

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

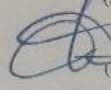
на тему

ПРОГРАММА ДЛЯ ПУБЛИКАЦИЙ
МЕРОПРИЯТИЙ ДОПОЛНИТЕЛЬНОГО
ОБРАЗОВАНИЯ

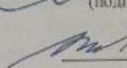
Студент

 Захаров А.С.
(подпись, дата) 29.05.2026

Руководитель

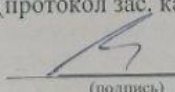
 01.06.2026 Федюнин Р.Н.
(подпись, дата)

Нормоконтролёр

 01.06.2026 Бычков А. С.

Работа допущена к защите (протокол зас. кафедры от 22.05.2026 г. №014)

Заведующий кафедрой

 М.А. Митрохин
(подпись)

Работа защищена с отметкой хорошо (протокол заседания ГЭК от

15 июня 2026 № 34)

Секретарь ГЭК


(подпись)

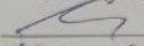
Е.И. Гурин

Пенза 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
 М.А. Митрохин
« 07 » 04 20 26 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Захаров Антон Сергеевич гр.22ВВП1 факультета вычислительной техники
2. Тема работы Программа для публикаций мероприятий дополнительного образования
Тема утверждена приказом ПГУ №347/о от "07" апреля 2026 г.
3. Руководитель бакалаврской работы к.т.н., доцент Федюнин Р. Н.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

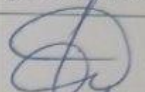
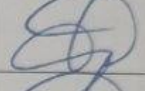
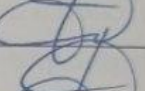
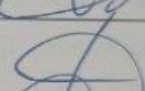
Разработать веб-приложение «Программа для публикации мероприятий дополнительного образования», обеспечивающее публикацию, поиск и регистрацию на мероприятия дополнительного образования. Приложение должно поддерживать разграничение ролей пользователей (неавторизованный пользователь, авторизованный пользователь, администратор), реализовывать функции модерации публикуемых мероприятий, а также предоставлять личный кабинет с историей записей и управлением профилем. Клиентская часть реализуется на библиотеке React совместно с Tailwind CSS и Radix UI, серверная — на платформе Node.js с фреймворком Express, хранение данных обеспечивается СУБД PostgreSQL с использованием ORM Prisma.

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметной области в сфере дополнительного образования и изучить аналоги существующих решений для публикации и поиска мероприятий. Необходимо определить функциональные требования к системе и спроектировать архитектуру клиент-серверного веб-приложения с применением паттерна MVVM. Следует изучить принципы построения REST API на платформе Node.js с фреймворком Express, освоить разработку клиентской части на библиотеке React, а также спроектировать и реализовать структуру реляционной базы данных средствами PostgreSQL и

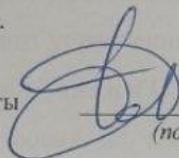
Prisma ORM. Необходимо реализовать систему аутентификации на основе JWT, разработать функционал модерации мероприятий и управления пользователями, а также провести тестирование разработанного приложения.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	40	07.04.2026– 17.04.2026	
2	Анализ требований	12	18.04.2026– 21.04.2026	
3	Выбор технологий, применяемых при проектировании и разработке	28	22.04.2026– 29.04.2026	
4	Описