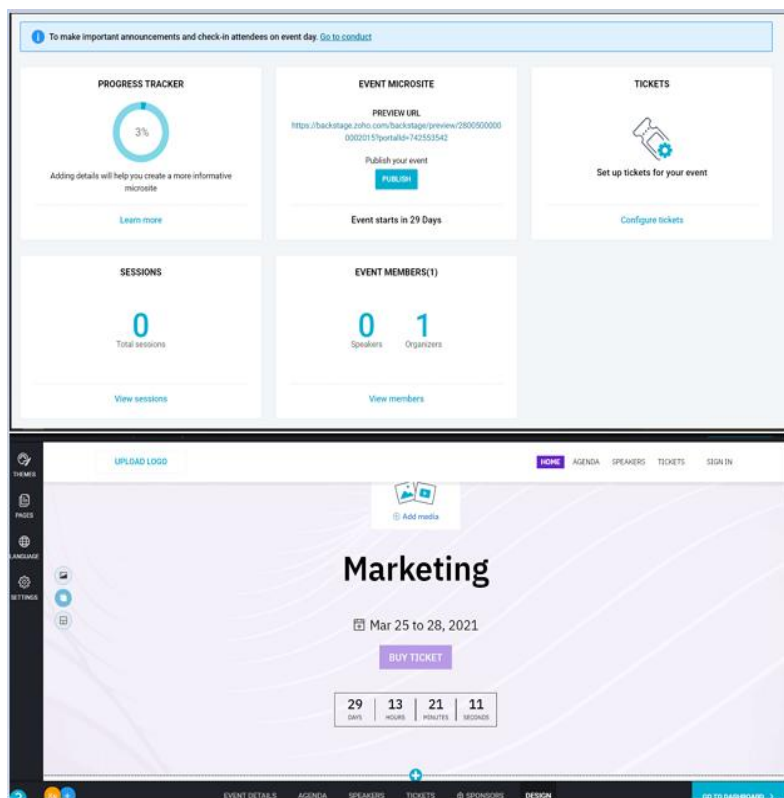


Рисунок 1 – Интерфейс Zoho Backstage

Event Rocks – это платформа и конструктор приложений для организации и проведения мероприятий. Помимо мобильного приложения, позволяющего участникам мероприятий выполнять различные действия, имеется версия Event Rocks, адаптированная для веб-браузера [2].

Сильные и слабые стороны данной платформы представлены в таблице 2.

Таблица 2 – Преимущества и недостатки Event Rocks

Преимущества	Недостатки
Полностью бесплатный продукт	Устаревший интерфейс
Наличие календаря событий	Ограниченный функционал в веб-версии
Поддержка русского языка	Сложности в использовании
Возможности для интеграции спонсоров	

Интерфейс Event Rocks представлен на рисунке 2.

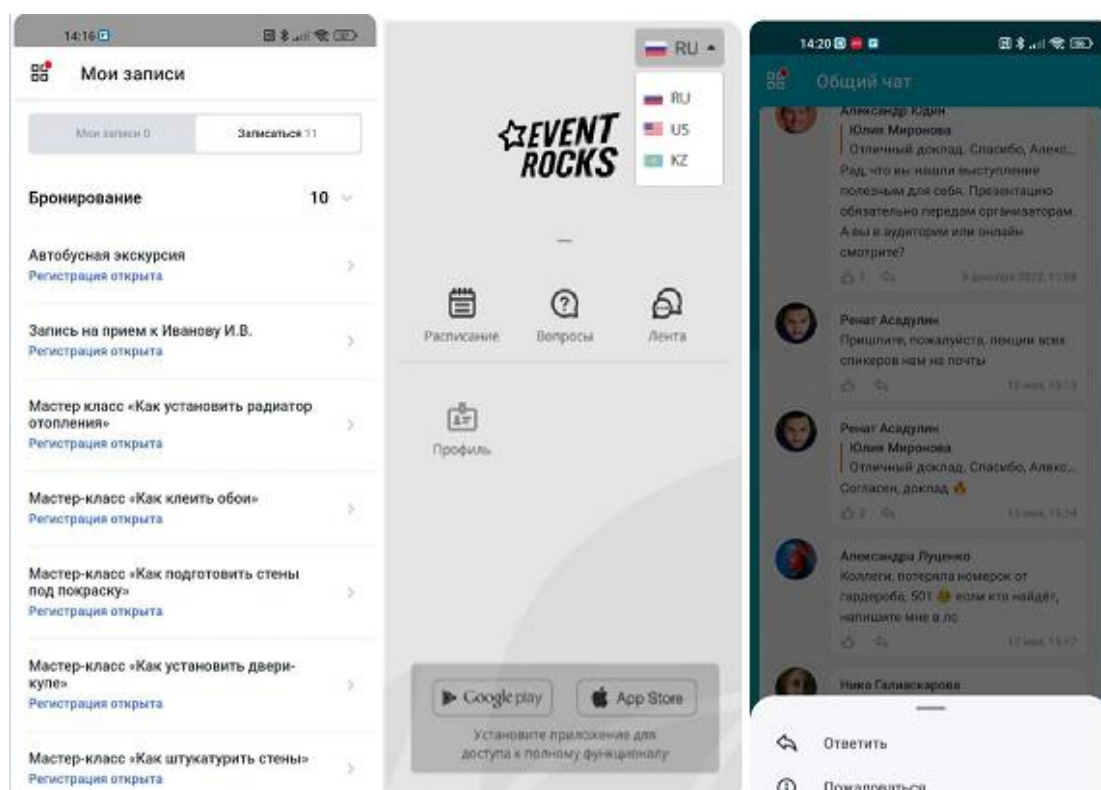


Рисунок 2 – Интерфейс Event Rocks

Svent – это программное обеспечение для планирования, управления и проведения очных, виртуальных и гибридных мероприятий, конференций и

совещаний. Платформа помогает планировать бюджет, подбирать локации, поставщиков и многое другое [3].

Плюсы и минусы данного продукта-аналога приведены в таблице 3.

Таблица 3 – Преимущества и недостатки Cvent

Преимущества	Недостатки
Предоставление детального анализа мероприятий	Для новых пользователей может быть сложно разобраться во всем многообразии функций, что потребует времени для ознакомления
Мобильное приложение совместимо с настольной версией	Полностью платный продукт, стоимость может достигать нескольких тысяч долларов в месяц
Широкие возможности интеграции с другими инструментами	Отсутствует русская локализация

Интерфейс данного продукта представлен на рисунке 3.

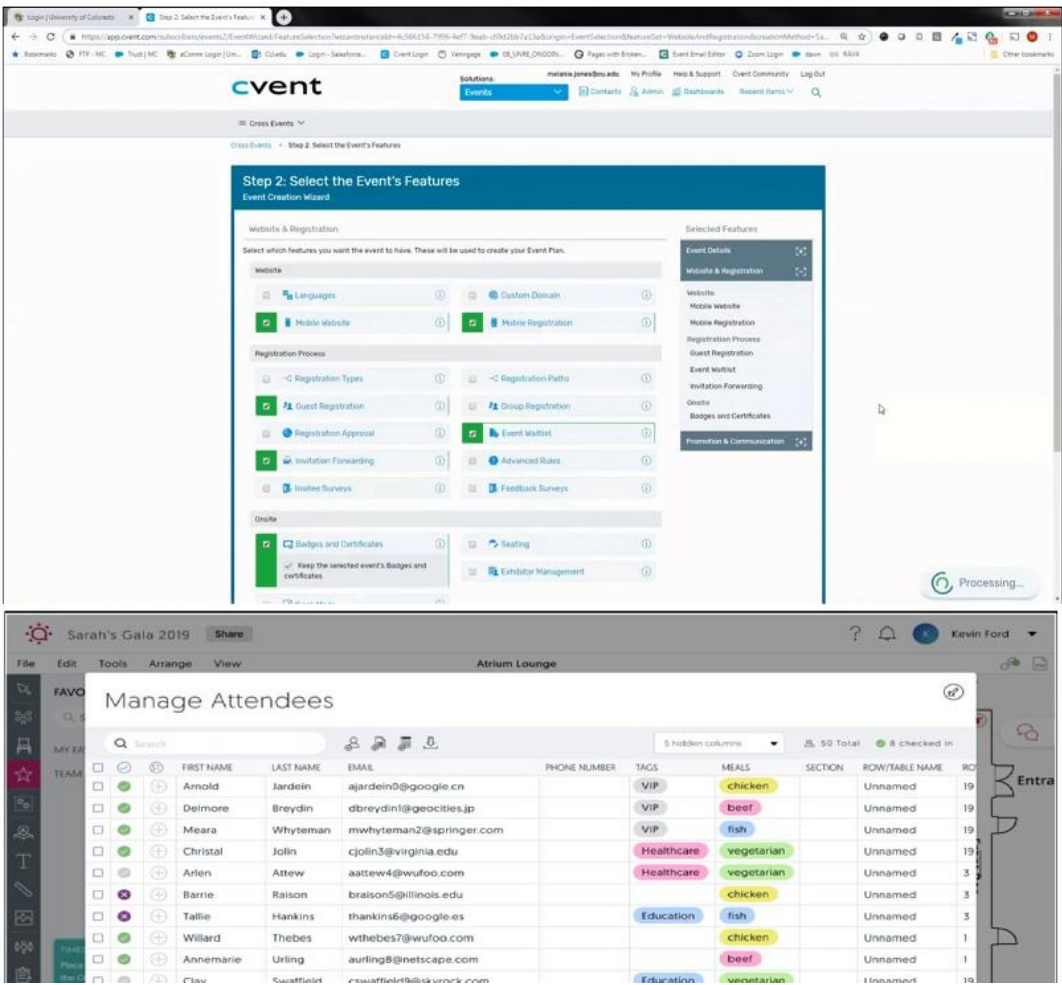


Рисунок 3 – Интерфейс Cvent

В результате анализа продуктов-аналогов были изучены их функциональные возможности, пользовательские интерфейсы, а также отзывы пользователей. Этот анализ позволил выявить их сильные и слабые стороны и определить ключевые требования к системе.

Разработанная система отличается от аналогов простотой использования, возможностью подписок на других пользователей, наличием ленты мероприятий, предоставлением бесплатного доступа ко всему функционалу, а также русской локализацией.

1.3 Постановка цели и задач

В результате анализа предметной области была определена цель выпускной квалификационной работы – создание автоматизированной системы для организации мероприятий и встреч в сообществе. Разрабатываемая система должна быть удобной и простой в использовании. Взаимодействие участников должно быть понятным и эффективным. Система должна иметь следующие функции: регистрация, авторизация, управление профилем, подписки на пользователей, создание своих мероприятий, поиск и фильтрация мероприятий, принятие участия в мероприятии.

Также важно реализовать простой и удобный интерфейс, чтобы пользователи могли легко и эффективно пользоваться системой. Необходимо обеспечить надежность и стабильность работы системы, а также предусмотреть возможность отправки приглашений и уведомлений.

1.4 Анализ требований к программным средствам

Прежде чем приступить к проектированию системы необходимо четко определить функции и принципы ее работы. Требования к программным продуктам делятся на функциональные и нефункциональные. Соответственно для разрабатываемой системы необходимо выявить и в дальнейшем учитывать следующие требования.

1.4.1 Анализ функциональных требований

В результате анализа предметной области были выделены требования к функционалу автоматизированной системы для организации мероприятий и встреч в сообществе.

В разрабатываемой системе должна быть возможность:

- зарегистрироваться;
- войти в аккаунт;
- редактировать данные профиля
- просмотреть список подписок и подписчиков;
- подписаться на пользователя;
- создать мероприятие;
- редактировать мероприятие;
- осуществить поиск мероприятий в ленте;
- принять участие в мероприятии;
- оставить комментарий к мероприятию.

Для разработки программной системы, исходя из заданных функциональных требований, была построена диаграмма вариантов использования (UML). Эта диаграмма отображает доступный функционал для каждой группы пользователей. Ее ключевые компоненты – актеры и прецеденты.

Актер представляет собой роль внешнего по отношению к системе объекта – человека, другой системы, подсистемы или класса, – взаимодействующего с системой. Прецедент же описывает последовательность действий системы, приводящую к конкретному результату, заметному для актера, и отражает взаимодействие актеров и системы [4].

Чтобы сделать диаграмму вариантов использования для создаваемой системы более понятной и наглядной, было принято решение разделить ее на две части. На рисунке 4 показан основной набор функций, доступных пользователю.

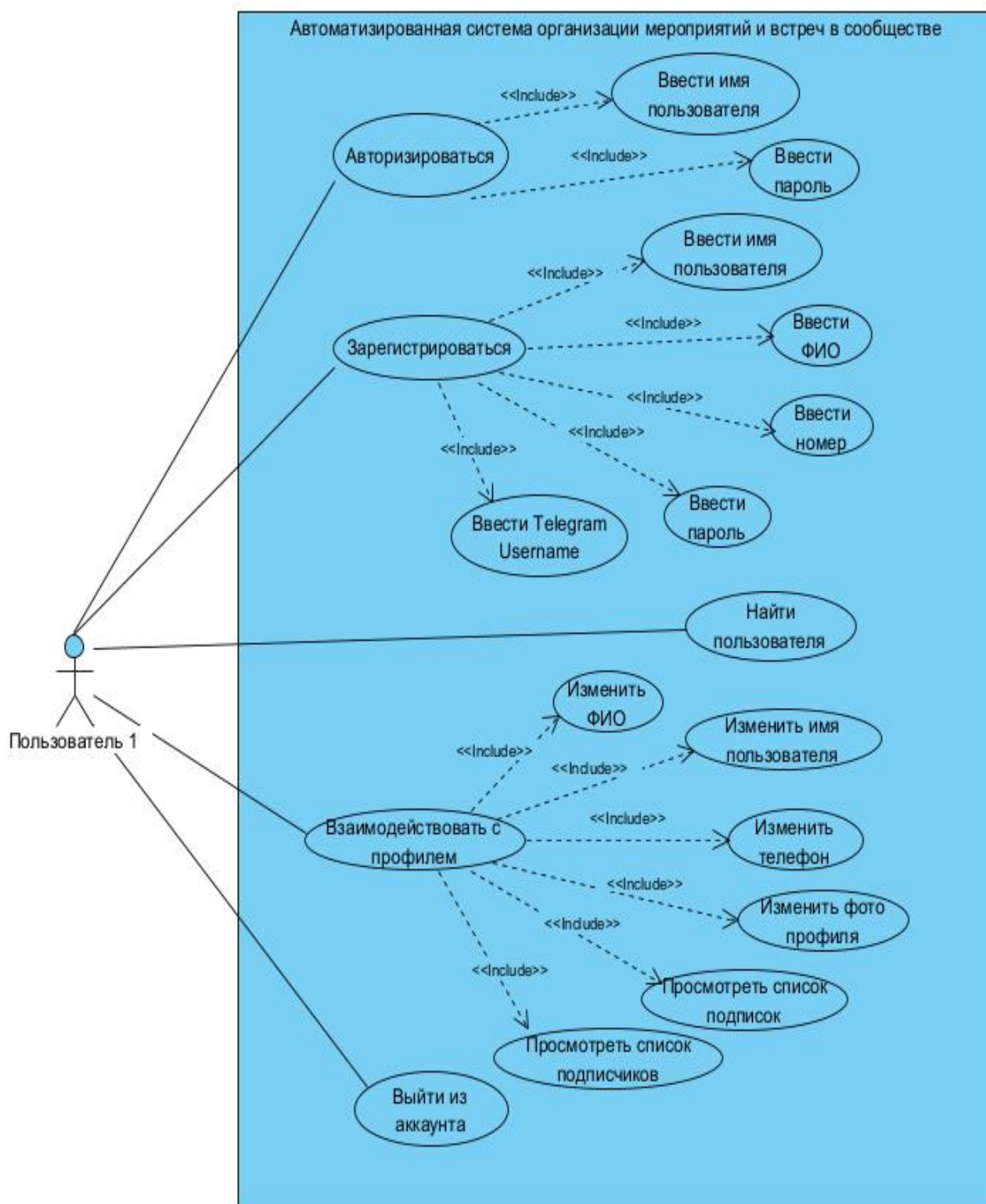


Рисунок 4 – Диаграмма вариантов использования для базовых функций системы

На рисунке 5 представлены функции системы, связанные с управлением мероприятиями.

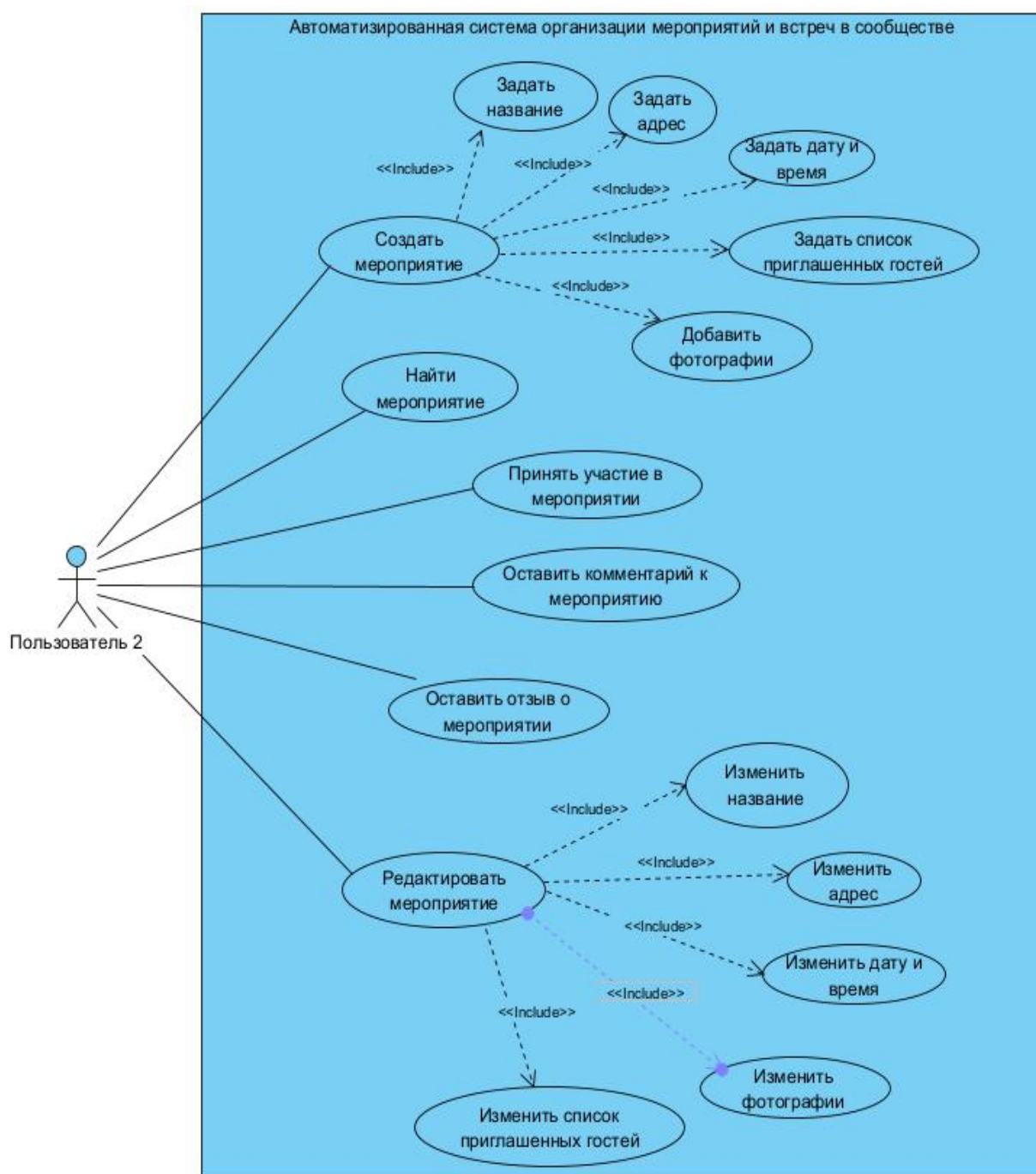


Рисунок 5 – Диаграмма вариантов использования для функций управления мероприятиями

Далее приведено описание основных вариантов использования для актеров системы.

Описание варианта использования «Авторизироваться» представлено на рисунке 6.

Вариант использования	Авторизироваться
ID	1
Краткое описание	Пользователь вводит имя и пароль, чтобы войти в систему
Главные актёры	Пользователь 1
Второстепенные актёры	Нет
Предусловия	1. Пользователь с таким именем и паролем существует; 2. Имя и пароль проходят валидацию.
Основной поток	1. Пользователь переходит на экран авторизации; 2. Пользователь указывает имя; 3. Пользователь указывает пароль.
Постусловия	Пользователь вошел в систему, ему доступен функционал

Рисунок 6 – Вариант использования «Авторизироваться»

Описание варианта использования «Создать мероприятие» представлено на рисунке 7.

Вариант использования	Создать мероприятие
ID	2
Краткое описание	Пользователь указывает всю информацию о мероприятии и выбирает список гостей
Главные актёры	Пользователь 2
Второстепенные актёры	Нет
Предусловия	1. Пользователь осуществил вход в систему.
Основной поток	1. Пользователь переходит на экран создания мероприятия; 2. Пользователь задает название мероприятия; 3. Пользователь задает адрес мероприятия; 4. Пользователь задает дату и время мероприятия; 5. Пользователь добавляет фотографии к мероприятию; 6. Пользователь выбирает приглашенных гостей. 7. Пользователь публикует мероприятие;
Постусловия	Мероприятие создано и добавлену в ленту подписчиков, приглашения отправлены

Рисунок 7 – Вариант использования «Создать мероприятие»

Описание варианта использования «Принять участие в мероприятии» представлено на рисунке 8.

Вариант использования	Принять участие в мероприятии
ID	3
Краткое описание	Пользователь выбирает мероприятие и принимает участие в нем
Главные актёры	Пользователь 2
Второстепенные актёры	Нет
Предусловия	1. Пользователь вошел в систему; 2. Пользователь подписан на организатора мероприятия.
Основной поток	1. Пользователь выбирает мероприятие в ленте; 2. Пользователь ознакомливается с информацией о мероприятии; 3. Пользователь нажимает кнопку «Принять участие».
Постусловия	Пользователь записан в участники мероприятия

Рисунок 8 – Вариант использования «Принять участие в мероприятии»

В результате анализа функциональных требований была разработана диаграмма вариантов использования и описаны сценарии основных прецедентов пользователя.

1.4.2 Анализ нефункциональных требований

Нефункциональные требования к автоматизированной системе организации мероприятий и встреч в сообществе [5]:

- поддержка работы в современных веб-браузерах последних версий: Chrome, Edge, Firefox, Safari;
- простота и интуитивность интерфейса, понятность навигации;
- адаптивность пользовательского интерфейса;
- надежность и стабильность работы системы при одновременном использовании несколькими пользователями;
- возможность масштабирования системы;
- поддержка функционала уведомлений для информирования пользователей о новых мероприятиях и обновлениях.

Таким образом, в ходе аналитической части данной работы были сформированы функциональные и нефункциональные требования для разрабатываемой системы.

2 Проектирование автоматизированной системы для организации мероприятий и встреч в сообществе

В процессе проектирования автоматизированной системы был определен набор технологий для разработки, была создана архитектура системы, продуман и смоделирован пользовательский интерфейс, а также проведен анализ бюджета разработки.

2.1 Выбор технологий разработки

Для создания визуального интерфейса системы был выбран фреймворк Next.js с использованием библиотеки Material UI (MUI). Эти инструменты были разработаны для упрощения процесса создания сложных пользовательских интерфейсов. Один из самых популярных фреймворков для этой цели – React. В основном он используется для разработки одностраничных сайтов. При работе с многостраничными сайтами у фреймворка возникают проблемы с поисковой оптимизацией (SEO). Поисковые системы испытывают трудности с анализом страниц.

React основан на принципах модульности и декларативности. Каждый элемент на странице существует независимо от других, а его внешний вид и поведение описываются с помощью декларативных методов. Такой подход отлично подходит для одностраничных приложений (SPA), но у него есть недостаток – SEO. Боты поисковых систем хорошо работают с HTML-разметкой, но испытывают трудности с разметкой, созданной с помощью JavaScript.

Next.js – это фреймворк на основе React, который поддерживает серверный рендеринг (SSR) и статическую генерацию сайтов (SSG). HTML-страницы на данном фреймворке формируются на сервере и передаются пользователю уже в готовом виде. Это ускоряет загрузку страниц и улучшает поисковую оптимизацию [6].

Маршрутизация в Next.js организована через файловую структуру. Каждая страница соответствует отдельному файлу, который может быть как

простым JavaScript-файлом, так и компонентом на React. Для создания новой страницы достаточно создать соответствующий файл или папку. Это упрощает процесс разработки, делая его более структурированным и понятным. Кроме того, такая организация улучшает поисковую оптимизацию (SEO), так как поисковые системы могут легко индексировать страницы благодаря четкой структуре.

Для разработки интерфейса выбрана библиотека Material UI. Она содержит много готовых элементов интерфейса в одном стиле. Одним из ключевых преимуществ Material UI является его адаптивность [7].

В качестве языка программирования был выбран Python, так как данный язык прост в написании, легко читается и имеет множество фреймворков и библиотек.

Серверная часть системы будет реализована с помощью фреймворка FastAPI. FastAPI – это фреймворк на Python с высокой производительностью и поддержкой асинхронных операций. FastAPI прост в использовании, а код на FastAPI легко читать. По данным TechEmpower, этот фреймворк показывает самую большую производительность среди всех фреймворков на Python [8].

Использование такого набора технологий значительно упрощает и ускоряет процесс создания приложения. Этот выбор позволяет быстро и эффективно создавать функциональные и масштабируемые приложения.

В качестве системы управления базами данных была выбрана PostgreSQL. Эта СУБД известна своей надежностью, производительностью и поддержкой расширяемых типов данных. Кроме того, PostgreSQL позволяет использовать функции, написанные на других языках программирования, что расширяет возможности для разработки и повышает гибкость системы [9].

Для изолированного запуска архитектура системы разделена на три Docker-контейнера: серверная часть, клиентский интерфейс и база данных.

Диаграмма развертывания системы показана на рисунке 9. На ней представлено размещение основных частей системы.

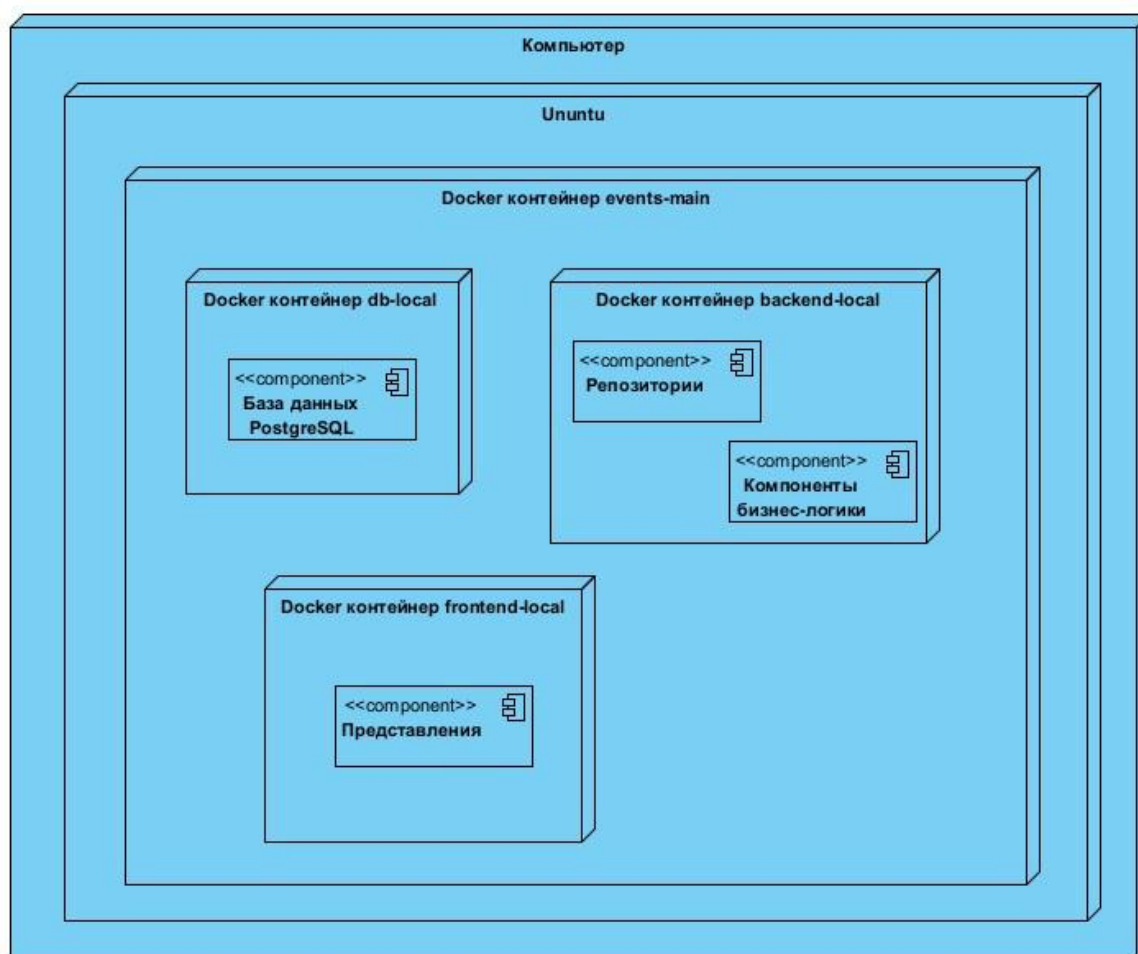


Рисунок 9 – Диаграмма развертывания разрабатываемой системы

Docker упаковывает систему и все ее компоненты в контейнеры. Эти контейнеры обеспечивают изоляцию и упрощают запуск системы. Внутри каждого контейнера находятся все важные файлы, файлы конфигурации, системные файлы.

Функционал уведомлений и приглашений будет реализован с помощью Telegram-бота, так как система рассчитана на русскоязычных пользователей.

2.2 Проектирование архитектуры приложения

Для создания системы необходимо продумать надежную и удобную архитектуру, которую будет легко масштабировать. Для этого подходит методология MVC (Model, View, Controller). MVC – это подход, который помогает связывать интерфейс с данными и упорядочивать код.

При использовании MVC программа делится на три основные части: модель, представление и контроллер.

Модель – это уровень, который считается самым низким по сравнению с уровнями представления и контроллера. Он в первую очередь представляет данные для пользователя и определяет хранение всех объектов данных приложения. Эта часть отвечает за бизнес-логику приложения и управление данными. Она хранит информацию о том, как данные будут обрабатываться и передаваться между компонентами.

Представление – это компонент, отвечающий за визуализацию данных и взаимодействие с пользователем. Этот уровень отвечает за отображение фактического результата для пользователя. Он также обеспечивает связь между пользователем (ввод данных, запросы и т. д.) и контроллером.

Контроллер служит посредником между моделью и представлением. Этот уровень отвечает за обработку запросов. Контроллер принимает входные данные от пользователя из интерфейса, обрабатывает их с помощью бизнес-логики модели, а затем передает данные в представление [10].

На рисунке 10 изображена общая схема архитектуры MVC.

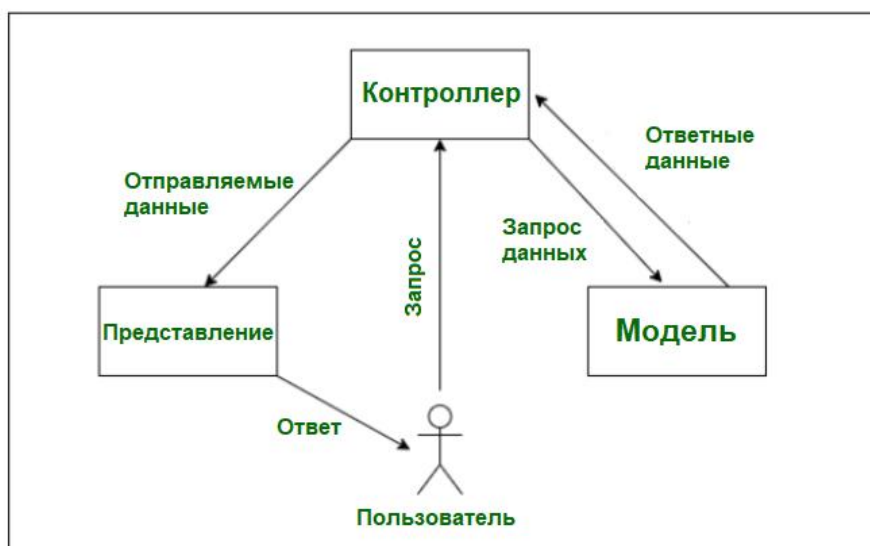


Рисунок 10 – Архитектура MVC

Визуальная часть системы будет реализовываться с помощью языка TypeScript. Роутинг будет осуществляться на основании файловой системы.

Структура каталогов и файлов будет определять пути доступа клиентской части к компонентам системы. Отдельные маршруты будут храниться в соответствующих каталогах в виде файлов с расширением .jsx.

Этот метод организации маршрутизации способствует повышению эффективности при разработке, поскольку отпадает потребность в ручном создании и настройке роутинга в компонентах. Кроме того, такой подход улучшает восприятие кода [11].

Основная бизнес-логика будет реализована в сервисах, разделенных по назначению: сервис аутентификации, сервис мероприятий, сервис приглашений и другие. За обработку запросов отвечают API маршрутизаторы, а для взаимодействия с данными из базы данных сервисы взаимодействуют с моделями, репозиториями и *pydantic* схемами. Модели хранят данные, доступ к которым предоставляется благодаря репозиториям, и логику работы с ними. *Pydantic* схемы позволяют гарантировать, что входные данные соответствуют установленным правилам и типам, а также обеспечивает их автоматическое преобразование в нужные форматы [12].

Таким образом, *frontend* будет включать в себя файловую структуру, которая будет отражать структуру маршрутизации и содержать компоненты пользовательского интерфейса. *Backend* будет состоять из контроллеров, моделей, репозиториях, маршрутизаторов, схем и сервисов.

Благодаря выбору данной архитектуры для проектируемой системы получится упростить разработку за счет разделения кода на отдельные части, разграниченные по своему назначению. Такое разделение позволит повысить поддерживаемость системы, облегчить процесс внесения изменений и тестирования программного продукта, а также поможет в дальнейшем масштабировании системы.

Для визуального представления частей системы разрабатываются диаграммы компонентов. Диаграмма компонентов — это наглядное представление структуры системы, которое позволяет понять расположение и взаимодействие ее компонентов.

Данная диаграмма помогает разработчикам лучше понимать, как система работает и какие компоненты необходимы для ее работы. Диаграмма компонентов является связующим звеном между логическим представлением проекта и его реализацией в виде кода [13].

Диаграмма компонентов для разрабатываемой системы изображена на рисунке 11.

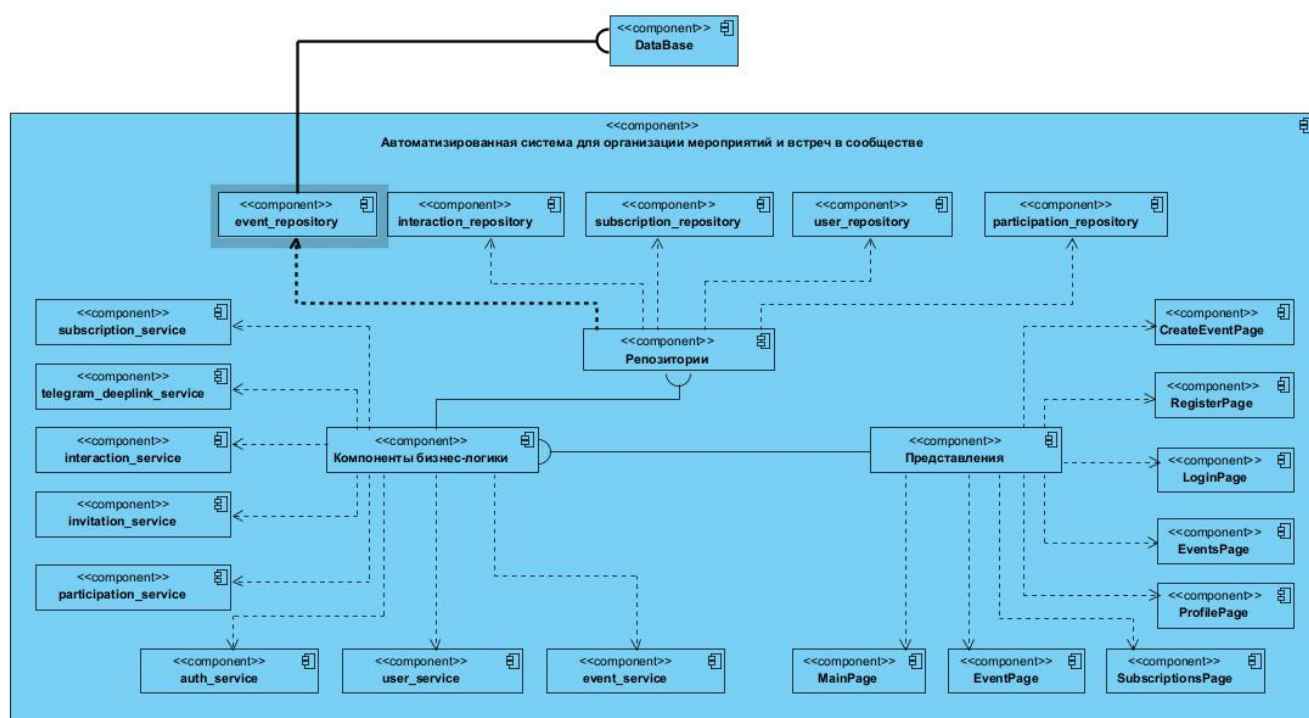


Рисунок 11 – Диаграмма компонентов разрабатываемой системы

Таким образом, разрабатываемая система имеет архитектуру, разделенную на три части. Часть представления отвечает за визуальную составляющую, компоненты бизнес-логики реализуют основной функционал системы, слой данных взаимодействует с базой данных для предоставления и обработки данных.

2.3 Проектирование пользовательского интерфейса

В процессе проектирования пользовательских интерфейсов особое значение придается двум ключевым аспектам: пользовательскому опыту (UX) и пользовательскому интерфейсу (UI). Они играют важную роль в создании

удобных, интуитивно понятных и привлекательных продуктов. Эти компоненты отвечают за следующие задачи:

- пользовательский опыт отвечает за восприятие и удобство использования системы в целом, в его рамках анализируются сценарии взаимодействия пользователя с системой, выявляются и устраняются возможные проблемы и трудности при выполнении задач, обеспечивается интуитивность действий;

- пользовательский интерфейс – это визуальная часть системы, он включает в себя создание кнопок, шрифтов, цветов и других элементов, интерфейс должен быть интуитивно понятным, легко читаемым и эстетически привлекательным.

Главной целью при создании интерфейса является объединение принципов UX и UI. Таким образом, получится удобный и понятный интерфейс, который выполняет все нужные функции. Важно также понимать поведение людей при работе с программой. Поэтому нужно понимать основные принципы создания простых и эффективных интерфейсов. Среди базовых принципов проектирования интерфейсов:

- принцип простоты означает, что информационные блоки должны быть максимально короткими и содержать только самое важное: это помогает пользователям быстро понять, что им нужно, ведь дизайн делает продукт удобным и легким для освоения, особенно для новых пользователей;

- повышение эффективности работы достигается за счет оптимизации пользовательских сценариев: автоматизация повторяющихся операций, внедрение автозаполнения форм, сокращение количества необходимых шагов позволяют достигать целей быстрее;

- ориентация на пользователя означает, что все части системы создаются с учетом нужд и интересов тех, кто будет ей пользоваться, внимание уделяется тому, что действительно важно для реальных людей, которые будут пользоваться платформой;

- наличие обратной связи является важнейшим условием

эффективного взаимодействия: система должна оперативно информировать пользователя о результате его действий, будь то успешное выполнение операции или возникшая ошибка;

– единообразие стиля всех графических элементов облегчает ориентирование в системе, создает ощущение целостности интерфейса и помогает избежать путаницы [14].

Далее представлены спроектированные макеты интерфейса системы. Интерфейс формы авторизации приведен на рисунке 12.

The image shows a login form with the following elements:

- 1**: Points to the 'Имя пользователя *' (Username) input field.
- 2**: Points to the 'Пароль *' (Password) input field.
- 3**: Points to the 'Войти' (Login) button.
- 4**: Points to the 'Нет аккаунта? Зарегистрироваться' (No account? Register) link.

Рисунок 12 – Форма авторизации

В интерфейсе стартовой страницы присутствуют следующие компоненты:

- 1 – поле для ввода имени пользователя;
- 2 – поле для ввода пароля;
- 3 – кнопка «Войти»;
- 4 – кнопка «Зарегистрироваться», которая перенаправляет на экран регистрации.

На рисунке 13 изображен макет формы регистрации.

The image shows a registration form titled "Регистрация" (Registration) with the subtitle "Создайте аккаунт для доступа к платформе" (Create an account for access to the platform). The form has a progress bar at the top with three steps: "1 Учетные данные" (Account details), "2 Личная информация" (Personal information), and "3 Завершение" (Completion). The first step is active, indicated by a blue checkmark. The form contains three input fields: "ФИО *" (Full name) with the value "Иванов Иван Иванович", "Телефон *" (Phone number) with a phone icon, and "Telegram Username (необязательно)" (Telegram Username (optional)) with the value "@username". At the bottom, there are two buttons: "Назад" (Back) with a left arrow and "Далее" (Next) with a right arrow. Five numbered callouts point to specific elements: 1 points to the "ФИО" field, 2 points to the "Телефон" field, 3 points to the "Telegram Username" field, 4 points to the "Назад" button, and 5 points to the "Далее" button.

Рисунок 13 – Форма регистрации

Визуальные компоненты формы регистрации:

- 1 – поле для ввода ФИО;
- 2 – поле для ввода телефонного номера;
- 3 – необязательное поле для ввода имени пользователя Telegram;
- 4 – кнопка «Назад» для возврата на предыдущий этап регистрации;
- 5 – кнопка «Далее» для перехода к следующему этапу регистрации.

На первом этапе пользователю необходимо ввести учетные данные для его аккаунта: имя пользователя и пароль, а также повторить пароль. Далее пользователь вводит информацию о себе: фамилия, имя, отчество и имя пользователя в Telegram по желанию. Личные данные в любой момент можно изменить на странице профиля. После входа в аккаунт пользователю становится доступен функционал системы, который распределен по разделам. Общий макет интерфейса системы представлен на рисунке 14.

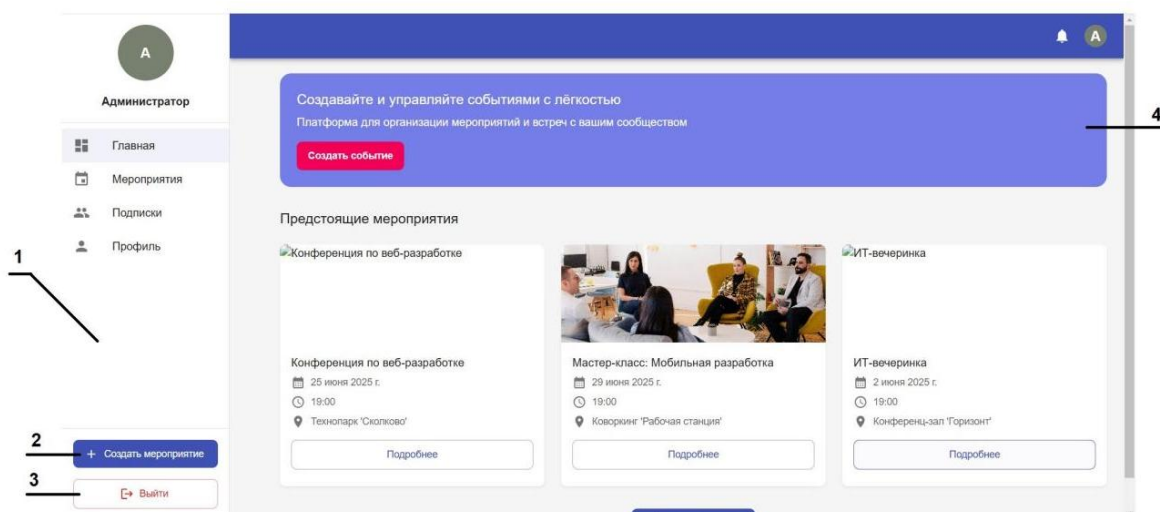


Рисунок 14 – Общий макет интерфейса системы

Состав общего макета интерфейса:

- 1 – меню разделов;
- 2 – кнопка создания мероприятия в быстром доступе;
- 3 – кнопка выхода из аккаунта;
- 4 – экран с основным контентом.

На рисунке 15 изображен макет экрана создания мероприятия.

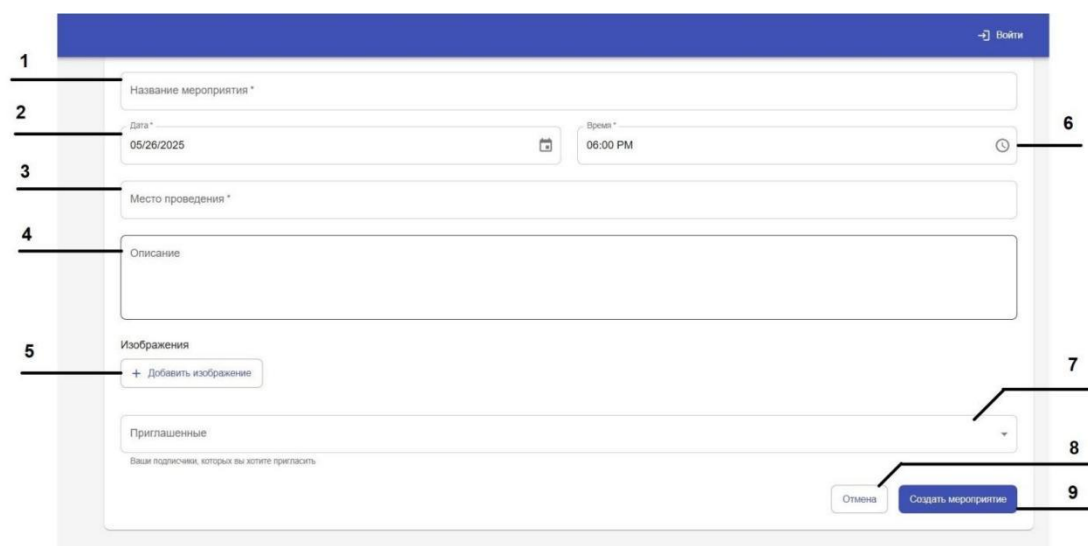


Рисунок 15 – Макет экрана создания мероприятия

Макет экрана создания мероприятия состоит из следующих элементов:

- 1 – поле для ввода названия мероприятия;
- 2 – поле для ввода даты мероприятия;
- 3 – поле для ввода адреса мероприятия;
- 4 – поле для ввода описания мероприятия;
- 5 – кнопка для добавления изображения к мероприятию;
- 6 – поле для ввода времени мероприятия;
- 7 – кнопка выбора приглашенных гостей;
- 8 – кнопка отмены создания мероприятия;
- 9 – кнопка создания мероприятия.

На рисунке 16 изображен макет личного кабинета.

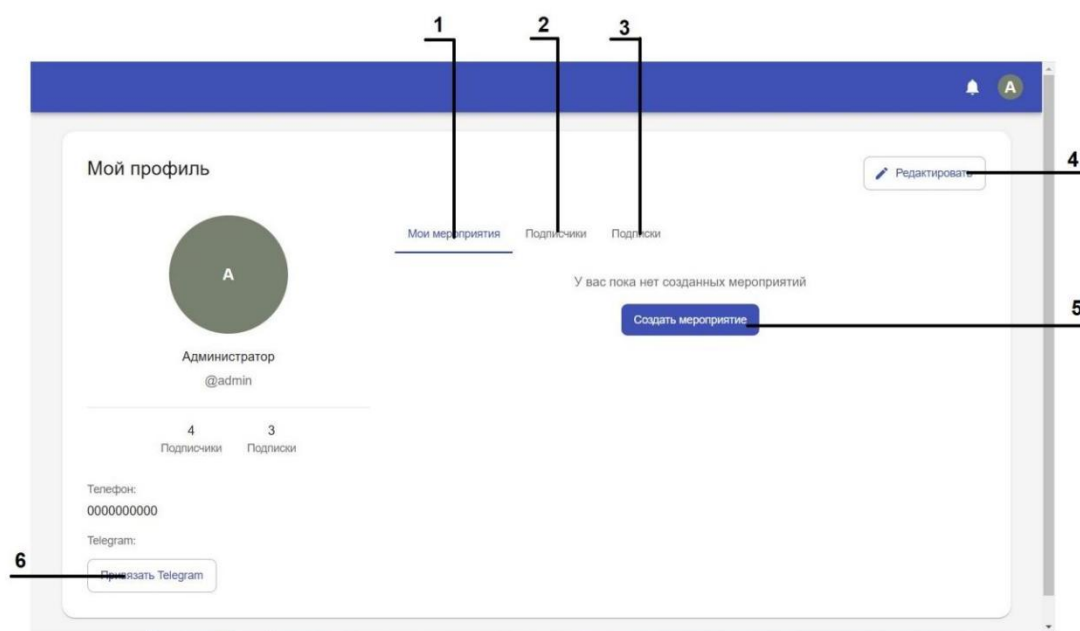


Рисунок 16 – Макет личного кабинета

Макет личного кабинета содержит следующие элементы:

- 1 – раздел «Мои мероприятия»;
- 2 – раздел «Подписчики»;
- 3 – раздел «Подписки»;
- 4 – кнопка редактирования профиля;
- 5 – кнопка создания мероприятия;
- 6 – кнопка привязки Telegram.

На рисунке 17 изображен макет интерфейса раздела мероприятий.

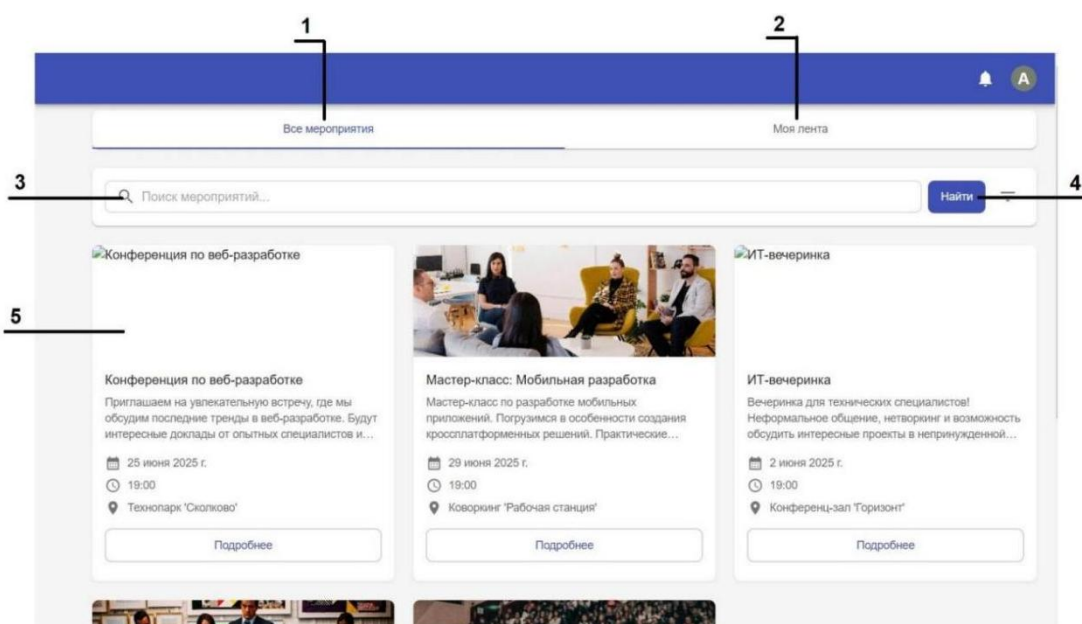


Рисунок 17 – Макет интерфейса раздела мероприятий

Макет интерфейса раздела мероприятий включает в себя следующие КОМПОНЕНТЫ:

- 1 – раздел «Все мероприятия»;
- 2 – раздел «Моя лента»;
- 3 – поле поиска мероприятий;
- 4 – кнопка «Найти»;
- 5 – список мероприятий.

На рисунке 18 изображен макет страницы мероприятия.

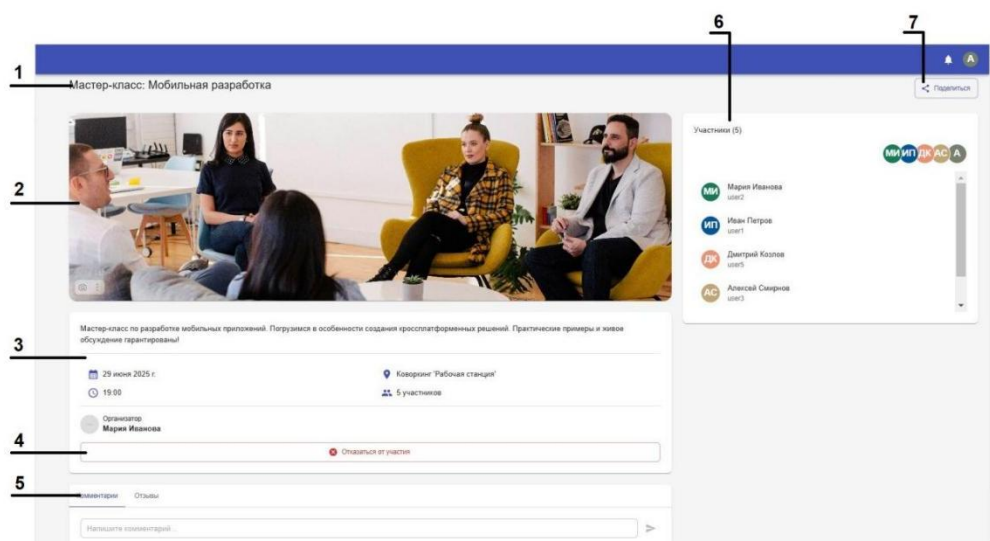


Рисунок 18 – Макет страницы мероприятия

Макет страницы мероприятия состоит из следующих компонентов:

- 1 – название мероприятия;
- 2 – фотография к мероприятию
- 3 – основная информация о мероприятии;
- 4 – кнопка принятия или отказа от участия;
- 5 – раздел комментариев и отзывов;
- 6 – список участников;
- 7 – кнопка «Поделиться».

На рисунке 19 изображен макет раздела подписок.

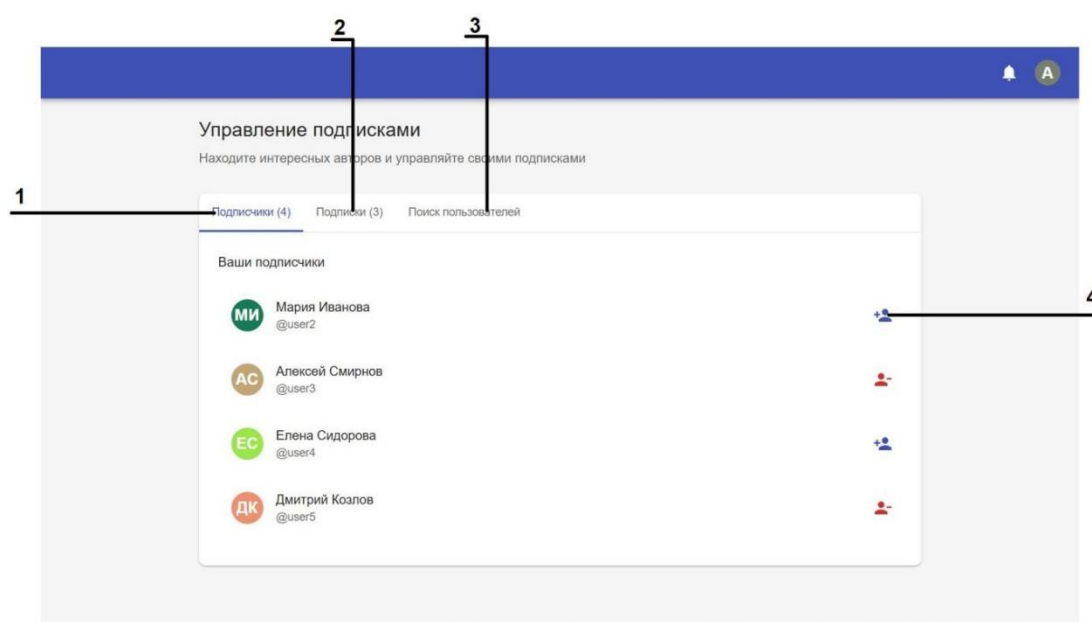


Рисунок 19 – Макет раздела подписок

Макет раздела подписок включает в себя следующие элементы:

- 1 – раздел подписчиков;
- 2 – раздел подписок;
- 3 – раздел поиска пользователей;
- 4 – кнопки подписки или отписки.

На данном этапе проектирования были созданы основные макеты интерфейса системы с учетом принципов простоты и понятности. Данные прототипы соответствуют требованиям минимализма, удобства и эффективности в использовании. На основе этих макетов будет реализована

визуальная составляющая автоматизированной системы для организации мероприятий и встреч в сообществе.

2.4 Оценка бюджета разработки системы

В любом проекте, включая разработку программного продукта, оценка бюджета зависит от множества факторов. Среди них:

- квалификация разработчиков: опытные специалисты стоят дороже, но могут ускорить процесс разработки и снизить количество ошибок;
- сроки проекта: если проект нужно сделать быстро, то могут потребоваться деньги на ускорение работы;
- интеграция с другими системами: если продукт должен работать вместе с другими продуктами, денег на разработку потребуется больше;
- дизайн: красивый дизайн привлекает больше пользователей, но делает разработку дороже;
- тестирование и качество: нужно время и деньги, чтобы проверить качество продукта.

Цена создания автоматизированной системы может быть разной и зависит от ряда факторов и особенностей, поэтому стоимость необходимо рассчитывать индивидуально.

Для успешной реализации автоматизированной системы для организации встреч и мероприятий в сообществе, требуется участие следующих специалистов: руководителя проекта, бизнес-аналитика, системного архитектора, программиста серверной части (backend), разработчика клиентского интерфейса (frontend) и инженера по тестированию.

Руководитель проекта осуществляет общее руководство процессом разработки. В его компетенции – постановка целей и задач, координация взаимодействия между участниками команды, определение приоритетов задач и контроль соблюдения сроков. Этот специалист общается с заказчиками, обеспечивает распределение заданий между членами команды и контролирует использование бюджета [15].

Аналитик занимается сбором требований к создаваемой системе: изучает особенности целевой предметной области, анализирует бизнес-процессы сообщества и формализует специфику будущего продукта. Он отвечает за подготовку технической документации – от функциональных требований до структурных схем сущностей и процессов, разрабатывает прототипы пользовательских интерфейсов или архитектурные решения.

Разработчик серверной части отвечает за программную реализацию логики приложения на стороне сервера. В его задачи входит обеспечение надежного и стабильного взаимодействия с базой данных для обеспечения доступа к данным, создание прикладных программных интерфейсов (API) и интеграция внешних сервисов.

Специалист по фронтенд-разработке занимается созданием всех видимых элементов приложения, включая структуру, дизайн, интерактивные компоненты и анимации. Он также отвечает за то, чтобы клиентская часть продукта корректно отображалась и функционировала во всех современных браузерах, обеспечивая единый и качественный пользовательский опыт на различных устройствах.

Для визуализации всех этапов разработки приложения и оценки затраты времени и ресурсов на каждый этап было принято решение разработать диаграмму Ганта с помощью ПО для управления проектами ProjectLibre.

Диаграмма Ганта – это инструмент, который помогает планировать и управлять проектами. Она представляет собой график, на котором по горизонтальной оси расположены задачи, а по вертикальной – их длительность. Задачи располагаются слева, а их продолжительность – справа.

Этот инструмент часто используется для наглядного представления процесса работы над проектом и контроля за выполнением всех задач [16].

Для оценки бюджета разработки необходимо в определить список задач разработки, их длительности, последовательность и ресурсы для их выполнения. Ресурсы, необходимые для реализации разрабатываемой системы, представлены на рисунке 20.

		Название	Тип	Максимальные единицы	Стандартная ставка	Ставка сверхурочных	Затраты на одно испол...	Начислять	Основной календарь
1		Менеджер проекта	Работа	100 %	900,00 Р/час	300,00 Р/час	0,00 Р Пропорционально	Пятидневка	
2		Аналитик	Работа	100 %	800,00 Р/час	250,00 Р/час	0,00 Р Пропорционально	Пятидневка	
3		Проектировщик	Работа	100 %	800,00 Р/час	200,00 Р/час	0,00 Р Пропорционально	Пятидневка	
4		Backend-разработчик	Работа	100 %	1200,00 Р/час	500,00 Р/час	0,00 Р Пропорционально	Пятидневка	
5		Frontend-разработчик	Работа	100 %	900,00 Р/час	400,00 Р/час	0,00 Р Пропорционально	Пятидневка	
6		Тестировщик	Работа	100 %	900,00 Р/час	350,00 Р/час	0,00 Р Пропорционально	Пятидневка	
7		ПК	Материал		0,00 Р		50000,00 Р Начало		
8		Бумага	Материал		0,00 Р/час		800,00 Р Начало		
9		Принтер	Материал		0,00 Р		10000,00 Р Начало		

Рисунок 20 – Ресурсы проекта

При планировании проекта были учтены два основных вида ресурсов: трудовые ресурсы (работа) и материальные ресурсы.

Трудовые ресурсы включают в себя специалистов, которые выполняют различные задачи и участвуют в проекте. Их количество и количество рабочих часов, затраченных на выполнение задач, напрямую влияют на общие затраты проекта.

Материальные ресурсы представляют собой единовременные затраты на оборудование, материалы, программное обеспечение и другие необходимые элементы для реализации проекта.

Разработка автоматизированной системы для организации мероприятий и встреч в сообществе была разбита на 26 основных задач. Каждая из этих задач имеет свои временные рамки и требования по ресурсам (рисунок 21).

	Название	Продолжительн...	Начало	Окончание	Предш...	Название ресурса
1	☐ Начало разработки автоматизированной системы	56 дней?	30.05.2025, 08:00	15.08.2025, 17:00		
2	Закупка материальных ресурсов	1 день?	30.05.2025, 08:00	30.05.2025, 17:00		ПК;Принтер;Бумага
3	☐ Анализ предметной области и постановка задачи	15 дней?	02.06.2025, 08:00	20.06.2025, 17:00		
4	Анализ предметной области организации мероприятий	3 дней?	02.06.2025, 08:00	04.06.2025, 17:00	2	Менеджер проекта[50 %];Аналитик[50 %]
5	Технический анализ существующих продуктов-аналогов	2 дней?	05.06.2025, 08:00	06.06.2025, 17:00	4	Менеджер проекта[50 %];Аналитик[50 %]
6	Постановка задачи	3 дней?	09.06.2025, 08:00	11.06.2025, 17:00	5	Менеджер проекта[50 %];Аналитик[50 %]
7	Анализ функциональных требований	4 дней?	12.06.2025, 08:00	17.06.2025, 17:00	6	Менеджер проекта[50 %];Аналитик[25 %];Проектировщик[25 %]
8	Анализ нефункциональных требований	3 дней?	18.06.2025, 08:00	20.06.2025, 17:00	7	Аналитик[25 %];Менеджер проекта[50 %];Проектировщик[25 %]
9	Завершение анализа разработки	0 дней?	20.06.2025, 17:00	20.06.2025, 17:00	8	
10	☐ Проектирование автоматизированной системы	8 дней?	23.06.2025, 08:00	02.07.2025, 17:00		
11	Выбор технологии разработки	1 день?	23.06.2025, 08:00	23.06.2025, 17:00	9	Аналитик[25 %];Менеджер проекта[50 %];Проектировщик[25 %]
12	Проектирование архитектуры приложения	4 дней?	24.06.2025, 08:00	27.06.2025, 17:00	11	Проектировщик[75 %];Backend-разработчик[25 %]
13	Проектирование пользовательского интерфейса	3 дней?	30.06.2025, 08:00	02.07.2025, 17:00	12	Проектировщик[75 %];Frontend-разработчик[25 %]
14	Завершение проектирования	0 дней?	02.07.2025, 17:00	02.07.2025, 17:00	13	
15	☐ Разработка автоматизированной системы	19 дней?	03.07.2025, 08:00	29.07.2025, 17:00		
16	Разработка серверной части	12 дней?	03.07.2025, 08:00	18.07.2025, 17:00	14	Backend-разработчик
17	Разработка клиентской части	7 дней?	21.07.2025, 08:00	29.07.2025, 17:00	16	Frontend-разработчик
18	Завершение разработки	0 дней?	29.07.2025, 17:00	29.07.2025, 17:00	17	
19	☐ Тестирование автоматизированной системы	7 дней?	30.07.2025, 08:00	07.08.2025, 17:00		
20	Тестирование методом комбинаторики и попарного тест	3 дней?	30.07.2025, 08:00	01.08.2025, 17:00	18	Тестировщик
21	Анализ сложности кода	2 дней?	04.08.2025, 08:00	05.08.2025, 17:00	20	Тестировщик
22	Тестирование совместимости	2 дней?	06.08.2025, 08:00	07.08.2025, 17:00	21	Тестировщик
23	Завершение тестирования автоматизированной системы	0 дней?	07.08.2025, 17:00	07.08.2025, 17:00	22	
24	Составление и написание документации	4 дней?	08.08.2025, 08:00	13.08.2025, 17:00	23	Менеджер проекта
25	Оформление и печать документации	2 дней?	14.08.2025, 08:00	15.08.2025, 17:00	24	Менеджер проекта
26	Завершение разработки автоматизированной системы	0 дней?	15.08.2025, 17:00	15.08.2025, 17:00	25	

Рисунок 21 – Задачи проекта

После анализа всех задач, определения их последовательности и длительности, а также назначения необходимых ресурсов, была создана диаграмма Ганта. Она наглядно представляет все этапы разработки автоматизированной системы. Диаграмма Ганта позволяет не только увидеть, какие задачи выполняются одновременно, а какие последовательно, но и оценить требуемые временные и ресурсные затраты на каждый этап проекта (рисунок 22).

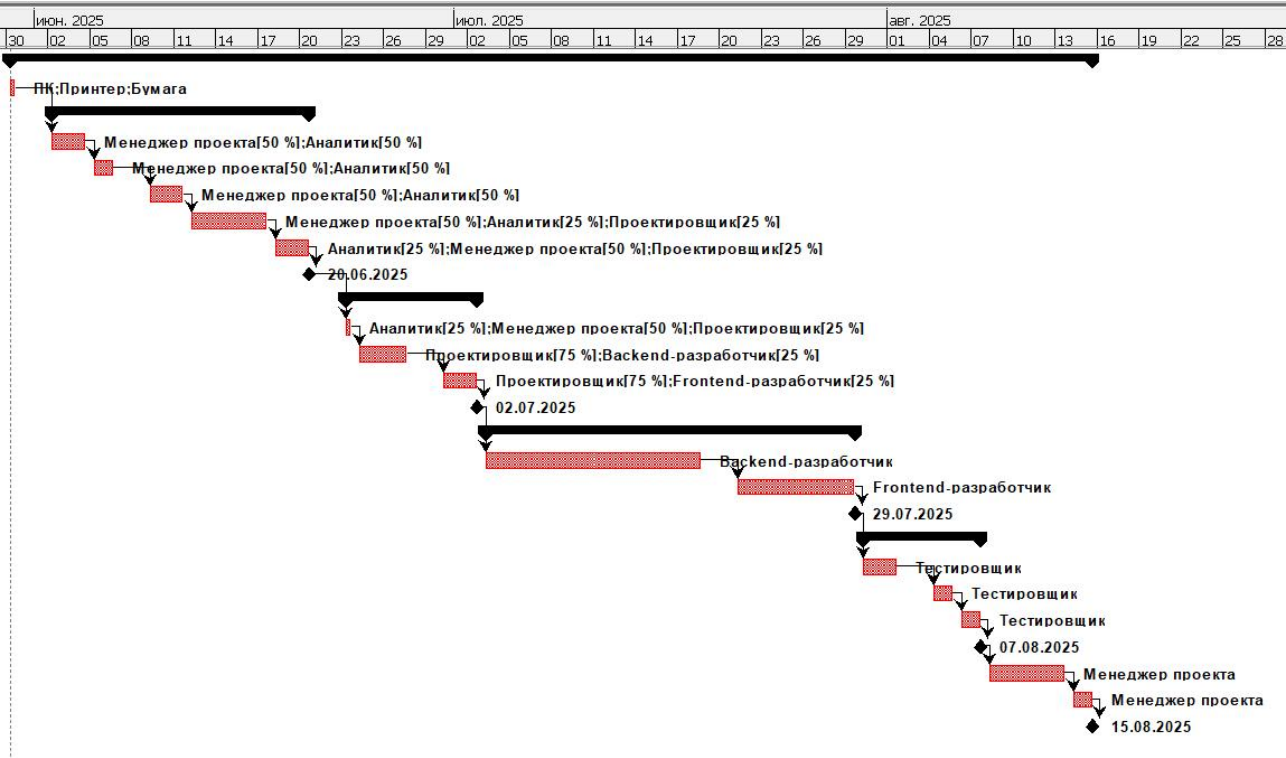


Рисунок 22 – Диаграмма Ганта проекта

На диаграмме представлено распределение ресурсов и общий план выполнения задач проекта. Видно, что основными задачами, требующими наибольшего времени и усилий, являются разработка серверной и клиентской частей системы. Эти этапы являются ключевыми для успешного завершения проекта и требуют тщательной проработки и координации между всеми участниками команды.

Составленный в программе LibreProject итоговый отчет с деталями проекта показан на рисунке 23.

Dates			
Start	30.05.2025, 08:00	Finish	15.08.2025, 17:00
Baseline Start		Baseline Finish	
Actual Start		Actual Finish	

Duration			
Scheduled	56	Remaining	56
Baseline	0	Actual	0
		Percent Complete	0 %

Work			
Scheduled	440	Remaining	440
Baseline	0	Actual	0

Costs			
Scheduled	477400,00	Remaining	477400,00
Baseline	0,00	Actual	0,00
		Variance	0,00

Рисунок 23 – Общий отчет проекта

В итоге, для того чтобы оценить стоимость разработки автоматизированной системы для организации мероприятий и встреч в сообществе, был составлен график работ с помощью программы LibreProject. Из общего отчета видно, что время, необходимое для разработки проекта, составляет 56 дней, среди которых 440 рабочих часов. Учитывая почасовую оплату трудовых ресурсов в виде специалистов и затраты на материальные ресурсы для разработки, бюджет проекта составляет 477 400 рублей.

3 Реализация автоматизированной системы для организации мероприятий и встреч в сообществе

После анализа предметной области и проектирования была разработана автоматизированная система для организации мероприятий и встреч в сообществе.

Процесс реализации данной системы разделен на три этапа:

- этап разработки серверной части, отвечающей за основной функционал и бизнес-логику;
- этап разработки клиентского интерфейса, соответствующего всем выдвинутым требованиям;
- этап организации взаимодействия между клиентской и серверной частями системы.

Код данной системы представлен в приложении А.

3.1 Реализация серверной части системы

Серверная часть системы была разработана с использованием языка программирования Python и фреймворка FastAPI, который обеспечивает удобство, высокую производительность и эффективность при работе с различными типами HTTP-запросов [17].

На стороне сервера логику маршрутизации запросов содержат файлы-роутеры. Эти файлы определяют HTTP-методы и перенаправляют запросы к соответствующим функциям, определенным в системе.

Данную работу выполняют файлы `auth.py`, `events.py`, `invitations.py`, `participations.py`, `telegram.py` и `users.py`. Пример обработки POST-запроса регистрации в файле `auth.py` представлен в листинге 1.

Листинг 1 – Программный код обработки POST-запроса

```
@router.post("/register", response_model=UserDisplay,
status_code=status.HTTP_201_CREATED)
async def register(user_data: UserCreate, db: Session =
Depends(get_db)):
    """Register a new user."""
    return AuthService.register(db, user_data)
```

В данном коде декоратор фреймворка FastAPI обозначает, что данная функция будет обрабатывать HTTP POST запросы на маршруте «/register». При совершении POST запроса FastAPI будет вызывать функцию register, которая в свою очередь обратится за методом register класса AuthService.

Пример обработчика GET-запроса представлен в листинге 2.

Листинг 2 – Программный код обработки GET-запроса

```
@router.get("/search", response_model=List[EventDisplay])
async def search_events(
    query: str = Query(..., min_length=3),
    upcoming_only: bool = False,
    skip: int = 0,
    limit: int = 10,
    db: Session = Depends(get_db)
):
    """Search events by title, location, or description."""
    return EventService.search_events(db, query, upcoming_only,
    skip, limit)
```

Таким образом, все компоненты-маршрутизаторы зависят от компонентов-сервисов.

Отдельного внимания заслуживает маршрутизатор, отвечающий за подключение Telegram-аккаунта (листинг 3).

Листинг 3 – Программный код обработки запроса привязки Telegram

```
@router.post("/link-token")
def create_telegram_link_token(
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_active_user)
):
    token = TelegramLinkService.create_link_token(db,
    current_user.id)
    bot_username = os.getenv("BOT_USERNAME")
    if not bot_username:
        # Try to get from aiogram bot instance if available
        if bot:
            import asyncio
            try:
                loop = asyncio.get_event_loop()
                bot_info = loop.run_until_complete(bot.get_me())
                bot_username = bot_info.username
            except Exception:
                bot_username = None
        # As a last resort, hardcode the username (replace with
        your bot's username)
```

```

    if not bot_username:
        bot_username = "psu_vkr_events_bot"
    link = f"https://t.me/{bot_username}?start={token}"
    return {"deep_link": link}

```

Данный код с помощью сервиса telegram_deerlink_service и контроллера telegram_controller создает ссылку на Telegram-бота. Эта ссылка содержит уникальный токен текущего пользователя, а также используется для запуска бота.

Помимо telegram_controller был разработан scheduler_controller, предназначенный для отправки напоминаний о предстоящих событиях. Функционал работы с Telegram-ботом предоставляет библиотека aiogram, а для работы с асинхронными задачами отправки уведомлений используется библиотека apscheduler.schedulers.asyncio.

Задачи работы с базой данных, обработки и представления данных выполняют модели (models), схемы (schemas) и репозитории (repositories). Модели представляют собой объекты, которые описывают данные системы и их отношения и по сути соответствуют таблицам в базе данных. Пример кода модели Comment представлен в листинге 4.

Листинг 4 – Программный код модели comment

```

from sqlalchemy import Column, ForeignKey, Integer, Text, Float,
CheckConstraint
from sqlalchemy.orm import relationship

from .base import Base, BaseModel

class Comment(Base, BaseModel):
    """Comment model for events."""
    __tablename__ = "comments"

    text = Column(Text, nullable=False)
    user_id = Column(Integer, ForeignKey("users.id",
ondelete="CASCADE"), nullable=False)
    event_id = Column(Integer, ForeignKey("events.id",
ondelete="CASCADE"), nullable=False)

    # Relationships
    user = relationship("User", back_populates="comments")
    event = relationship("Event", back_populates="comments")

```

Данный код использует библиотеку sqlalchemy для взаимодействия с

базой данных и описывает поля и отношения объекта «Комментарий».

Для валидации, сериализации данных и преобразования типов при помощи библиотеки Pydantic были разработаны схемы. Пример схемы данных для работы с пользователем представлен в листинге 5.

Листинг 5 – Программный код схемы user

```
from pydantic import BaseModel, Field, validator
from typing import Optional, List
from datetime import datetime

# Base User Schema
class UserBase(BaseModel):
    username: str
    full_name: str
    phone: str

# User Creation Schema
class UserCreate(UserBase):
    password: str

    @validator('password')
    def password_strength(cls, v):
        if len(v) < 6:
            raise ValueError('Пароль должен содержать минимум 6
СИМВОЛОВ')
        return v
```

В данном фрагменте определяются общие атрибуты пользователя и добавляется валидация пароля при создании нового пользователя.

Репозитории используют модели и схемы для предоставления методов выполнения операций создания, чтения, обновления и удаления. Фрагмент кода репозитория подписок представлен в листинге 6.

Листинг 6 – Программный код репозитория subscriptions

```
from sqlalchemy.orm import Session
from sqlalchemy import func
from typing import List, Optional

from ..models import Subscription, User

class SubscriptionRepository:
    @staticmethod
    def follow_user(db: Session, follower_id: int, followed_id:
int):
        """Create a subscription (follow a user)."""
```

```

# Check if already following
existing = db.query(Subscription).filter(
    Subscription.follower_id == follower_id,
    Subscription.followed_id == followed_id
).first()

if existing:
    return existing

# Can't follow yourself
if follower_id == followed_id:
    return None

# Create new subscription
db_subscription = Subscription(
    follower_id=follower_id,
    followed_id=followed_id
)
db.add(db_subscription)
db.commit()
db.refresh(db_subscription)
return db_subscription

```

Данный метод создает новую подписку и добавляет ее в базу данных.

Таким образом, организуется взаимодействие с хранилищем данных и работа с данными. Сам файл конфигурации базы данных представлен в листинге 7.

Листинг 7 – Файл конфигурации БД

```

DATABASE_URL = os.getenv("DATABASE_URL",
    "postgresql://postgres:postgres@localhost/events_db")

# Create session
SessionLocal = sessionmaker(False, False, engine)

# Create base class
Base = declarative_base()

# Dependency
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
from sqlalchemy.orm import Session

```

Бизнес-логика системы реализуется в сервисах. Для разрабатываемой системы были созданы следующие сервисы, отвечающие за основной

функционал: `auth_service`, `event_service`, `interaction_service`, `invitation_service`, `participation_service`, `subscription_service`, `telegram_deeplink_service` и `telegram_deeplink_service`. Код сервиса аутентификации, отвечающий за регистрацию нового пользователя, приведен в листинге 8.

Листинг 8 – Программный код сервиса `auth_service`

```
from sqlalchemy.orm import Session
from fastapi import HTTPException, status
from datetime import datetime, timedelta

from ..utils.security import verify_password, create_access_token
from ..repositories import UserRepository
from ..schemas import UserCreate, UserLogin, Token

class AuthService:
    @staticmethod
    def register(db: Session, user_data: UserCreate):
        # Check if username exists
        db_user = UserRepository.get_by_username(db,
user_data.username)
        if db_user:
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail="Имя пользователя уже занято"
            )

        # Check if phone exists
        db_user = UserRepository.get_by_phone(db, user_data.phone)
        if db_user:
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail="Телефон уже зарегистрирован"
            )

        # Create user
        return UserRepository.create(db, user_data)
```

Каждый сервис для выполнения своих функций использует методы репозитория, так реализуется бизнес-логика системы.

Для визуального представления взаимодействия сервисов и репозитория была разработана диаграмма классов, разделенная на две части. Сервисы `interaction_service` и `event_service` представлены на рисунке 24.

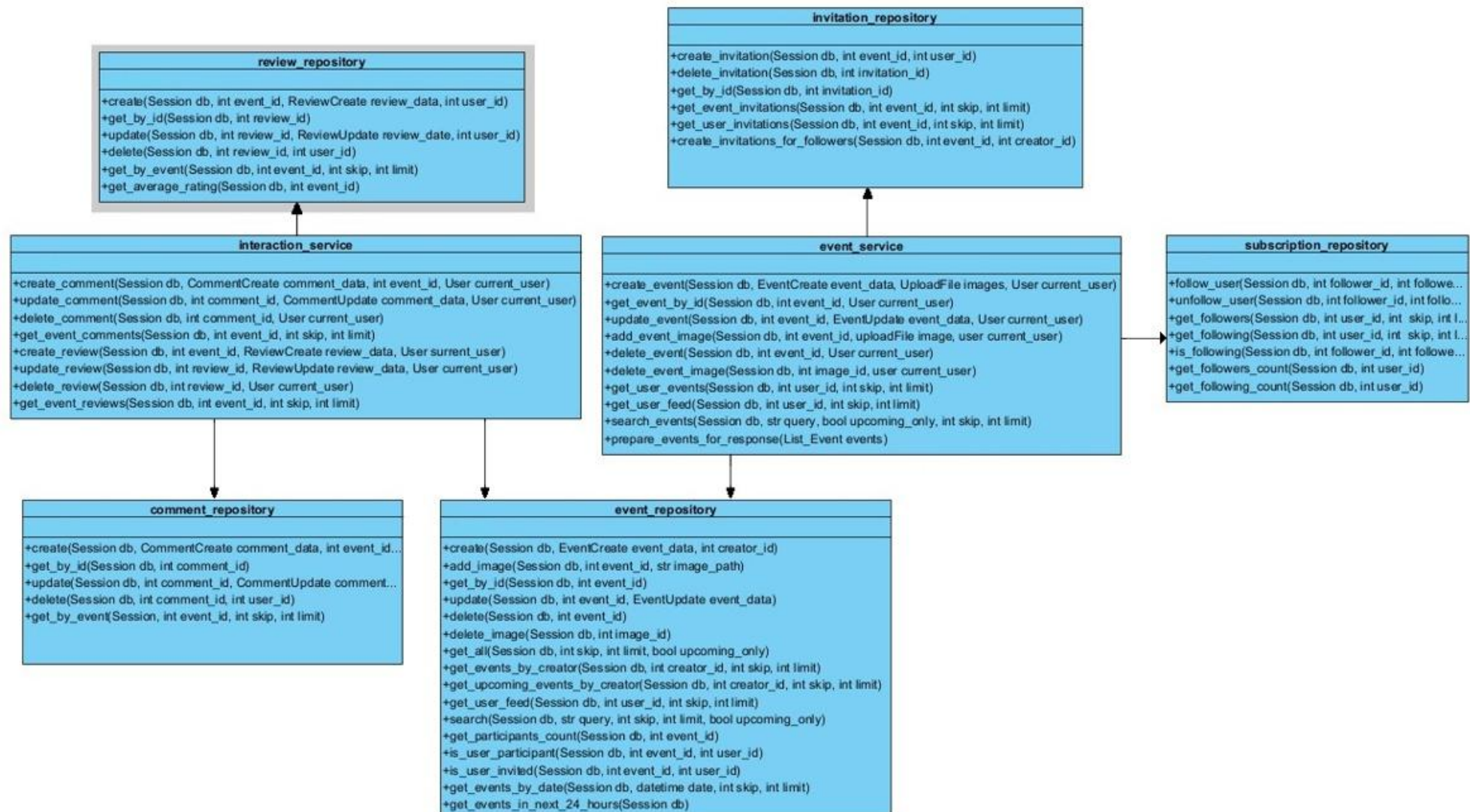


Рисунок 24 – Диаграмма классов для event_service и interaction_service

Диаграмма классов это графическое отображение классов в системе и их взаимосвязей. Она также показывает свойства классов, их методы и ограничения, применяемые к связям между ними. Диаграмма классов является итогом проектирования и основой для разработки. Классы включают в себя данные, поведение и взаимосвязи, которые определяют их функциональность и взаимодействие с другими элементами системы.

Основные компоненты диаграммы классов:

- классы: абстракции, которые описывают структуру и поведение объектов;
- атрибуты: переменные, которые принадлежат классу и описывают его состояние;
- методы: функции, которые определяют поведение класса;
- взаимосвязи между классами: различные типы отношений между классами, которые показывают, как классы взаимодействуют друг с другом;
- ограничения и стереотипы: указывают на специфические характеристики или правила, применимые к диаграмме для более точного описания классов и их взаимосвязей.

Диаграмма классов является результатом этапа проектирования и служит основой для дальнейшей разработки, включая написание кода, создание интерфейсов и реализацию функциональности.

Одним из важных преимуществ диаграммы классов является ее способность заменить ER-диаграммы при описании моделей данных. ER-диаграммы больше ориентированы на реляционные базы данных и описание структуры данных, диаграмма классов фокусируется на классах и их поведении. Это делает ее более подходящей для объектно-ориентированных систем. Диаграмма классов позволяет фиксировать не только структуру данных, но и операции, которые могут быть выполнены над этими данными, что особенно важно для динамических систем [18].

Сервисы `participation_service`, `invitation_service`, `user_service`, `auth_service` показаны на рисунке 25.

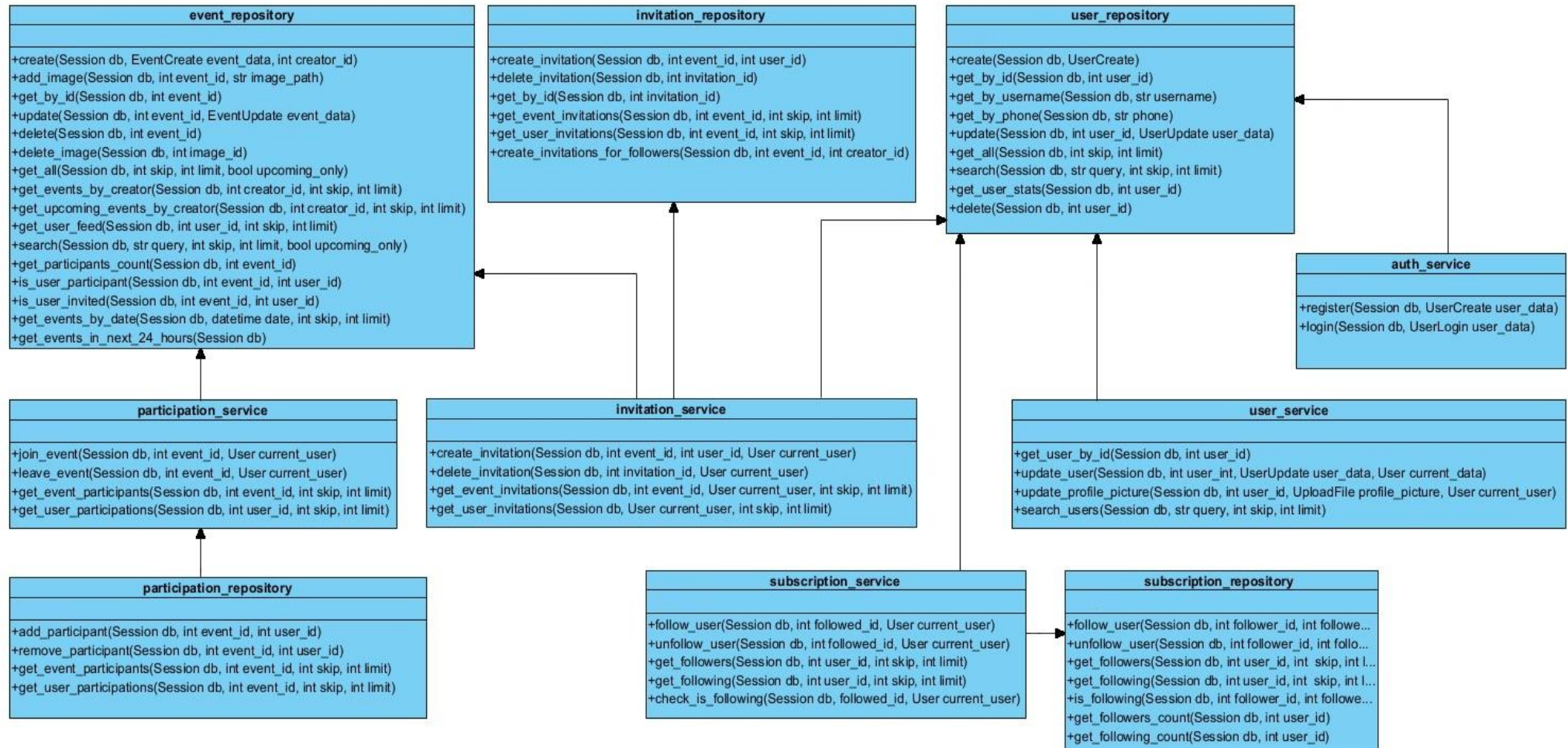


Рисунок 25 – Диаграмма классов для participation_service, invitation_service, user_service, auth_service

Помимо файлов системы, серверная часть содержит файл запуска сервера с использованием библиотеки `uvicorn` (листинг 9).

Листинг 9 – Файл запуска сервера

```
import uvicorn
import os
from dotenv import load_dotenv

load_dotenv()

HOST = os.getenv("HOST", "0.0.0.0")
PORT = int(os.getenv("PORT", "8000"))

if __name__ == "__main__":
    uvicorn.run("app.main:app", host=HOST, port=PORT, reload=True)
```

Для развертывания данной системы в Docker-контейнерах необходим файл конфигурации Docker Compose. Конфигурация для серверной части приведена на рисунке 26.

```
backend:
  build:
    context: ./backend
    dockerfile: Dockerfile.dev
  container_name: events-backend-local
  environment:
    - BASE_URL=http://localhost:3000
    - DATABASE_URL=postgresql://postgres:postgres@db:5432/events_db
    - SECRET_KEY=your-secret-key-for-local-development
    - ALGORITHM=HS256
    - ACCESS_TOKEN_EXPIRE_MINUTES=30
    - ALLOWED_ORIGINS=http://localhost:3000,http://localhost:3001
    - UPLOAD_DIR=static/uploads
    - MAX_FILE_SIZE=10485760
    - ENVIRONMENT=development
  ports:
    - "8000:8000"
  restart: unless-stopped
  volumes:
    - ./backend:/app
    - uploads_data_local:/app/static/uploads
  depends_on:
    db:
      condition: service_healthy
  networks:
    - app_network_local
  healthcheck:
    test: ["CMD", "curl", "-f", "http://localhost:8000/health"]
    interval: 30s
    timeout: 10s
    retries: 3
    start_period: 40s
```

Рисунок 26 – Конфигурация Docker Compose для серверной части

На данном этапе выполнения выпускной квалификационной работы был реализован программный код серверной части автоматизированной системы для организации мероприятий и встреч в сообществе.

3.2 Реализация пользовательского интерфейса системы

Пользовательский интерфейс системы был создан с помощью фреймворка Next.js. Этот инструмент позволяет удобно и быстро создавать производительные и масштабируемые веб-приложения.

Было принято решение разделить основной функционал на четыре раздела: главная страница, страница мероприятий, страница управления подписками и страница личного профиля. Навигация по страницам осуществляется с помощью меню, расположенного в левой части экрана.

При переходе в систему пользователь первым делом видит экран авторизации, где он может ввести имя и пароль от аккаунта, а также нажать кнопку «Зарегистрироваться», если у него еще нет аккаунта (рисунок 27).

Рисунок 27 – Экран авторизации

При нажатии кнопки регистрации пользователь попадает на экран регистрации, где ему нужно будет задать имя и пароль от аккаунта и ввести личные данные (рисунок 28).

Регистрация
Создайте аккаунт для доступа к платформе

1 Учетные данные — 2 Личная информация — 3 Завершение

Имя пользователя *
admin

Пароль *
Testpassword01

Подтверждение пароля *

← Назад Далее →

Уже есть аккаунт? Войти

Регистрация
Создайте аккаунт для доступа к платформе

✓ 1 Учетные данные — 2 Личная информация — 3 Завершение

ФИО *
Резепов Александр Александрович

Телефон *
89632555847

Telegram Username (необязательно)
@Asodgiwe

← Назад Далее →

Рисунок 28 – Экран регистрации

После входа в аккаунт пользователь перенаправляется на главную страницу, где он может просмотреть список предстоящих мероприятий. Перейти в другой раздел системы он может с помощью бокового меню, а для быстрого перехода к созданию мероприятия предусмотрена кнопка в левой нижней части экрана (рисунок 29).

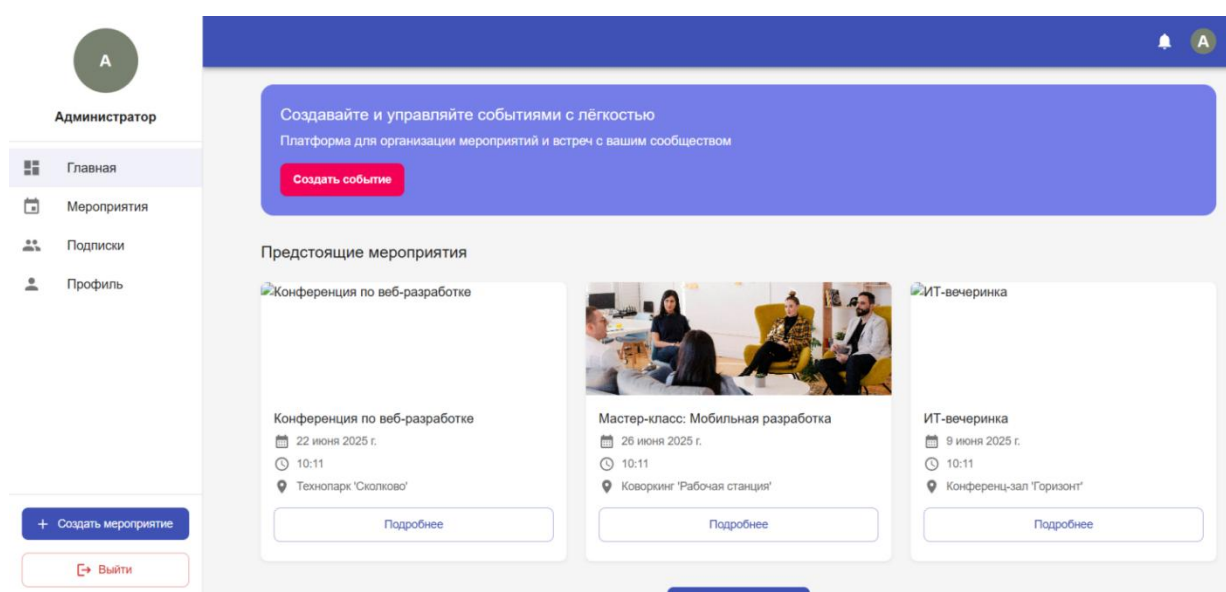


Рисунок 29 – Главная страница

На странице мероприятий находится список всех созданных мероприятий и список мероприятий из подписок. Также можно искать мероприятия через поле поиска или фильтровать события по дате (рисунок 30).

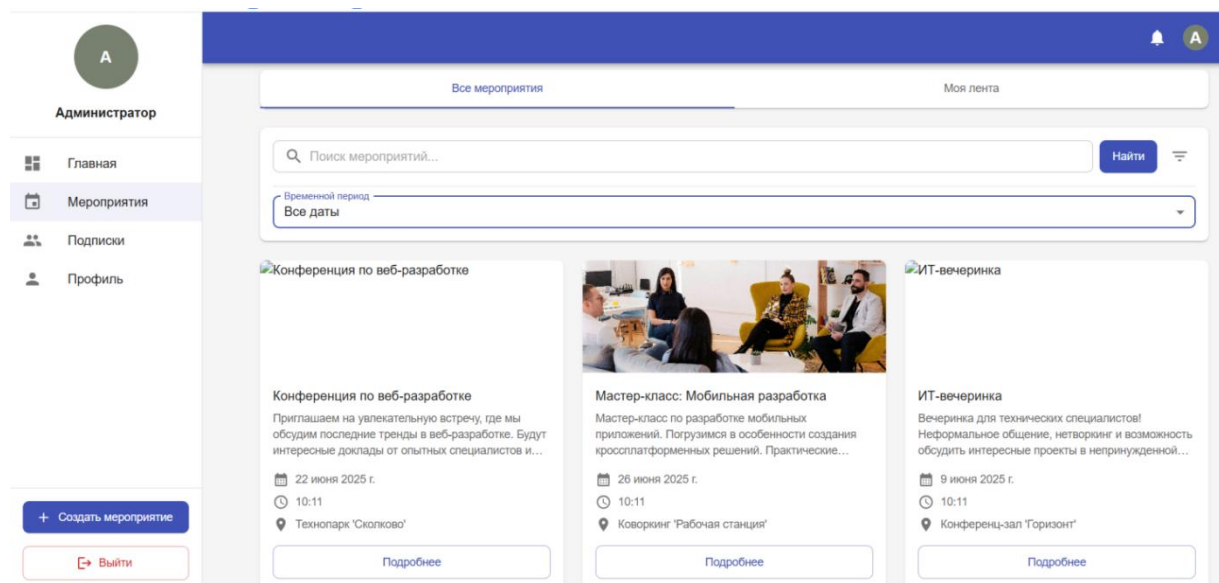


Рисунок 30 – Страница мероприятий

На странице подписок находятся список подписчиков, список подписок и вкладка поиска пользователей. У каждого пользователя в списке есть кнопка, чтобы подписаться или отписаться (рисунок 31).

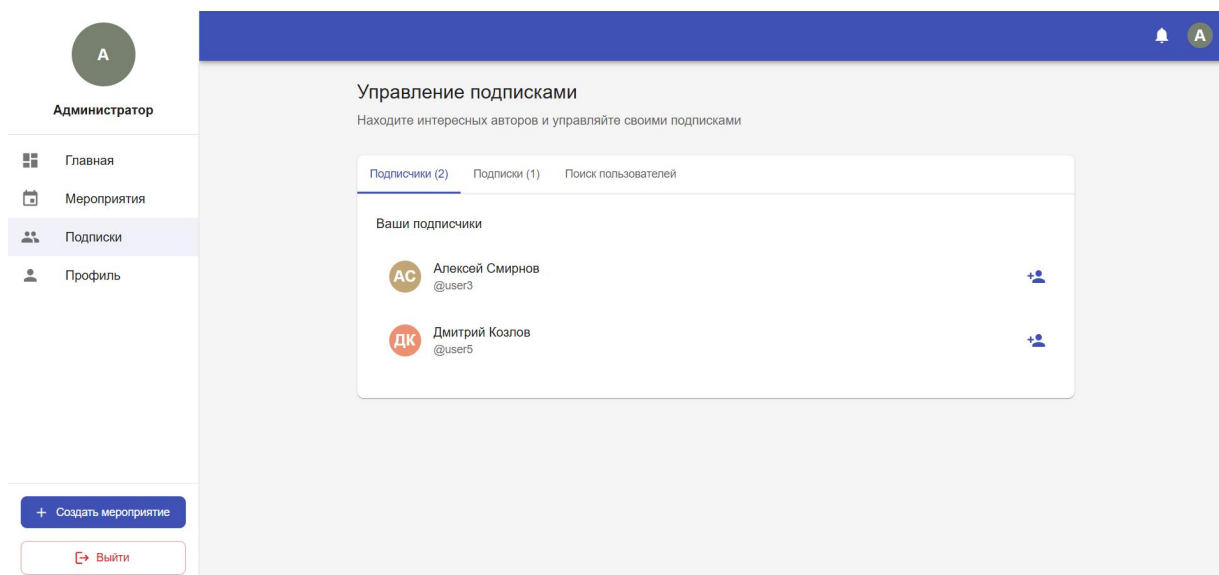


Рисунок 31 – Страница управления подписками

Просмотреть личную информацию, список созданных мероприятий, подписок и подписчиков можно на странице профиля. Находясь на этой

странице пользователь может привязать аккаунт Telegram или начать создание нового мероприятия. Перейти на страницу профиля можно как по кнопке «Профиль» в навигационном меню, так и по нажатию на иконку в правом верхнем меню (рисунок 32).

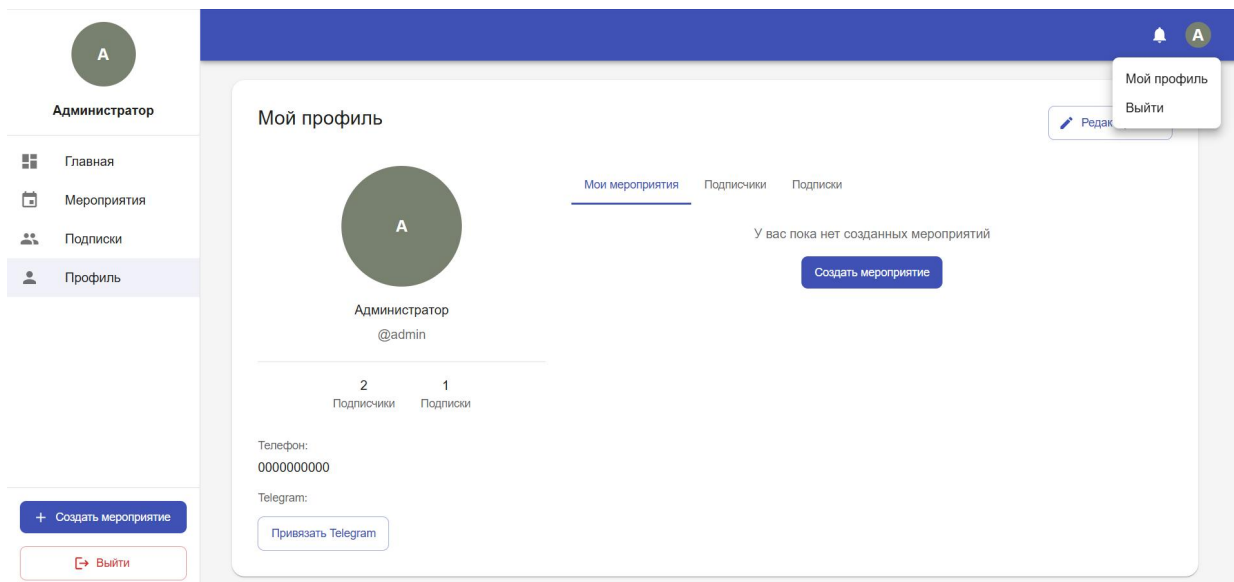


Рисунок 32 – Страница личного профиля

На странице профиля есть возможность изменить личные данные и загрузить фото профиля (рисунок 33).

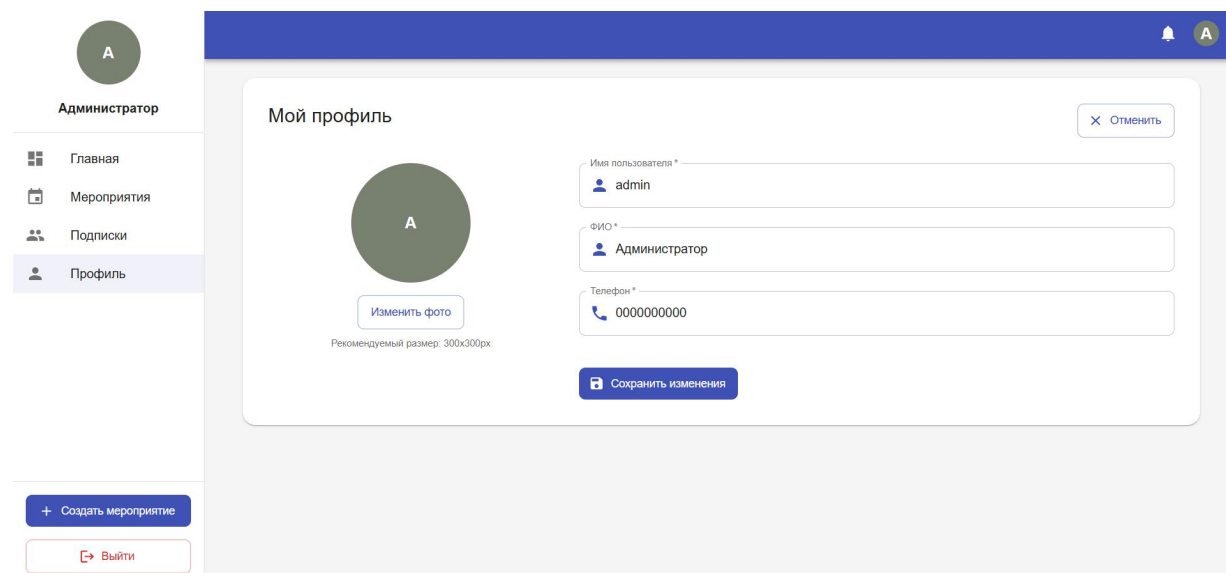


Рисунок 33 – Экран редактирования профиля

Нажав кнопку «Создать мероприятие» откроется страница ввода данных о мероприятии (рисунок 34).

Рисунок 34 – Страница создания мероприятия

Для ознакомления с подробной информацией о мероприятии пользователю нужно нажать кнопку «Подробнее» на иконке нужного мероприятия в списке. На этой странице находится кнопка принятия участия или отказа от участия в мероприятии (рисунок 35).

Рисунок 35 – Страница мероприятия

В нижней части страницы каждого мероприятия располагается раздел отзывов и комментариев (рисунок 36).

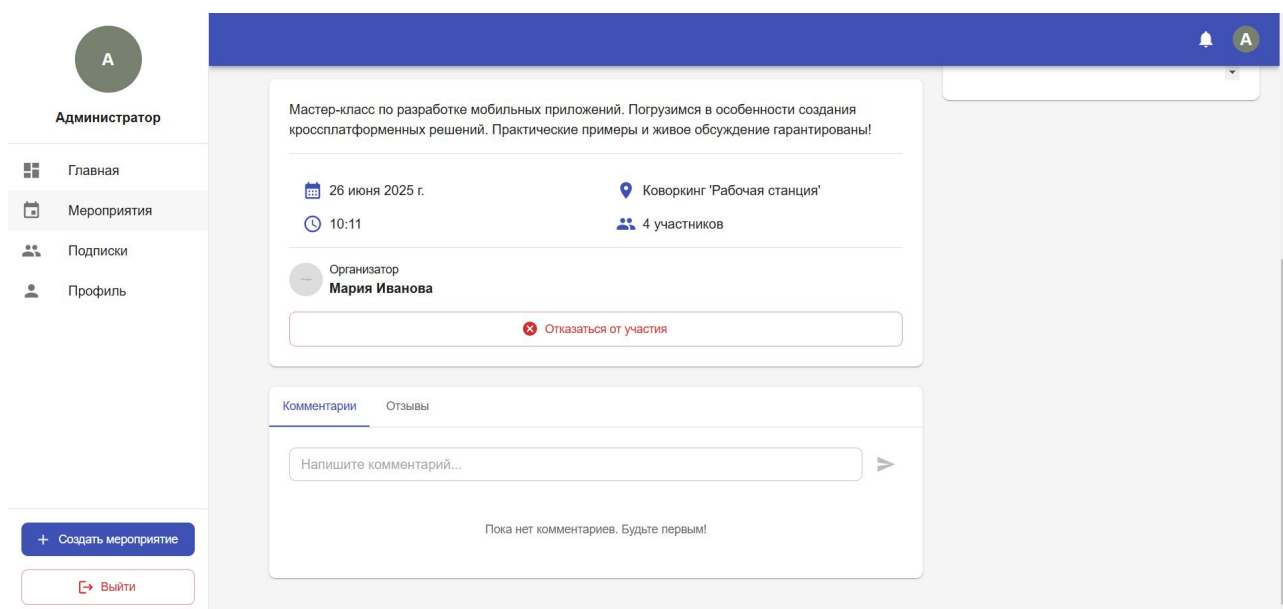


Рисунок 36 – Отзывы и комментарии

На текущем этапе разработки системы с использованием Next.js был создан понятный и простой пользовательский интерфейс, который отвечает требованиям ориентированности на удобство пользователя.

3.3 Реализация взаимодействия между компонентами системы

Фреймворк FastAPI работает совместно с сервером Uvicorn, выбор этого сервера обусловлен его производительностью, простотой интеграции и способностью обрабатывать множество входящих запросов. Uvicorn является реализацией веб-сервера ASGI (интерфейс асинхронного серверного шлюза) для Python и поддерживает HTTP/1.1 и WebSockets [19].

Благодаря данному набору технологий можно легко создавать веб-приложения. Для этого необходимо создать FastAPI и запустить его на сервере. Фрагмент кода создания FastAPI представлен в листинге 10.

Листинг 10 – Программный код создания FastAPI

```
app = FastAPI(
    title="Сервис организации мероприятий и встреч",
    description="API для управления мероприятиями, подписками и уведомлениями",
    version="1.0.0",
    docs_url="/api/docs",
    openapi_url="/api/openapi.json"
```

```
)
app.include_router(router)
```

В данном фрагменте происходит создание FastAPI и подключение маршрутизаторов для связывания нужных URL с нужным содержимым. Запуск приложения на сервере осуществляется в отдельном файле. Фрагмент кода запуска представлен на листинге 11.

Листинг 11 – Программный код запуска FastAPI на сервере

```
HOST = os.getenv("HOST", "0.0.0.0")
PORT = int(os.getenv("PORT", "8000"))

if __name__ == "__main__":
    uvicorn.run("app.main:app", host=HOST, port=PORT, reload=True)
```

Маршруты веб-страниц основаны на файловой системе. В каждом файле отдельной страницы помимо контента страницы представлены функции, вызываемые при взаимодействии с ее элементами. Эти функции вызывают методы из необходимых сервисов для отправки HTTP запросов на сервер. Например, код, отвечающий за управление формой регистрации, представлен ниже (листинг 12).

Листинг 12 – Программный код, реализующий форму регистрации

```
const renderStepActions = () => {
  return (
    <Box sx={{ display: 'flex', justifyContent: 'space-between',
mt: 3 }}>
      <Button
        disabled={activeStep === 0 || isSubmitting}
        onClick={handleBack}
        startIcon={<ArrowBack />}
      >
        Назад
      </Button>

      <Button
        variant="contained"
        onClick={activeStep === steps.length - 1 ? handleSubmit :
handleNext}
        endIcon={activeStep === steps.length - 1 ? null :
<ArrowForward />}
        type={activeStep === steps.length - 1 ? 'submit' :
'button'}
        disabled={isSubmitting}
      >
```

```

        >
        {activeStep === steps.length - 1
          ? (isSubmitting ? <CircularProgress size={24}
color="inherit" /> : 'Зарегистрироваться')
          : 'Далее'}
        </Button>
      </Box>
    );
  };
};

```

Здесь при завершении последнего шага регистрации выполняется определенная в этом же файле функция, которая обрабатывает событие отправки формы. В ней вызывается метод регистрации пользователя из сервиса authService. Метод регистрации представлен на листинге 13.

Листинг 13 – Программный код метода регистрации сервиса authService

```

const authService = {
  register: async (data: RegisterData): Promise<AuthResponse> => {
    try {
      const userData = {
        username: data.username,
        password: data.password,
        full_name: data.full_name,
        phone: data.phone,
      };

      const response = await
apiClient.post<AuthResponse>('/auth/register', userData);

      // Сохраняем токен в localStorage
      if (response.data.access_token) {
        localStorage.setItem('token', response.data.access_token);
      }

      return response.data;
    } catch (error) {
      console.error('Registration error:', error);
      throw error;
    }
  }
}

```

Данный метод создает объект userData, который содержит переданные из формы данные о пользователе, и отправляет POST-запрос на сервер по адресу «/auth/register» с данными userData. Далее происходит возврат данных ответа сервера. Если в процессе выполнения запроса возникает ошибка, то она

обрабатывается в блоке `catch`.

Для отправки POST-запроса используется `ApiClient`, экземпляр `axios` из одноименной библиотеки. `Axios` – это популярная библиотека, используемая для выполнения HTTP-запросов из веб-браузеров или сред `Node.js`. Она упрощает процесс отправки асинхронных HTTP-запросов к конечным точкам REST, обработку ответов и выполнение различных задач, связанных с сетью.

Построенный на основе собственного `XMLHttpRequest` в JavaScript и API выборки, `Axios` предлагает удобный API с такими функциями, как перехватчики, автоматическое преобразование данных в JSON, обработка ошибок и поддержка старых браузеров [20].

На стороне сервера запрос по данному маршруту примет роутер аутентификации. Код обработки POST запроса на серверной части представлен в листинге 14.

Листинг 14 – Программный код обработки POST запроса регистрации

```
router = APIRouter(
    prefix="/api/auth",
    tags=["auth"],
    responses={401: {"description": "Unauthorized"}},
)

@router.post("/register", response_model=UserDisplay,
status_code=status.HTTP_201_CREATED)
async def register(user_data: UserCreate, db: Session =
Depends(get_db)):
    """Register a new user."""
    return AuthService.register(db, user_data)
```

Для возврата ответа используется сервис на стороне сервера, который взаимодействует с базой данных. Таким же образом происходит взаимодействие всех остальных компонентов системы.

Одним из ключевых преимуществ использования `FastAPI` является его способность автоматически генерировать интерактивную документацию API с помощью `Swagger UI`. `Swagger UI` – это мощный инструмент, который позволяет визуализировать и взаимодействовать с API прямо в браузере. Этот инструмент создает интерактивную документацию на основе спецификации `OpenAPI`, что

делает процесс разработки и тестирования API более удобным и эффективным

С помощью Swagger UI можно легко просматривать доступные конечные точки API, их параметры запросов и ответов. Это позволяет быстро понять, как работает API, и упрощает процесс тестирования. Кроме того, Swagger UI позволяет отправлять тестовые запросы к API и получать ответы в реальном времени, что значительно ускоряет процесс отладки и разработки [21].

В качестве примера работы API приведен POST-запрос на регистрацию пользователя, выполненный в Swagger UI (рисунок 37).

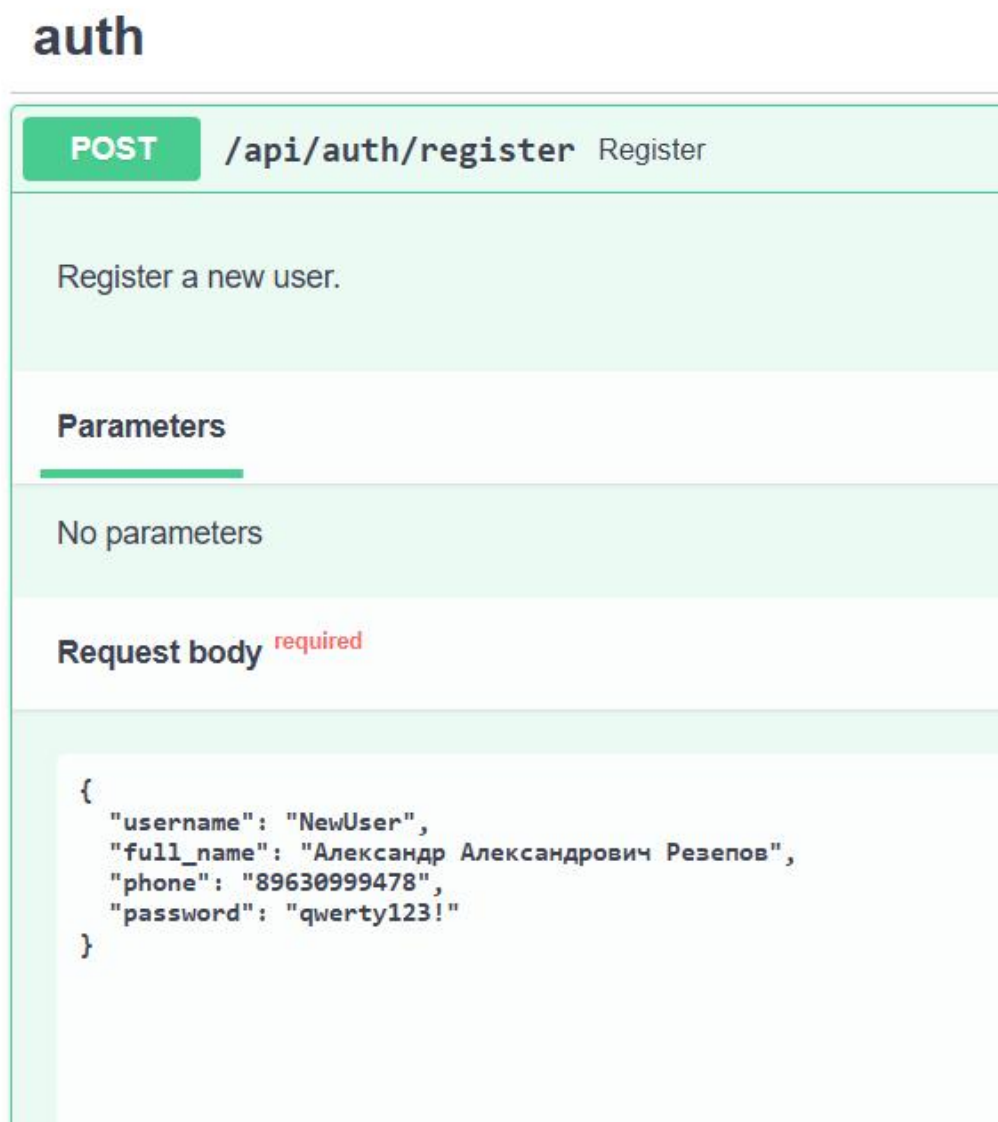


Рисунок 37 – POST-запрос: регистрация пользователя

Ответ на данный запрос представлен на рисунке 38.

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8000/api/auth/register' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "username": "NewUser",
    "full_name": "Александр Александрович Резепов",
    "phone": "89630999478",
    "password": "qwerty123!"
  }'
```

Request URL

```
http://localhost:8000/api/auth/register
```

Server response

Code	Details
201	<p>Response body</p> <pre>{ "username": "NewUser", "full_name": "Александр Александрович Резепов", "phone": "89630999478", "id": 10, "profile_picture": null, "created_at": "2025-06-09T14:39:29.540563Z", "telegram_chat_id": null }</pre> <p>Response headers</p> <pre>access-control-allow-credentials: true access-control-allow-origin: * content-length: 217 content-type: application/json date: Mon, 09 Jun 2025 14:39:28 GMT server: uvicorn</pre>

Responses

Рисунок 38 – Ответ на POST-запрос: регистрация пользователя

Таким образом, было организовано взаимодействие между компонентами серверной и клиентской части автоматизированной системы для организации мероприятий и встреч в сообществе. Благодаря фреймворку FastAPI для просмотра и тестирования HTTP-запросов была автоматически составлена документация по API в Swagger UI.

4 Тестирование и оценка качества кода

Для обеспечения высокого уровня надежности и качества разрабатываемой системы было принято решение провести комплексное тестирование. На данном этапе работы будет проведено функциональное тестирование методом комбинаторики и попарного тестирования, а также проведен анализ сложности кода по метрике Холстеда. В качестве нефункционального тестирования было выбрано тестирование совместимости.

4.1 Тестирование функционала регистрации методом комбинаторики и попарного тестирования

Комбинаторное тестирование представляет собой метод проверки программных продуктов, ориентированный на выявление дефектов, возникающих при различных сочетаниях входных параметров. Одной из самых эффективных стратегий является попарное тестирование.

Суть метода попарного тестирования заключается в предположении, согласно которому подавляющее большинство ошибок обусловлено взаимодействием одного или двух параметров системы. Методика предусматривает построение набора проверок для всех возможных сочетаний значений каждой пары параметров. Такой подход обеспечивает существенно более широкое покрытие пространства входных данных по сравнению с изолированным рассмотрением отдельных параметров.

Для реализации попарного тестирования применяются специальные алгоритмы, использующие ортогональные массивы либо принцип построения all-pairs-комбинаций. В основе этих алгоритмов лежат фундаментальные результаты теории комбинаторики и дискретной математики – например, свойства латинских квадратов [22].

В разработанной системе пользователю для прохождения регистрации необходимо пройти два этапа: ввод учетных данных и ввод личной информации.

На этапе ввода учетных данных задается имя, состоящее из букв, цифр и символов нижнего подчеркивания и содержащее от 3 до 30 символов, и пароль,

включающий в себя минимум 1 букву, 1 цифру и 1 специальный символ, а также содержащий от 6 до 30 символов.

На этапе заполнения личной информации необходимо ввести фамилию, имя и отчество: последовательность букв и символов пробела длиной не более 50 символов, а также номер телефона, начинающийся с российского телефонного кода и состоящий из 11 цифр.

Тестирование методом комбинаторики и попарного тестирования было проведено на примере прецедента регистрации в системе. Для процесса прохождения регистрации была составлена схема, описывающая последовательность выполнения данного прецедента (рисунок 39).

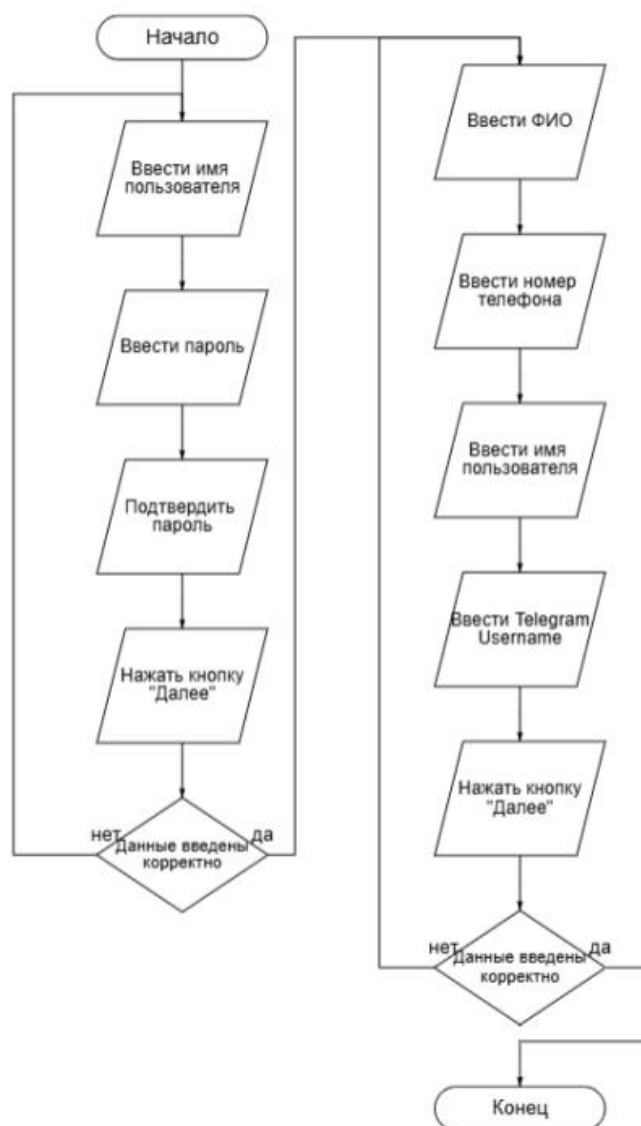


Рисунок 39 – Схема регистрации в системе

Для функционала регистрации в системе с помощью инструмента pairwise были составлены тест-кейсы для регистрационных полей. Тест-кейсы для этапа ввода учетных данных представлены на рисунке 40.

	Длина имени	Содержание имени пользователя	Длина пароля	Требования безопасности к паролю
1	0 - 2	Русские или английские буквы, цифры и символ " _ "	0 - 5	Минимум 1 буква, минимум 1 цифра, минимум 1 спец. символ
2	0 - 2	Остальное	6 - 30	Остальное
3	0 - 2	Русские или английские буквы, цифры и символ " _ "	31+	Минимум 1 буква, минимум 1 цифра, минимум 1 спец. символ
4	31+	Остальное	31+	Минимум 1 буква, минимум 1 цифра, минимум 1 спец. символ
5	31+	Русские или английские буквы, цифры и символ " _ "	0 - 5	Остальное
6	31+	Русские или английские буквы, цифры и символ " _ "	6 - 30	Минимум 1 буква, минимум 1 цифра, минимум 1 спец. символ
7	3 - 30	Русские или английские буквы, цифры и символ " _ "	6 - 30	Минимум 1 буква, минимум 1 цифра, минимум 1 спец. символ
8	3 - 30	Русские или английские буквы, цифры и символ " _ "	31+	Остальное
9	3 - 30	Остальное	0 - 5	Минимум 1 буква, минимум 1 цифра, минимум 1 спец. символ

Рисунок 40 – Тест-кейсы pairwise для учетных данных

Тест-кейсы для этапа заполнения полей личной информации представлены на рисунке 41.

	Длина имени	Содержание имени пользователя	Длина пароля	Требования безопасности к паролю
1	0 - 2	Русские или английские буквы, цифры и символ " _ "	0 - 5	Минимум 1 буква, минимум 1 цифра, минимум 1 спец. символ
2	0 - 2	Остальное	6 - 30	Остальное
3	0 - 2	Русские или английские буквы, цифры и символ " _ "	31+	Минимум 1 буква, минимум 1 цифра, минимум 1 спец. символ
4	31+	Остальное	31+	Минимум 1 буква, минимум 1 цифра, минимум 1 спец. символ
5	31+	Русские или английские буквы, цифры и символ " _ "	0 - 5	Остальное
6	31+	Русские или английские буквы, цифры и символ " _ "	6 - 30	Минимум 1 буква, минимум 1 цифра, минимум 1 спец. символ
7	3 - 30	Русские или английские буквы, цифры и символ " _ "	6 - 30	Минимум 1 буква, минимум 1 цифра, минимум 1 спец. символ
8	3 - 30	Русские или английские буквы, цифры и символ " _ "	31+	Остальное
9	3 - 30	Остальное	0 - 5	Минимум 1 буква, минимум 1 цифра, минимум 1 спец. символ

Рисунок 41 – Тест-кейсы pairwise для личной информации

Функционал регистрации пользователя был протестирован на основе полученных тест-кейсов (таблица Б.1).

По результатам организованного в соответствии с тест-кейсами, составленными в pairwise, тестирования функционала регистрации методом комбинаторики и попарного тестирования были получены успешные результаты.

4.2 Анализ сложности кода модуля регистрации

Для количественной оценки характеристик программного обеспечения используются метрики кода. Эти показатели представляют собой числовые значения, позволяющие анализировать различные критерии качества программных решений, например, архитектуру, сложность реализации, удобство поддержки.

Применение метрик программного кода значительно облегчает работу разработчиков и команд проектирования программного обеспечения, метрики служат инструментом контроля качества исходных текстов и позволяют выявлять проблемные участки, принимать решения по оптимизации и улучшению программного кода. Кроме того, по данным показателям происходит сравнение исходных программных текстов до и после проведения рефакторинга [23].

Одна из наиболее простых метрик – количество строк исходного текста (SLOC). Изначально она предназначалась для оценки трудоемкости проектов. Однако со временем из-за различий в синтаксисе современных языков программирования ее информативность снизилась. В связи с этим выделяют два типа SLOC: логические строки и физические строки. Логические строки кода – это количество операторов программы. Физические строки кода – это все непустые строки текстового файла [24].

Для расчета метрик был выбран фрагмент кода регистрации нового пользователя в системе (листинг 15).

Листинг 15 – Функция регистрации нового пользователя

```
def register(db: Session, user_data: UserCreate):
    # Check if username exists
    db_user = UserRepo.get_by_name(db, username)
    if db_user:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Имя пользователя уже занято"
        )

    # Check if phone exists
    db_user = UserRepository.get_by_phone(db, user_data.phone)
    if db_user:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Телефон уже зарегистрирован"
        )

    # Create user
    return UserRepository.create(db, user_data)
```

Расчет метрики SLOC на примере кода функции из листинга 15 представлен на таблице 4.

Таблица 4 – Расчет метрики SLOC для функции регистрации пользователя

Метрика	Результат
Физические строки кода	19
Логические строки кода	12
Строки комментариев	3
Пустые строки	2

Результат расчета метрики показывает, что код функции, представленный на листинге 15, не имеет излишней сложности, его просто анализировать и использовать.

Обычно размер программ оценивается по числу строк кода, при этом под строкой понимают отдельный оператор программы как основную смысловую единицу работы – именно эти элементы позволяют приблизительно судить о сложности разработки.

Тем не менее только оценка размера далеко не всегда дает однозначное представление о сложности системы. Она подходит для грубой классификации продуктов с заметной разницей в объеме кода. При минимальных отличиях по размеру на первый план выходят дополнительные факторы сложности архитектуры проекта. Поэтому подобные оценки позволяют дать лишь приблизительную категоризацию без анализа структуры каждой группы приложений. К числу метрических показателей размера относится также метрика Холстеда. Методика Холстеда состоит из четырех основных измеряемых величин [25]:

$n1$ – число уникальных операторов программы;

$n2$ – число уникальных операндов программы;

$N1$ – общее количество операторов программы;

$N2$ – общее количество операндов программы.

В таблице 5 приведены результаты подсчета количества операторов и операндов для тестируемого блока кода.

Таблица 5 – Операнды и операторы

№ п/п	Операторы	Число операторов	Операнды	Число операндов
1	def	1	db	4
2	:	5	Session	1
3	=	6	user_data	4
4	(	6	UserCreate	1
5	)	6	db_user	4
6	if	2	UserRepository	3
7	raise	2	username	1
8	,	6	HTTPException	2
9	.	7	status_code	2
10	“	4	status	2
11	return	1	HTTP_400_BAD_REQUEST	2
12	register	1	detail	2
13	get_by_username	1	“Имя пользователя уже занято”	1
14	get_by_phone	1	“Телефон уже зарегистрирован”	1
15	create	1		

Произведем расчет метрики Холстеда на примере кода из листинга 15.

Длина кода программы рассчитывается по формуле:

$$N = N1 + N2 = 50 + 31 = 81,$$

где $N1$ – это общее число операторов в программе;

$N2$ – общее число операндов в программе.

Для расчета словаря нужно сложить количество уникальных операторов программы с количеством уникальных операндов программы:

$$n = n1 + n2 = 15 + 14 = 29,$$

где $n1$ – число операторов;

$n2$ – число операндов.

Потенциальный объем программы и объем кода программы вычисляется следующим образом:

$$N = n1 * \log_2(n1) + n2 * \log_2(n2) = 111,9,$$

$$V = N * \log_2(n) = 393,5,$$

где N – длина программы;

n – словарь программы.

Сложность кода программы рассчитывается следующим образом:

$$D = (n1 / 2) * (N2 / n2) = 16,6.$$

Усилие для написания и понимания кода рассчитывается по формуле:

$$E = D * V = 6532,1.$$

Время в секундах для написания кода рассчитывается по формуле:

$$T = E / 18 = 362,9.$$

Количество доставленных кодом ошибок рассчитывается по формуле:

$$B = V / 3000 = 0,13.$$

В результате анализа сложности кода с использованием метрики Холстеда было установлено, что программный код данной функции соответствует необходимым требованиям по метрикам качества. Соответствие этим требованиям позволяет считать код успешной реализацией программы.

4.3 Тестирование совместимости системы

Тестирование на совместимость – это вид тестирования, направленный на подтверждение корректности работы программных продуктов при работе с различными видами аппаратного обеспечения, операционными системами,

браузерами. Основная цель такого тестирования – своевременное выявление возможных сбоев или некорректной работы при эксплуатации программного обеспечения в различных технических и программных условиях.

Особое значение данное тестирование приобретает для программных решений, рассчитанных на широкую аудиторию пользователей. Его проведение позволяет убедиться в том, что доступ к функциональности системы будет обеспечен для всех конечных пользователей независимо от используемых устройств или платформ. Тесты на совместимость способствуют раннему обнаружению дефектов, которые могут негативно сказаться на стабильности работы приложения после внедрения либо привести к снижению производительности. Кроме того, результаты тестирования на совместимость дают понять, нуждается ли интерфейс в доработке или повышении удобства.

Поскольку взаимодействие пользователя осуществляется через веб-страницу, актуальной становится проверка корректности отображения и функционирования интерфейса во множестве браузеров. В таблице 6 приведены результаты тестирования совместимости автоматизированной системы организации мероприятий и встреч в сообществе.

Таблица 6 – Операнды и операторы

Устройство/Браузер	Сценарий	Результат	Тест пройден/не пройден
Windows 10, Yandex 25.4	Зарегистрироваться в системе, создать мероприятие	Регистрация прошла успешно, мероприятие создано, компоненты интерфейса отображаются корректно	Тест пройден
Windows 10, Google Chrome 137.0	Зарегистрироваться в системе, принять участие в мероприятии	Регистрация прошла успешно, пользователь добавлен в список участников, компоненты интерфейса отображаются корректно	Тест пройден
Windows 10, Microsoft Edge 137.0	Зарегистрироваться в системе, подписаться на пользователя, просмотреть его мероприятие в ленте	Регистрация прошла успешно, подписка осуществлена, мероприятие отобразилось в ленте, компоненты интерфейса отображаются корректно	Тест пройден

Продолжение таблицы 6

Устройство/Браузер	Сценарий	Результат	Тест пройден/не пройден
Linux 6.14, Vivaldi 7.4	Зарегистрироваться в системе, загрузить фото профиля	Регистрация прошла успешно, фотография успешно загружена, компоненты интерфейса отображаются корректно	Тест пройден
macOS 15.5 Sequoia, Safari 18	Зарегистрироваться в системе, создать мероприятие, редактировать мероприятие	Регистрация прошла успешно, мероприятие создано, изменения успешно применены, компоненты интерфейса отображаются корректно	Тест пройден
Android 15, Chrome 126	Зарегистрироваться в системе, найти мероприятие, оставить комментарий	Регистрация прошла успешно, мероприятие отобразилось в поиске, комментарий успешно оставлен, компоненты интерфейса немного съезжают	Тест пройден

В результате проведенного тестирования совместимости системы было установлено, что система успешно интегрируется с существующими браузерами. Все ключевые функции работают без сбоев и обеспечивают высокий уровень пользовательского опыта.

Заключение

В результате выполнения выпускной квалификационной работы была создана автоматизированная система для организации мероприятий и встреч в сообществе. Для этого выполнены следующие этапы:

- проведено исследование предметной области вместе с анализом существующих аналогичных программных продуктов, сформулированы задачи разработки согласно выявленным функциональным и нефункциональным требованиям;
- выполнено архитектурное проектирование системы, определен набор технологического стека, спроектированы макеты пользовательского интерфейса;
- спланированы задачи проекта, осуществлено распределение ресурсов разработки и оценка бюджета;
- выполнена реализация автоматизированной системы для организации мероприятий и встреч в сообществе, включающая в себя разработку серверной части, клиентской части и осуществление взаимодействия между ними;
- проведено попарное тестирование и тестирование совместимости, по результатам которых был сделан вывод о корректности работы системы, а также выполнена оценка качества кода разработанной системы, показавшее соответствие программного кода критериям качества.

Таким образом, была достигнута цель и выполнены все задачи, выдвинутые на этапе анализа. Система, разработанная в рамках выпускной квалификационной работы, отвечает всем необходимым требованиям, а тестирование подтвердило правильность ее работы.

Список использованных источников

1. Zoho Backstage Reviews [Электронный ресурс]. URL: <https://www.softwareadvice.com/website-builder/zoho-backstage-profile/reviews/> (дата обращения: 19.05.2025).
2. Приложение для мероприятий: обзор новых возможностей Event Rocks [Электронный ресурс]. URL: <https://blog.eventrocks.ru/obzor-novykh-vozmozhnostiei-event-rocks/> (дата обращения: 19.05.2025).
3. Cvent Event Management Software In-Depth Review [Электронный ресурс]. URL: <https://thehotelgm.com/tools/cvent-review/> (дата обращения: 19.05.2025).
4. Современные информационные технологии в проектировании программных систем и комплексов: учеб. пособие / И. Ю. Балашова, Д. В. Такташкин; под ред. П. П. Макарычева. – Пенза : Изд-во ПГУ, 2019. – 106 с.
5. Разработка требований к программному обеспечению. 3-е изд., дополненное / К. Вигерс, Д. Битти; Пер. с англ. – М. : Издательство «Русская редакция» ; СПб. : БХВ-Петербург, 2014. – 736 с.
6. Next.js – обзор, руководство, архитектура, примеры, плюсы и минусы [Электронный ресурс]. URL: <https://www.sravni.ru/kursy/info/next-js/> (дата обращения: 21.05.2025).
7. 15 лучших UI-библиотек React 2025: сравнение и статистика использования [Электронный ресурс]. URL: <https://proglib.io/p/top-15-ui-bibliotek-dlya-react-v-2025-godu-2025-02-24/> (дата обращения: 22.05.2025).
8. Плюсы и минусы FastAPI [Электронный ресурс]. URL: <https://habr.com/ru/articles/748552/> (дата обращения: 23.05.2025).
9. СУБД PostgreSQL: принцип работы, преимущества и недостатки [Электронный ресурс]. URL: <https://serverflow.ru/blog/stati/subd-postgresql-printsip-raboty-preimushchestva-i-nedostatki/> (дата обращения: 23.05.2025).
10. Benefit of using MVC [Электронный ресурс]. URL: <https://www.geeksforgeeks.org/benefit-of-using-mvc/> (дата обращения: 24.05.2025).

11. Next.js. Технология современной веб-разработки [Электронный ресурс]. URL: <https://habr.com/ru/companies/auriga/articles/786912/> (дата обращения: 25.05.2025).
12. Pydantic2: Полное руководство для Python-разработчиков – от основ до продвинутых техник [Электронный ресурс]. URL: <https://habr.com/ru/companies/amvera/articles/851642/> (дата обращения: 26.05.2025).
13. UML: обзор основных типов диаграмм, диаграмма компонентов. Часть 2 [Электронный ресурс]. URL: <https://habr.com/ru/articles/756552/> (дата обращения: 27.05.2025).
14. Основы проектирования интерфейсов для системных аналитиков: ключевые принципы и подходы [Электронный ресурс]. URL: <https://habr.com/ru/articles/872168/> (дата обращения: 27.05.2025).
15. Чем занимается менеджер проекта в IT и как им стать [Электронный ресурс]. URL: <https://education.yandex.ru/journal/chem-zanimaetsya-menedzher-proektov-v-it-i-kak-im-stat> (дата обращения: 27.05.2025).
16. Диаграмма Ганта – инструкция по созданию, шаблоны и примеры [Электронный ресурс]. URL: <https://getcompass.ru/blog/posts/diagramma-ganta> (дата обращения: 28.05.2025).
17. Создание собственного API на Python (FastAPI) [Электронный ресурс]. URL: <https://habr.com/ru/companies/amvera/articles/826196/> (дата обращения: 28.05.2025).
18. Диаграмма классов (Class diagram) [Электронный ресурс]. URL: <https://docs.ensi.tech/analyst-guides/tools/diagrams/uml/class-diagram> (дата обращения: 29.05.2025).
19. FastAPI Uvicorn [Электронный ресурс]. URL: <https://www.geeksforgeeks.org/fastapi-uvicorn/> (дата обращения: 29.05.2025).
20. What is Axios? [Электронный ресурс]. URL: <https://www.geeksforgeeks.org/html/what-is-axios/> (дата обращения: 29.05.2025).
21. Использование Swagger UI в FastAPI для документации и

тестирования API [Электронный ресурс]. URL: http://www.monogenes.ru/python/fast_api/fast_api_04_swagger/ (дата обращения: 29.05.2025).

22. Миф о комбинаторике в тестировании [Электронный ресурс]. URL: <https://habr.com/ru/companies/hh/articles/736870/> (дата обращения: 30.05.2025).

23. Метрики кода. Цикломатическая сложность [Электронный ресурс]. URL: <https://marchelos.github.io/posts/cyclomatic-complexity/> (дата обращения: 30.05.2025).

24. Программный код и его метрики [Электронный ресурс]. URL: <https://habr.com/ru/companies/intel/articles/106082/> (дата обращения: 30.05.2025).

25. Что делает код трудным для чтения? Визуальные паттерны сложности [Электронный ресурс]. URL: <https://habr.com/ru/articles/893820/> (дата обращения: 31.05.2025).

Приложение А

(обязательное)

Исходный код программного обеспечения

Листинг А.1 – Файл event_repository.py

```

from sqlalchemy.orm import Session
from sqlalchemy import func, desc
from datetime import datetime, timedelta
from typing import List, Optional, Dict, Any

from ..models import Event, EventImage, EventParticipant, Invitation, User,
Subscription
from ..schemas import EventCreate, EventUpdate

class EventRepository:
    @staticmethod
    def create(db: Session, event_data: EventCreate, creator_id: int):
        """Create a new event."""
        db_event = Event(
            title=event_data.title,
            event_date=event_data.event_date,
            location=event_data.location,
            description=event_data.description,
            creator_id=creator_id
        )
        db.add(db_event)
        db.commit()
        db.refresh(db_event)
        return db_event

    @staticmethod
    def add_image(db: Session, event_id: int, image_path: str):
        """Add an image to an event."""
        db_image = EventImage(
            image_path=image_path,
            event_id=event_id
        )
        db.add(db_image)
        db.commit()
        db.refresh(db_image)
        return db_image

    @staticmethod
    def get_by_id(db: Session, event_id: int):
        """Get event by ID."""
        event = db.query(Event).filter(Event.id == event_id).first()
        return event

    @staticmethod
    def update(db: Session, event_id: int, event_data: EventUpdate):
        """Update event."""
        db_event = db.query(Event).filter(Event.id == event_id).first()
        if not db_event:
            return None

        # Only update if event has not passed
        now = datetime.now()
        if db_event.event_date < now:
            return None

        # Handle existing_images if provided
        existing_images = event_data.existing_images
        if existing_images is not None:
            # Get current image paths for this event
            current_images = [img.image_path for img in db_event.images]

```

```

        # Find images to delete (images that were in current_images but not
        in existing_images)
        images_to_delete = [img for img in current_images if img not in
        existing_images]

        # Delete images that are no longer in the existing_images list
        if images_to_delete:
            for image_path in images_to_delete:
                db_image = db.query(EventImage).filter(
                    EventImage.event_id == event_id,
                    EventImage.image_path == image_path
                ).first()
                if db_image:
                    db.delete(db_image)

        # Remove existing_images from update_data as it's not a direct event
attribute
        update_data = event_data.dict(exclude_unset=True)
        if 'existing_images' in update_data:
            del update_data['existing_images']

        # Update remaining fields
        for key, value in update_data.items():
            setattr(db_event, key, value)

        db.commit()
        db.refresh(db_event)
        return db_event

    @staticmethod
    def delete(db: Session, event_id: int):
        """Delete event."""
        db_event = db.query(Event).filter(Event.id == event_id).first()
        if db_event:
            db.delete(db_event)
            db.commit()
            return True
        return False

    @staticmethod
    def delete_image(db: Session, image_id: int):
        """Delete an event image."""
        db_image = db.query(EventImage).filter(EventImage.id == image_id).first()
        if db_image:
            db.delete(db_image)
            db.commit()
            return True
        return False

    @staticmethod
    def get_all(db: Session, skip: int = 0, limit: int = 100, upcoming_only:
bool = False) -> List[Dict[str, Any]]:
        """Get all events."""
        query = db.query(Event)

        if upcoming_only:
            now = datetime.now()
            query = query.filter(Event.event_date >= now)

        events =
query.order_by(desc(Event.created_at)).offset(skip).limit(limit).all()

        # Convert events to dictionaries with serialized image paths
        return [event.to_dict() for event in events]

```

```

    @staticmethod
    def get_events_by_creator(db: Session, creator_id: int, skip: int = 0, limit:
int = 100) -> List[Event]:
        """Get events by creator ID."""
        events = db.query(Event).filter(
            Event.creator_id == creator_id
        ).order_by(desc(Event.created_at)).offset(skip).limit(limit).all()
        return events

    @staticmethod
    def get_upcoming_events_by_creator(db: Session, creator_id: int, skip: int =
0, limit: int = 100) -> List[Event]:
        """Get upcoming events by creator ID."""
        now = datetime.now()
        events = db.query(Event).filter(
            Event.creator_id == creator_id,
            Event.event_date >= now
        ).order_by(Event.event_date).offset(skip).limit(limit).all()
        return events

    @staticmethod
    def get_user_feed(db: Session, user_id: int, skip: int = 0, limit: int = 100)
-> List[Event]:
        """Get events from users the current user is following."""
        # Get IDs of users the current user is following
        following_ids = db.query(Subscription.followed_id).filter(
            Subscription.follower_id == user_id
        ).all()
        following_ids = [followed_id for (followed_id,) in following_ids]

        if not following_ids:
            return []

        events = db.query(Event).filter(
            Event.creator_id.in_(following_ids)
        ).order_by(desc(Event.created_at)).offset(skip).limit(limit).all()
        return events

    @staticmethod
    def search(db: Session, query: str, skip: int = 0, limit: int = 100,
upcoming_only: bool = False) -> List[Event]:
        """Search events by title or location."""
        search_query = f"%{query}%"
        base_query = db.query(Event).filter(
            (Event.title.ilike(search_query)) |
            (Event.location.ilike(search_query)) |
            (Event.description.ilike(search_query))
        )

        if upcoming_only:
            now = datetime.now()
            base_query = base_query.filter(Event.event_date >= now)

        events =
base_query.order_by(desc(Event.created_at)).offset(skip).limit(limit).all()
        return events

    @staticmethod
    def get_participants_count(db: Session, event_id: int):
        """Get the number of participants for an event."""
        return db.query(func.count(EventParticipant.id)).filter(
            EventParticipant.event_id == event_id
        ).scalar()

```

```

    @staticmethod
    def is_user_participant(db: Session, event_id: int, user_id: int):
        """Check if a user is a participant of an event."""
        participant = db.query(EventParticipant).filter(
            EventParticipant.event_id == event_id,
            EventParticipant.user_id == user_id
        ).first()
        return participant is not None

    @staticmethod
    def is_user_invited(db: Session, event_id: int, user_id: int):
        """Check if a user is invited to an event."""
        invitation = db.query(Invitation).filter(
            Invitation.event_id == event_id,
            Invitation.user_id == user_id
        ).first()
        return invitation is not None

    @staticmethod
    def get_events_by_date(db: Session, date: datetime, skip: int = 0, limit:
int = 100) -> List[Event]:
        """Get events occurring on a specific date."""
        # Get events where the date part matches
        events = db.query(Event).filter(
            func.date(Event.event_date) == func.date(date)
        ).order_by(Event.event_date).offset(skip).limit(limit).all()
        return events

    @staticmethod
    def get_events_in_next_24_hours(db: Session) -> List[Event]:
        """Get events occurring in the next 24 hours."""
        now = datetime.now()
        future = now + timedelta(days=1)
        events = db.query(Event).filter(
            Event.event_date >= now,
            Event.event_date <= future
        ).all()
        return events

```

Листинг А.2 – Файл `interaction_repository.py`

```

from sqlalchemy.orm import Session
from sqlalchemy import func, desc
from typing import List, Optional
from datetime import datetime

from ..models import Comment, Review, Event
from ..schemas import CommentCreate, CommentUpdate, ReviewCreate, ReviewUpdate

class CommentRepository:
    @staticmethod
    def create(db: Session, comment_data: CommentCreate, event_id: int, user_id:
int):
        """Create a new comment."""
        db_comment = Comment(
            text=comment_data.text,
            event_id=event_id,
            user_id=user_id
        )
        db.add(db_comment)
        db.commit()
        db.refresh(db_comment)
        return db_comment

```

```

    @staticmethod
    def get_by_id(db: Session, comment_id: int):
        """Get comment by ID."""
        return db.query(Comment).filter(Comment.id == comment_id).first()

    @staticmethod
    def update(db: Session, comment_id: int, comment_data: CommentUpdate,
user_id: int):
        """Update comment."""
        db_comment = db.query(Comment).filter(
            Comment.id == comment_id,
            Comment.user_id == user_id
        ).first()

        if not db_comment:
            return None

        # Update fields
        update_data = comment_data.dict(exclude_unset=True)
        for key, value in update_data.items():
            setattr(db_comment, key, value)

        db.commit()
        db.refresh(db_comment)
        return db_comment

    @staticmethod
    def delete(db: Session, comment_id: int, user_id: int):
        """Delete comment."""
        db_comment = db.query(Comment).filter(
            Comment.id == comment_id,
            Comment.user_id == user_id
        ).first()

        if db_comment:
            db.delete(db_comment)
            db.commit()
            return True
        return False

    @staticmethod
    def get_by_event(db: Session, event_id: int, skip: int = 0, limit: int =
100):
        """Get comments for an event."""
        return db.query(Comment).filter(
            Comment.event_id == event_id
        ).order_by(desc(Comment.created_at)).offset(skip).limit(limit).all()

class ReviewRepository:
    @staticmethod
    def create(db: Session, event_id: int, review_data: ReviewCreate, user_id:
int):
        """Create a new review."""
        # Check if the event has already passed
        event = db.query(Event).filter(Event.id == event_id).first()
        if not event:
            return None

        # Compare with Python datetime objects instead of using func.now()
        current_time = datetime.now()
        if event.event_date > current_time:
            return None

```

```

# Check if user has already reviewed this event
existing_review = db.query(Review).filter(
    Review.event_id == event_id,
    Review.user_id == user_id
).first()

if existing_review:
    return None

db_review = Review(
    text=review_data.text,
    rating=review_data.rating,
    event_id=event_id,
    user_id=user_id
)
db.add(db_review)
db.commit()
db.refresh(db_review)
return db_review

@staticmethod
def get_by_id(db: Session, review_id: int):
    """Get review by ID."""
    return db.query(Review).filter(Review.id == review_id).first()

@staticmethod
def update(db: Session, review_id: int, review_data: ReviewUpdate, user_id:
int):
    """Update review."""
    db_review = db.query(Review).filter(
        Review.id == review_id,
        Review.user_id == user_id
    ).first()

    if not db_review:
        return None

    # Update fields
    update_data = review_data.dict(exclude_unset=True)
    for key, value in update_data.items():
        setattr(db_review, key, value)

    db.commit()
    db.refresh(db_review)
    return db_review

@staticmethod
def delete(db: Session, review_id: int, user_id: int):
    """Delete review."""
    db_review = db.query(Review).filter(
        Review.id == review_id,
        Review.user_id == user_id
    ).first()

    if db_review:
        db.delete(db_review)
        db.commit()
        return True
    return False

@staticmethod
def get_by_event(db: Session, event_id: int, skip: int = 0, limit: int =
100):
    """Get reviews for an event."""

```

```

        return db.query(Review).filter(
            Review.event_id == event_id
        ).order_by(desc(Review.created_at)).offset(skip).limit(limit).all()

    @staticmethod
    def get_average_rating(db: Session, event_id: int):
        """Get average rating for an event."""
        result = db.query(func.avg(Review.rating)).filter(
            Review.event_id == event_id
        ).scalar()
        return result if result else 0.0

```

Листинг А.3 – Файл participation_repository.py

```

from sqlalchemy.orm import Session
from sqlalchemy import func
from typing import List, Optional

from ..models import EventParticipant, Invitation, User, Subscription, Event

class ParticipationRepository:
    @staticmethod
    def add_participant(db: Session, event_id: int, user_id: int):
        """Add a participant to an event."""
        # Check if the user is already a participant
        existing = db.query(EventParticipant).filter(
            EventParticipant.event_id == event_id,
            EventParticipant.user_id == user_id
        ).first()

        if existing:
            return existing

        # Add the participant
        db_participant = EventParticipant(
            event_id=event_id,
            user_id=user_id
        )
        db.add(db_participant)
        db.commit()
        db.refresh(db_participant)
        return db_participant

    @staticmethod
    def remove_participant(db: Session, event_id: int, user_id: int):
        """Remove a participant from an event."""
        db_participant = db.query(EventParticipant).filter(
            EventParticipant.event_id == event_id,
            EventParticipant.user_id == user_id
        ).first()

        if db_participant:
            db.delete(db_participant)
            db.commit()
            return True
        return False

    @staticmethod
    def get_event_participants(db: Session, event_id: int, skip: int = 0, limit:
int = 100):
        """Get all participants for an event."""
        return db.query(EventParticipant).filter(
            EventParticipant.event_id == event_id
        ).offset(skip).limit(limit).all()

```

```

    @staticmethod
    def get_user_participations(db: Session, user_id: int, skip: int = 0, limit:
int = 100):
        """Get all events a user is participating in."""
        return db.query(EventParticipant).filter(
            EventParticipant.user_id == user_id
        ).offset(skip).limit(limit).all()

class InvitationRepository:
    @staticmethod
    def create_invitation(db: Session, event_id: int, user_id: int):
        """Create an invitation for a user to an event."""
        # Check if the user is already invited
        existing = db.query(Invitation).filter(
            Invitation.event_id == event_id,
            Invitation.user_id == user_id
        ).first()

        if existing:
            return existing

        # Create the invitation
        db_invitation = Invitation(
            event_id=event_id,
            user_id=user_id
        )
        db.add(db_invitation)
        db.commit()
        db.refresh(db_invitation)
        return db_invitation

    @staticmethod
    def delete_invitation(db: Session, invitation_id: int):
        """Delete an invitation."""
        db_invitation = db.query(Invitation).filter(Invitation.id ==
invitation_id).first()
        if db_invitation:
            db.delete(db_invitation)
            db.commit()
            return True
        return False

    @staticmethod
    def get_by_id(db: Session, invitation_id: int):
        """Get invitation by ID."""
        return db.query(Invitation).filter(Invitation.id ==
invitation_id).first()

    @staticmethod
    def get_event_invitations(db: Session, event_id: int, skip: int = 0, limit:
int = 100):
        """Get all invitations for an event."""
        return db.query(Invitation).filter(
            Invitation.event_id == event_id
        ).offset(skip).limit(limit).all()

    @staticmethod
    def get_user_invitations(db: Session, user_id: int, skip: int = 0, limit:
int = 100):
        """Get all invitations for a user."""
        return db.query(Invitation).filter(
            Invitation.user_id == user_id
        ).offset(skip).limit(limit).all()

```

```

    @staticmethod
    def create_invitations_for_followers(db: Session, event_id: int, creator_id:
int):
        """Create invitations for all followers of the event creator."""
        # Get all followers of the creator
        followers = db.query(Subscription).filter(
            Subscription.followed_id == creator_id
        ).all()

        # Create invitations for each follower
        invitations = []
        for follower in followers:
            invitation = InvitationRepository.create_invitation(db, event_id,
follower.follower_id)
            invitations.append(invitation)

        return invitations

```

Листинг А.4 – Файл subscription_repository.py

```

from sqlalchemy.orm import Session
from sqlalchemy import func
from typing import List, Optional

from ..models import Subscription, User

class SubscriptionRepository:
    @staticmethod
    def follow_user(db: Session, follower_id: int, followed_id: int):
        """Create a subscription (follow a user)."""
        # Check if already following
        existing = db.query(Subscription).filter(
            Subscription.follower_id == follower_id,
            Subscription.followed_id == followed_id
        ).first()

        if existing:
            return existing

        # Can't follow yourself
        if follower_id == followed_id:
            return None

        # Create new subscription
        db_subscription = Subscription(
            follower_id=follower_id,
            followed_id=followed_id
        )
        db.add(db_subscription)
        db.commit()
        db.refresh(db_subscription)
        return db_subscription

    @staticmethod
    def unfollow_user(db: Session, follower_id: int, followed_id: int):
        """Delete a subscription (unfollow a user)."""
        db_subscription = db.query(Subscription).filter(
            Subscription.follower_id == follower_id,
            Subscription.followed_id == followed_id
        ).first()

        if db_subscription:
            db.delete(db_subscription)

```

```

        db.commit()
        return True
    return False

    @staticmethod
    def get_followers(db: Session, user_id: int, skip: int = 0, limit: int =
100):
        """Get all followers of a user."""
        followers = db.query(Subscription).filter(
            Subscription.followed_id == user_id
        ).offset(skip).limit(limit).all()
        return followers

    @staticmethod
    def get_following(db: Session, user_id: int, skip: int = 0, limit: int =
100):
        """Get all users a user is following."""
        following = db.query(Subscription).filter(
            Subscription.follower_id == user_id
        ).offset(skip).limit(limit).all()
        return following

    @staticmethod
    def is_following(db: Session, follower_id: int, followed_id: int):
        """Check if a user is following another user."""
        existing = db.query(Subscription).filter(
            Subscription.follower_id == follower_id,
            Subscription.followed_id == followed_id
        ).first()
        return existing is not None

    @staticmethod
    def get_followers_count(db: Session, user_id: int):
        """Get the number of followers of a user."""
        return db.query(func.count(Subscription.id)).filter(
            Subscription.followed_id == user_id
        ).scalar()

    @staticmethod
    def get_following_count(db: Session, user_id: int):
        """Get the number of users a user is following."""
        return db.query(func.count(Subscription.id)).filter(
            Subscription.follower_id == user_id
        ).scalar()

```

Листинг А.5 – Файл user_repository.py

```

from sqlalchemy.orm import Session
from sqlalchemy import func
from ..models import User, Event, Subscription
from ..schemas import UserCreate, UserUpdate
from ..utils.security import get_password_hash

class UserRepository:
    @staticmethod
    def create(db: Session, user_data: UserCreate):
        """Create a new user."""
        # Hash the password
        hashed_password = get_password_hash(user_data.password)
        db_user = User(
            username=user_data.username,
            password=hashed_password,
            full_name=user_data.full_name,
            phone=user_data.phone

```

```

    )
    db.add(db_user)
    db.commit()
    db.refresh(db_user)
    return db_user

    @staticmethod
    def get_by_id(db: Session, user_id: int):
        """Get user by ID."""
        return db.query(User).filter(User.id == user_id).first()

    @staticmethod
    def get_by_username(db: Session, username: str):
        """Get user by username."""
        return db.query(User).filter(User.username == username).first()

    @staticmethod
    def get_by_phone(db: Session, phone: str):
        """Get user by phone."""
        return db.query(User).filter(User.phone == phone).first()

    @staticmethod
    def update(db: Session, user_id: int, user_data: UserUpdate):
        """Update user."""
        db_user = db.query(User).filter(User.id == user_id).first()
        if not db_user:
            return None

        # Update fields if provided
        update_data = user_data.dict(exclude_unset=True)

        # Hash password if it's being updated
        if "password" in update_data and update_data["password"]:
            update_data["password"] = get_password_hash(update_data["password"])

        for key, value in update_data.items():
            setattr(db_user, key, value)

        db.commit()
        db.refresh(db_user)
        return db_user

    @staticmethod
    def get_all(db: Session, skip: int = 0, limit: int = 100):
        """Get all users."""
        return db.query(User).offset(skip).limit(limit).all()

    @staticmethod
    def search(db: Session, query: str, skip: int = 0, limit: int = 100):
        """Search users by username or phone."""
        search_query = f"%{query}%"
        return db.query(User).filter(
            (User.username.ilike(search_query)) |
            (User.phone.ilike(search_query)) |
            (User.full_name.ilike(search_query))
        ).offset(skip).limit(limit).all()

    @staticmethod
    def get_user_stats(db: Session, user_id: int):
        """Get user statistics: event count, follower count, following count."""
        events_count = db.query(func.count(Event.id)).filter(Event.creator_id ==
user_id).scalar()

```

```

        followers_count =
db.query(func.count(Subscription.id)).filter(Subscription.followed_id ==
user_id).scalar()
        following_count =
db.query(func.count(Subscription.id)).filter(Subscription.follower_id ==
user_id).scalar()

    return {
        "events_count": events_count,
        "followers_count": followers_count,
        "following_count": following_count
    }

    @staticmethod
    def delete(db: Session, user_id: int):
        """Delete user."""
        db_user = db.query(User).filter(User.id == user_id).first()
        if db_user:
            db.delete(db_user)
            db.commit()
            return True
        return False

```

Листинг А.6 – Файл auth_service.py

```

from sqlalchemy.orm import Session
from fastapi import HTTPException, status
from datetime import datetime, timedelta

from ..utils.security import verify_password, create_access_token
from ..repositories import UserRepository
from ..schemas import UserCreate, UserLogin, Token

class AuthService:
    @staticmethod
    def register(db: Session, user_data: UserCreate):
        """Register a new user."""
        # Check if username exists
        db_user = UserRepository.get_by_username(db, user_data.username)
        if db_user:
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail="Имя пользователя уже занято"
            )

        # Check if phone exists
        db_user = UserRepository.get_by_phone(db, user_data.phone)
        if db_user:
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail="Телефон уже зарегистрирован"
            )

        # Create user
        return UserRepository.create(db, user_data)

    @staticmethod
    def login(db: Session, user_data: UserLogin):
        """Login a user."""
        # Get user by username
        user = UserRepository.get_by_username(db, user_data.username)

        # Check if user exists and password is correct
        if not user or not verify_password(user_data.password, user.password):

```

```

        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Неверное имя пользователя или пароль",
            headers={"WWW-Authenticate": "Bearer"},
        )

    # Check if user is active
    if not user.is_active:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Пользователь неактивен"
        )

    # Create access token
    access_token = create_access_token(data={"sub": user.username})

    return Token(
        access_token=access_token,
        token_type="bearer"
    )

```

Листинг А.7 – Файл event_service.py

```

from sqlalchemy.orm import Session
from fastapi import HTTPException, status, UploadFile
from typing import List, Optional, Dict, Any
from datetime import datetime

from ..models import User, Event
from ..repositories import EventRepository, InvitationRepository,
SubscriptionRepository
from ..schemas import EventCreate, EventUpdate, EventDetail
from ..utils.upload import save_image, delete_file

class EventService:
    @staticmethod
    async def create_event(
        db: Session,
        event_data: EventCreate,
        images: Optional[List[UploadFile]],
        current_user: User
    ):
        """Create a new event."""
        # Create event
        event = EventRepository.create(db, event_data, current_user.id)

        # Add images if provided
        if images:
            for image in images:
                image_path = await save_image(image, folder="events")
                EventRepository.add_image(db, event.id, image_path)

        # Create invitations for followers if not explicitly specified
        if event_data.invited_users is None:
            InvitationRepository.create_invitations_for_followers(db, event.id,
current_user.id)
        # Create invitations for specified users
        elif event_data.invited_users:
            for user_id in event_data.invited_users:
                InvitationRepository.create_invitation(db, event.id, user_id)

        # Refresh the event to get the images
        db.refresh(event)
        return event.to_dict()

```

```

    @staticmethod
    def get_event_by_id(db: Session, event_id: int, current_user: Optional[User]
= None):
        """Get event by ID with additional details."""
        event = EventRepository.get_by_id(db, event_id)
        if not event:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="Мероприятие не найдено"
            )

        # Create event detail dict
        event_dict = event.to_dict()

        # Add creator info
        event_dict["creator"] = {
            "id": event.creator.id,
            "username": event.creator.username,
            "full_name": event.creator.full_name,
            "phone": event.creator.phone or "", # Ensure phone is never null
            "created_at": event.creator.created_at.isoformat() if
event.creator.created_at else datetime.now().isoformat(), # Ensure created_at
is never null
            "profile_picture": event.creator.profile_picture
        }

        # Get participants count
        event_dict["participants_count"] =
EventRepository.get_participants_count(db, event_id)

        # Check if current user is participant and invited if user is logged in
        if current_user:
            event_dict["is_user_participant"] =
EventRepository.is_user_participant(db, event_id, current_user.id)
            event_dict["is_user_invited"] = EventRepository.is_user_invited(db,
event_id, current_user.id)
        else:
            event_dict["is_user_participant"] = False
            event_dict["is_user_invited"] = False

        return event_dict

    @staticmethod
    async def update_event(
        db: Session,
        event_id: int,
        event_data: EventUpdate,
        current_user: User
    ):
        """Update an event."""
        # Get event
        event = EventRepository.get_by_id(db, event_id)
        if not event:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="Мероприятие не найдено"
            )

        # Check if user is the creator
        if event.creator_id != current_user.id:
            raise HTTPException(
                status_code=status.HTTP_403_FORBIDDEN,
                detail="У вас нет прав на редактирование этого мероприятия"
            )

```

```

    )

    # Check if event has already passed
    if event.event_date < datetime.now():
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Нельзя редактировать прошедшее мероприятие"
        )

    # Handle deleted images - if existing_images is provided
    if event_data.existing_images is not None:
        # Get current images
        current_images = [img.image_path for img in event.images]

        # Find images to delete
        images_to_delete = [path for path in current_images if path not in
event_data.existing_images]

        # Delete the image files
        for image_path in images_to_delete:
            await delete_file(image_path)

    # Update event
    updated_event = EventRepository.update(db, event_id, event_data)
    return updated_event.to_dict()

@staticmethod
async def add_event_image(
    db: Session,
    event_id: int,
    image: UploadFile,
    current_user: User
):
    """Add an image to an event."""
    # Get event
    event = EventRepository.get_by_id(db, event_id)
    if not event:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Мероприятие не найдено"
        )

    # Check if user is the creator
    if event.creator_id != current_user.id:
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN,
            detail="У вас нет прав на редактирование этого мероприятия"
        )

    # Check if event has already passed
    if event.event_date < datetime.now():
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Нельзя редактировать прошедшее мероприятие"
        )

    # Save image
    image_path = await save_image(image, folder="events")

    # Add image to event
    event_image = EventRepository.add_image(db, event_id, image_path)
    return event_image

@staticmethod

```

```

async def delete_event(db: Session, event_id: int, current_user: User):
    """Delete an event."""
    # Get event
    event = EventRepository.get_by_id(db, event_id)
    if not event:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Мероприятие не найдено"
        )

    # Check if user is the creator
    if event.creator_id != current_user.id:
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN,
            detail="У вас нет прав на удаление этого мероприятия"
        )

    # Delete images
    for image in event.images:
        await delete_file(image.image_path)

    # Delete event
    EventRepository.delete(db, event_id)
    return {"status": "success", "message": "Мероприятие успешно удалено"}

@staticmethod
async def update_event(
    db: Session,
    event_id: int,
    event_data: EventUpdate,
    current_user: User
):
    """Update an event."""
    # Get event
    event = EventRepository.get_by_id(db, event_id)
    if not event:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Мероприятие не найдено"
        )

    # Check if user is the creator
    if event.creator_id != current_user.id:
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN,
            detail="У вас нет прав на редактирование этого мероприятия"
        )

@staticmethod
async def update_event(
    db: Session,
    event_id: int,
    event_data: EventUpdate,
    current_user: User
):
    """Update an event."""
    # Get event
    event = EventRepository.get_by_id(db, event_id)
    if not event:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Мероприятие не найдено"
        )

```

```

        # Check if user is the creator
        if event.creator_id != current_user.id:
            raise HTTPException(
                status_code=status.HTTP_403_FORBIDDEN,
                detail="У вас нет прав на редактирование этого мероприятия"

    @staticmethod
    async def delete_event_image(db: Session, image_id: int, current_user: User):
        """Delete an event image."""
        # Check if image exists and get the event
        from ..models import EventImage
        image = db.query(EventImage).filter(EventImage.id == image_id).first()
        if not image:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="Изображение не найдено"
            )

        # Get event
        event = EventRepository.get_by_id(db, image.event_id)

        # Check if user is the creator
        if event.creator_id != current_user.id:
            raise HTTPException(
                status_code=status.HTTP_403_FORBIDDEN,
                detail="У вас нет прав на удаление изображений этого
мероприятия"
            )

        # Delete image file
        await delete_file(image.image_path)

        # Delete image from database
        EventRepository.delete_image(db, image_id)
        return {"status": "success", "message": "Изображение успешно удалено"}

    @staticmethod
    def get_user_events(db: Session, user_id: int, skip: int = 0, limit: int =
10):
        """Get events created by a user."""
        events = EventRepository.get_events_by_creator(db, user_id, skip, limit)
        return [event.to_dict() for event in events]

    @staticmethod
    def get_user_feed(db: Session, user_id: int, skip: int = 0, limit: int = 10):
        """Get events from users the current user is following."""
        events = EventRepository.get_user_feed(db, user_id, skip, limit)
        return [event.to_dict() for event in events]

    @staticmethod
    async def delete_event_image(db: Session, image_id: int, current_user: User):
        """Delete an event image."""
        # Check if image exists and get the event
        from ..models import EventImage
        image = db.query(EventImage).filter(EventImage.id == image_id).first()
        if not image:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="Изображение не найдено"
            )

        # Get event
        event = EventRepository.get_by_id(db, image.event_id)

```

```

        # Check if user is the creator
        if event.creator_id != current_user.id:
            raise HTTPException(
                status_code=status.HTTP_403_FORBIDDEN,
                detail="У вас нет прав на удаление изображений этого
мероприятия"
            )

        # Delete image file
        await delete_file(image.image_path)

        # Delete image from database
        EventRepository.delete_image(db, image_id)
        return {"status": "success", "message": "Изображение успешно удалено"}

    @staticmethod
    def get_user_events(db: Session, user_id: int, skip: int = 0, limit: int =
10):
        """Get events created by a user."""
        events = EventRepository.get_events_by_creator(db, user_id, skip, limit)
        return [event.to_dict() for event in events]

    @staticmethod
    def get_user_feed(db: Session, user_id: int, skip: int = 0, limit: int = 10):
        """Get events from users the current user is following."""
        events = EventRepository.get_user_feed(db, user_id, skip, limit)
        return [event.to_dict() for event in events]

    @staticmethod
    def search_events(db: Session, query: str, upcoming_only: bool = False, skip:
int = 0, limit: int = 10):
        """Search events by title, location, or description."""
        if not query or len(query) < 3:
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail="Поисковый запрос должен содержать не менее 3 символов"
            )

        events = EventRepository.search(db, query, skip, limit, upcoming_only)
        return [event.to_dict() for event in events]

    @staticmethod
    def prepare_events_for_response(events: List[Event]) -> List[Dict[str, Any]]:
        """Convert Event models to dictionaries with serialized image paths."""
        return [event.to_dict() for event in events]

```

Листинг A.8 – Файл interaction_service.py

```

from sqlalchemy.orm import Session
from fastapi import HTTPException, status
from datetime import datetime

from ..models import User, Review
from ..repositories import CommentRepository, ReviewRepository, EventRepository
from ..schemas import CommentCreate, CommentUpdate, ReviewCreate, ReviewUpdate

class CommentService:
    @staticmethod
    def create_comment(db: Session, comment_data: CommentCreate, event_id: int,
current_user: User):
        """Create a new comment."""
        # Check if event exists
        event = EventRepository.get_by_id(db, event_id)
        if not event:

```

```

        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Мероприятие не найдено"
        )

        # Create comment with event_id
        comment = CommentRepository.create(db, comment_data, event_id,
current_user.id)
        return comment

    @staticmethod
    def update_comment(db: Session, comment_id: int, comment_data: CommentUpdate,
current_user: User):
        """Update a comment."""
        # Update comment
        updated_comment = CommentRepository.update(db, comment_id, comment_data,
current_user.id)
        if not updated_comment:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="Комментарий не найден или у вас нет прав на его
редактирование"
            )

        return updated_comment

    @staticmethod
    def delete_comment(db: Session, comment_id: int, current_user: User):
        """Delete a comment."""
        # Delete comment
        deleted = CommentRepository.delete(db, comment_id, current_user.id)
        if not deleted:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="Комментарий не найден или у вас нет прав на его
удаление"
            )

        return {"status": "success", "message": "Комментарий успешно удален"}

    @staticmethod
    def get_event_comments(db: Session, event_id: int, skip: int = 0, limit: int
= 100):
        """Get all comments for an event."""
        # Check if event exists
        event = EventRepository.get_by_id(db, event_id)
        if not event:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="Мероприятие не найдено"
            )

        # Get comments
        return CommentRepository.get_by_event(db, event_id, skip, limit)

class ReviewService:
    @staticmethod
    def create_review(db: Session, event_id: int, review_data: ReviewCreate,
current_user: User):
        """Create a new review."""
        # Check if event exists
        event = EventRepository.get_by_id(db, event_id)
        if not event:
            raise HTTPException(

```

```

        status_code=status.HTTP_404_NOT_FOUND,
        detail="Мероприятие не найдено"
    )

    # Check if event has passed
    if event.event_date > datetime.now():
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Нельзя оставить отзыв о мероприятии, которое еще не
прошло"
        )

    # Check if user has already reviewed
    existing_review = db.query(Review).filter(
        Review.event_id == event_id,
        Review.user_id == current_user.id
    ).first()

    if existing_review:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Вы уже оставили отзыв о данном мероприятии"
        )

    # Create review
    review = ReviewRepository.create(db, event_id, review_data,
current_user.id)
    if not review:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Не удалось создать отзыв"
        )

    return review

    @staticmethod
    def update_review(db: Session, review_id: int, review_data: ReviewUpdate,
current_user: User):
        """Update a review."""
        # Update review
        updated_review = ReviewRepository.update(db, review_id, review_data,
current_user.id)
        if not updated_review:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="Отзыв не найден или у вас нет прав на его
редактирование"
            )

        return updated_review

    @staticmethod
    def delete_review(db: Session, review_id: int, current_user: User):
        """Delete a review."""
        # Delete review
        deleted = ReviewRepository.delete(db, review_id, current_user.id)
        if not deleted:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="Отзыв не найден или у вас нет прав на его удаление"
            )

        return {"status": "success", "message": "Отзыв успешно удален"}

```

```

    @staticmethod
    def get_event_reviews(db: Session, event_id: int, skip: int = 0, limit: int
= 100):
        """Get all reviews for an event."""
        # Check if event exists
        event = EventRepository.get_by_id(db, event_id)
        if not event:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="Мероприятие не найдено"
            )

        # Get reviews
        reviews = ReviewRepository.get_by_event(db, event_id, skip, limit)

        # Get average rating
        avg_rating = ReviewRepository.get_average_rating(db, event_id)

        return {
            "reviews": reviews,
            "average_rating": avg_rating
        }

```

Листинг А.9 – Файл invitation_service.py

```

from sqlalchemy.orm import Session
from fastapi import HTTPException, status

from ..models import User, Event
from ..repositories import InvitationRepository, EventRepository, UserRepository

class InvitationService:
    @staticmethod
    def create_invitation(db: Session, event_id: int, user_id: int, current_user:
User):
        """Create an invitation for a user to an event."""
        # Check if event exists and user is the creator
        event = EventRepository.get_by_id(db, event_id)
        if not event:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="Мероприятие не найдено"
            )

        # Check if user is the creator
        if event.creator_id != current_user.id:
            raise HTTPException(
                status_code=status.HTTP_403_FORBIDDEN,
                detail="Только создатель может приглашать пользователей"
            )

        # Check if user exists
        user = UserRepository.get_by_id(db, user_id)
        if not user:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="Пользователь не найден"
            )

        # Check if already invited
        if EventRepository.is_user_invited(db, event_id, user_id):
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail="Пользователь уже приглашен"
            )

```

```

        )

        # Create invitation
        invitation = InvitationRepository.create_invitation(db, event_id,
user_id)
        return invitation

    @staticmethod
    def delete_invitation(db: Session, invitation_id: int, current_user: User):
        """Delete an invitation."""
        # Check if invitation exists
        invitation = InvitationRepository.get_by_id(db, invitation_id)
        if not invitation:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="Приглашение не найдено"
            )

        # Check if user is the creator of the event
        event = EventRepository.get_by_id(db, invitation.event_id)
        if event.creator_id != current_user.id:
            raise HTTPException(
                status_code=status.HTTP_403_FORBIDDEN,
                detail="Только создатель может удалять приглашения"
            )

        # Delete invitation
        deleted = InvitationRepository.delete_invitation(db, invitation_id)
        if not deleted:
            raise HTTPException(
                status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
                detail="Не удалось удалить приглашение"
            )

        return {"status": "success", "message": "Приглашение успешно удалено"}

    @staticmethod
    def get_event_invitations(db: Session, event_id: int, current_user: User,
skip: int = 0, limit: int = 100):
        """Get all invitations for an event."""
        # Check if event exists
        event = EventRepository.get_by_id(db, event_id)
        if not event:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="Мероприятие не найдено"
            )

        # Check if user is the creator
        if event.creator_id != current_user.id:
            raise HTTPException(
                status_code=status.HTTP_403_FORBIDDEN,
                detail="Только создатель может просматривать список
приглашенных"
            )

        # Get invitations
        return InvitationRepository.get_event_invitations(db, event_id, skip,
limit)

    @staticmethod
    def get_user_invitations(db: Session, current_user: User, skip: int = 0,
limit: int = 100):
        """Get all invitations for the current user."""

```

```

        # Get invitations
        return InvitationRepository.get_user_invitations(db, current_user.id,
skip, limit)

```

Листинг А.10 – Файл participation_service.py

```

from sqlalchemy.orm import Session
from fastapi import HTTPException, status

from ..models import User, Event
from ..repositories import ParticipationRepository, EventRepository

class ParticipationService:
    @staticmethod
    def join_event(db: Session, event_id: int, current_user: User):
        """Join an event as a participant."""
        # Check if event exists
        event = EventRepository.get_by_id(db, event_id)
        if not event:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="Мероприятие не найдено"
            )

        # Check if already a participant
        if EventRepository.is_user_participant(db, event_id, current_user.id):
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail="Вы уже являетесь участником этого мероприятия"
            )

        # Add participant
        participant = ParticipationRepository.add_participant(db, event_id,
current_user.id)
        return participant

    @staticmethod
    def leave_event(db: Session, event_id: int, current_user: User):
        """Leave an event as a participant."""
        # Check if event exists
        event = EventRepository.get_by_id(db, event_id)
        if not event:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="Мероприятие не найдено"
            )

        # Check if a participant
        if not EventRepository.is_user_participant(db, event_id,
current_user.id):
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail="Вы не являетесь участником этого мероприятия"
            )

        # Remove participant
        removed = ParticipationRepository.remove_participant(db, event_id,
current_user.id)
        if not removed:
            raise HTTPException(
                status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
                detail="Не удалось покинуть мероприятие"
            )

```

```

        return {"status": "success", "message": "Вы успешно покинули
мероприятие"}

    @staticmethod
    def get_event_participants(db: Session, event_id: int, skip: int = 0, limit:
int = 100):
        """Get all participants for an event."""
        # Check if event exists
        event = EventRepository.get_by_id(db, event_id)
        if not event:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="Мероприятие не найдено"
            )

        # Get participants
        return ParticipationRepository.get_event_participants(db, event_id, skip,
limit)

    @staticmethod
    def get_user_participations(db: Session, user_id: int, skip: int = 0, limit:
int = 100):
        """Get all events a user is participating in."""
        # Check if user exists
        from ..repositories import UserRepository
        user = UserRepository.get_by_id(db, user_id)
        if not user:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="Пользователь не найден"
            )

        # Get participations
        return ParticipationRepository.get_user_participations(db, user_id, skip,
limit)

```

Листинг A.11 – Файл subscription_service.py

```

from sqlalchemy.orm import Session
from fastapi import HTTPException, status

from ..models import User
from ..repositories import SubscriptionRepository, UserRepository

class SubscriptionService:
    @staticmethod
    def follow_user(db: Session, followed_id: int, current_user: User):
        """Follow a user."""
        # Check if user exists
        followed_user = UserRepository.get_by_id(db, followed_id)
        if not followed_user:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="Пользователь не найден"
            )

        # Check if trying to follow oneself
        if followed_id == current_user.id:
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail="Вы не можете подписаться на самого себя"
            )

        # Check if already following

```

```

        if SubscriptionRepository.is_following(db, current_user.id, followed_id):
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail="Вы уже подписаны на этого пользователя"
            )

        # Create subscription
        subscription = SubscriptionRepository.follow_user(db, current_user.id,
followed_id)
        return subscription

    @staticmethod
    def unfollow_user(db: Session, followed_id: int, current_user: User):
        """Unfollow a user."""
        # Check if user exists
        followed_user = UserRepository.get_by_id(db, followed_id)
        if not followed_user:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="Пользователь не найден"
            )

        # Check if following
        if not SubscriptionRepository.is_following(db, current_user.id,
followed_id):
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail="Вы не подписаны на этого пользователя"
            )

        # Delete subscription
        unfollow_result = SubscriptionRepository.unfollow_user(db,
current_user.id, followed_id)
        if not unfollow_result:
            raise HTTPException(
                status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
                detail="Не удалось отписаться от пользователя"
            )

        return {"status": "success", "message": "Вы успешно отписались"}

    @staticmethod
    def get_followers(db: Session, user_id: int, skip: int = 0, limit: int =
100):
        """Get all followers of a user."""
        # Check if user exists
        user = UserRepository.get_by_id(db, user_id)
        if not user:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="Пользователь не найден"
            )

        # Get followers
        return SubscriptionRepository.get_followers(db, user_id, skip, limit)

    @staticmethod
    def get_following(db: Session, user_id: int, skip: int = 0, limit: int =
100):
        """Get all users a user is following."""
        # Check if user exists
        user = UserRepository.get_by_id(db, user_id)
        if not user:
            raise HTTPException(

```

```

        status_code=status.HTTP_404_NOT_FOUND,
        detail="Пользователь не найден"
    )

    # Get following
    return SubscriptionRepository.get_following(db, user_id, skip, limit)

@staticmethod
def check_is_following(db: Session, followed_id: int, current_user: User):
    """Check if current user is following another user."""
    # Check if user exists
    followed_user = UserRepository.get_by_id(db, followed_id)
    if not followed_user:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Пользователь не найден"
        )
    def follow_user(db: Session, followed_id: int, current_user: User):
        """Follow a user."""
        # Check if user exists
        followed_user = UserRepository.get_by_id(db, followed_id)
        if not followed_user:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="Пользователь не найден"
            )

        # Check if trying to follow oneself
        if followed_id == current_user.id:
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail="Вы не можете подписаться на самого себя"
            )

        # Check if already following
        if SubscriptionRepository.is_following(db, current_user.id, followed_id):
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail="Вы уже подписаны на этого пользователя"
            )

        # Create subscription
        subscription = SubscriptionRepository.follow_user(db, current_user.id,
followed_id)
        return subscription

    @staticmethod
    def unfollow_user(db: Session, followed_id: int, current_user: User):
        """Unfollow a user."""
        # Check if user exists
        followed_user = UserRepository.get_by_id(db, followed_id)
        if not followed_user:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="Пользователь не найден"
            )

        # Check if following
        if not SubscriptionRepository.is_following(db, current_user.id,
followed_id):
    def follow_user(db: Session, followed_id: int, current_user: User):
        """Follow a user."""
        # Check if user exists
        followed_user = UserRepository.get_by_id(db, followed_id)

```

```

    if not followed_user:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Пользователь не найден"
        )

    # Check if trying to follow oneself
    if followed_id == current_user.id:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Вы не можете подписаться на самого себя"
        )

    # Check if already following
    if SubscriptionRepository.is_following(db, current_user.id, followed_id):
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Вы уже подписаны на этого пользователя"
        )

    # Create subscription
    subscription = SubscriptionRepository.follow_user(db, current_user.id,
followed_id)
    return subscription

@staticmethod
def unfollow_user(db: Session, followed_id: int, current_user: User):
    """Unfollow a user."""
    # Check if user exists
    followed_user = UserRepository.get_by_id(db, followed_id)
    if not followed_user:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Пользователь не найден"
        )

    # Check if following
    if not SubscriptionRepository.is_following(db, current_user.id,
followed_id):
def follow_user(db: Session, followed_id: int, current_user: User):
    """Follow a user."""
    # Check if user exists
    followed_user = UserRepository.get_by_id(db, followed_id)
    if not followed_user:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Пользователь не найден"
        )

    # Check if trying to follow oneself
    if followed_id == current_user.id:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Вы не можете подписаться на самого себя"
        )

    # Check if already following
    if SubscriptionRepository.is_following(db, current_user.id, followed_id):
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Вы уже подписаны на этого пользователя"
        )

    # Create subscription

```

```

        subscription = SubscriptionRepository.follow_user(db, current_user.id,
followed_id)
        return subscription

    @staticmethod
    def unfollow_user(db: Session, followed_id: int, current_user: User):
        """Unfollow a user."""
        # Check if user exists
        followed_user = UserRepository.get_by_id(db, followed_id)
        if not followed_user:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="Пользователь не найден"
            )

        # Check if following
        if not SubscriptionRepository.is_following(db, current_user.id,
followed_id):
            return {"is_following": is_following}

```

Листинг A.12 – Файл telegram_deeplink_service.py

```

from datetime import datetime, timedelta
import uuid

from sqlalchemy.orm import Session

from app.models.telegram import TelegramLinkToken

class TelegramLinkService:
    @staticmethod
    def create_link_token(db: Session, user_id: int) -> str:
        token = f"link_{uuid.uuid4().hex[:16]}"
        expires_at = datetime.utcnow() + timedelta(minutes=15)

        db_token = TelegramLinkToken(user_id=user_id, token=token,
expires_at=expires_at)
        db.add(db_token)
        db.commit()
        db.refresh(db_token)
        return token

    @staticmethod
    def get_user_by_token(db: Session, token: str):
        db_token = db.query(TelegramLinkToken).filter(
            TelegramLinkToken.token == token,
            TelegramLinkToken.expires_at > datetime.utcnow()
        ).first()
        if db_token:
            user = db_token.user
            db.delete(db_token)
            db.commit()
            return user
        return None

```

Листинг A.13 – Файл user_service.py

```

from sqlalchemy.orm import Session
from fastapi import HTTPException, status, UploadFile

from ..repositories import UserRepository
from ..schemas import UserUpdate, UserDetail
from ..utils.upload import save_image, delete_file

```

```

from ..models import User

class UserService:
    @staticmethod
    def get_user_by_id(db: Session, user_id: int):
        """Get user by ID with detail."""
        user = UserRepository.get_by_id(db, user_id)
        if not user:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="Пользователь не найден"
            )

        # Get user stats
        stats = UserRepository.get_user_stats(db, user_id)

        # Create UserDetail object
        user_detail = UserDetail.from_orm(user)
        user_detail.events_count = stats["events_count"]
        user_detail.followers_count = stats["followers_count"]
        user_detail.following_count = stats["following_count"]

        return user_detail

    @staticmethod
    async def update_user(db: Session, user_id: int, user_data: UserUpdate,
current_user: User):
        """Update user profile."""
        # Check if user is authorized to update
        if user_id != current_user.id:
            raise HTTPException(
                status_code=status.HTTP_403_FORBIDDEN,
                detail="Нет прав на изменение профиля другого пользователя"
            )

        # Check if username exists if changing
        if user_data.username and user_data.username != current_user.username:
            user = UserRepository.get_by_username(db, user_data.username)
            if user:
                raise HTTPException(
                    status_code=status.HTTP_400_BAD_REQUEST,
                    detail="Имя пользователя уже занято"
                )

        # Check if phone exists if changing
        if user_data.phone and user_data.phone != current_user.phone:
            user = UserRepository.get_by_phone(db, user_data.phone)
            if user:
                raise HTTPException(
                    status_code=status.HTTP_400_BAD_REQUEST,
                    detail="Телефон уже зарегистрирован"
                )

        # Update user
        updated_user = UserRepository.update(db, user_id, user_data)
        if not updated_user:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="Пользователь не найден"
            )

        return updated_user

    @staticmethod

```

```

    async def update_profile_picture(db: Session, user_id: int, profile_picture:
UploadFile, current_user: User):
    """Update user profile picture."""
    # Check if user is authorized to update
    if user_id != current_user.id:
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN,
            detail="Нет прав на изменение профиля другого пользователя"
        )

    # Delete old profile picture if exists
    if current_user.profile_picture:
        await delete_file(current_user.profile_picture)

    # Save new profile picture
    image_path = await save_image(profile_picture, folder="profiles")

    # Update user profile picture path
    user_data = UserUpdate(profile_picture=image_path)
    updated_user = UserRepository.update(db, user_id, user_data)

    return updated_user

@staticmethod
def search_users(db: Session, query: str, skip: int = 0, limit: int = 10):
    """Search users by username, phone, or full name."""
    if not query or len(query) < 3:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Поисковый запрос должен содержать не менее 3 символов"
        )

    return UserRepository.search(db, query, skip, limit)

```

Приложение Б

(рекомендуемое)

Тестирование функционала регистрации

Таблица Б.1 – Тест-кейсы для регистрации пользователя и их результаты

№ п/п	Входные данные	Ожидаемый результат	Наблюдаемый результат	Тест пройден/не пройден
1	Имя: us Пароль: pa2!	Сообщения о некорректном вводе имени пользователя и пароля	Сообщения об ошибках выведены на экран	Тест пройден (рис. Б.1)
2	Имя: @@ Пароль: password	Сообщения о некорректном вводе имени пользователя и пароля	Сообщения об ошибках выведены на экран	Тест пройден (рис. Б.2)
3	Имя: us Пароль: password...d	Сообщения о некорректном вводе имени пользователя и пароля	Сообщения об ошибках выведены на экран	Тест пройден (рис. Б.3)
4	Имя: user_@...@ Пароль: password1@d...d	Сообщения о некорректном вводе имени пользователя и пароля	Сообщения об ошибках выведены на экран	Тест пройден (рис. Б.4)
5	Имя: user_12...2 Пароль: pass	Сообщения о некорректном вводе имени пользователя и пароля	Сообщения об ошибках выведены на экран	Тест пройден (рис. Б.5)
6	Имя: user_12...2 Пароль: password12@	Сообщение о некорректном вводе имени пользователя	Сообщение об ошибке выведено на экран	Тест пройден (рис. Б.6)
7	Имя: user_12 Пароль: pasword12@	Имя и пароль введены корректно, перенаправление к этапу ввода личной информации	Перенаправление к форме заполнения личной информации	Тест пройден (рис. Б.7)
8	Имя: user_12 Пароль: passwordd...d	Сообщение о некорректном вводе пароля	Сообщение об ошибке выведено на экран	Тест пройден (рис. Б.8)
9	Имя: user_12#### Пароль: pa2!	Сообщения о некорректном вводе имени пользователя и пароля	Сообщения об ошибках выведены на экран	Тест пройден (рис. Б.9)
10	ФИО: Резепов...в Номер телефона: 68958595856	Сообщения о некорректном вводе ФИО и номера телефона	Сообщения об ошибках выведены на экран	Тест пройден (рис. Б.10)
11	ФИО: Резепов Александр Александрович Номер телефона: 8123456789548548	Сообщения о некорректном вводе номера телефона	Сообщение об ошибке выведено на экран	Тест пройден (рис. Б.11)
12	ФИО: ##### ##...# Номер телефона: +71589	Сообщения о некорректном вводе ФИО и номера телефона	Сообщения об ошибках выведены на экран	Тест пройден (рис. Б.12)
13	ФИО: ##### ##...# Номер телефона: +71589658789	Сообщения о некорректном вводе ФИО	Сообщение об ошибке выведено на экран	Тест пройден (рис. Б.13)
14	ФИО: ##### ##...# Номер телефона: p1548595689584	Сообщения о некорректном вводе ФИО и номера телефона	Сообщения об ошибках выведены на экран	Тест пройден (рис. Б.14)
15	ФИО: Rezepov Alex Alexandrovic Номер телефона: 452	Сообщения о некорректном вводе номера телефона	Сообщение об ошибке выведено на экран	Тест пройден (рис. Б.15)

Продолжение таблицы Б.1

№ п/п	Входные данные	Ожидаемый результат	Наблюдаемый результат	Тест пройден/не пройден
16	ФИО: Rezerov Alex Alexandrovichgggg... g Номер телефона: 6666666666	Сообщения о некорректном вводе ФИО и номера телефона	Сообщения об ошибках выведены на экран	Тест пройден (рис. Б.16)
17	ФИО: Резепов Александр Александрович Номер телефона: 89630587859	Аккаунт пользователя успешно создан	Аккаунт пользователя успешно создан	Тест пройден (рис. Б.17)

Регистрация

Создайте аккаунт для доступа к платформе

1 Учетные данные — 2 Личная информация — 3 Завершение

Имя пользователя *

us

Имя пользователя должно содержать минимум 3 символа

Пароль *

pa2!

Пароль должен содержать минимум 6 символов

Подтверждение пароля *

....

← Назад

Далее →

Уже есть аккаунт? [Войти](#)

Рисунок Б.1 – Результат теста 1 для этапа ввода учетных данных

Регистрация

Создайте аккаунт для доступа к платформе

1 Учетные данные — 2 Личная информация — 3 Завершение

Имя пользователя *

Имя пользователя должно содержать минимум 3 символа

Пароль *

Пароль должен содержать минимум 1 букву, 1 цифру и 1 специальный символ

Подтверждение пароля *

← Назад
Далее →

Уже есть аккаунт? [Войти](#)

Рисунок Б.2 – Результат теста 2 для этапа ввода учетных данных

Регистрация

Создайте аккаунт для доступа к платформе

1 Учетные данные — 2 Личная информация — 3 Завершение

Имя пользователя *

Имя пользователя должно содержать минимум 3 символа

Пароль *

Пароль должен содержать не более 30 символов

Подтверждение пароля *

← Назад
Далее →

Уже есть аккаунт? [Войти](#)

Рисунок Б.3 – Результат теста 3 для этапа ввода учетных данных

[illegible]

Рисунок Б.5 – Результат теста 5 для этапа ввода учетных данных

Регистрация

Создайте аккаунт для доступа к платформе

✓ Учетные данные

2 Личная информация

3 Завершение

ФИО *



Иванов Иван Иванович

Телефон *



Telegram Username (необязательно)



@username

← Назад

Далее →

Рисунок Б.7 – Результат теста 7 для этапа ввода учетных данных

Регистрация

Создайте аккаунт для доступа к платформе

1 Учетные данные — 2 Личная информация — 3 Завершение

Имя пользователя *

Пароль *

Пароль должен содержать не более 30 символов

Подтверждение пароля *

← Назад
Далее →

Уже есть аккаунт? [Войти](#)

Рисунок Б.8 – Результат теста 8 для этапа ввода учетных данных

Регистрация

Создайте аккаунт для доступа к платформе

1 Учетные данные — 2 Личная информация — 3 Завершение

Имя пользователя *

Имя пользователя может содержать только буквы, цифры и символ "_"

Пароль *

Пароль должен содержать минимум 6 символов

Подтверждение пароля *

← Назад
Далее →

Уже есть аккаунт? [Войти](#)

Рисунок Б.9 – Результат теста 9 для этапа ввода учетных данных

Регистрация

Создайте аккаунт для доступа к платформе

✓ Учетные данные —

2 Личная информация —

3 Завершение

👤

ФИО *

ФИО должно содержать не более 50 символов

☎️

Телефон *

Введите корректный номер телефона формата (+7(8)8596587859)

👤

Telegram Username (необязательно)

[← Назад](#)

Далее →

Рисунок Б.10 – Результат теста 1 для этапа ввода личной информации

Регистрация

Создайте аккаунт для доступа к платформе

✓

Учетные данные

2

Личная информация

3

Завершение

ФИО *

Резепов Александр Александрович

Телефон *

8123456789548548

Введите корректный номер телефона формата (+7/8)8596587859

Telegram Username (необязательно)

@username

← Назад

Далее →

Рисунок Б.11 – Результат теста 2 для этапа ввода личной информации

Регистрация

Создайте аккаунт для доступа к платформе

✓ Учетные данные

2 Личная информация

3 Завершение

ФИО *



#####

ФИО должно содержать не более 50 символов

Телефон *



+71589

Телефонный номер должен иметь формат (+7/8)8596587859

Telegram Username (необязательно)



@username

← Назад

Далее →

Рисунок Б.12 – Результат теста 3 для этапа ввода личной информации

Регистрация

Создайте аккаунт для доступа к платформе

✓ Учетные данные

2 Личная информация

3 Завершение

ФИО *

 #####

ФИО должно содержать только русские и английские буквы

Телефон *

 +71589658789

Telegram Username (необязательно)

 @username

← Назад

Далее →

Рисунок Б.13 – Результат теста 4 для этапа ввода личной информации

Регистрация

Создайте аккаунт для доступа к платформе

✓ Учетные данные
2 Личная информация
3 Завершение

ФИО *

#####

ФИО должно содержать только русские и английские буквы

Телефон *

p1548595689584

Телефонный номер должен иметь формат (+7/8)8596587859

Telegram Username (необязательно)

@username

← Назад
Далее →

Рисунок Б.14 – Результат теста 5 для этапа ввода личной информации

Регистрация

Создайте аккаунт для доступа к платформе

✓ Учетные данные
2 Личная информация
3 Завершение

ФИО *

Rezepov Alex Alexandrovich

Телефон *

452

Телефонный номер должен иметь формат (+7/8)8596587859

Telegram Username (необязательно)

@username

← Назад
Далее →

Рисунок Б.15 – Результат теста 6 для этапа ввода личной информации

А

Администратор

Главная

Мероприятия

Подписки

Профиль

Создать мероприятие

Выйти

Создавайте и управляйте событиями с лёгкостью

Платформа для организации мероприятий и встреч с вашим сообществом

Создать событие

Предстоящие мероприятия

Конференция по веб-разработке

Конференция по веб-разработке

16 июня 2025 г.

13:24

Технопарк 'Сколково'

Подробнее

Мастер-класс: Мобильная разработка

Мастер-класс: Мобильная разработка

3 июня 2025 г.

13:24

Коворкинг 'Рабочая станция'

Подробнее

ИТ-вечеринка

ИТ-вечеринка

24 июня 2025 г.

13:24

Конференц-зал 'Горизонт'

Подробнее

Рисунок Б.17 – Результат теста 7 для этапа ввода личной информации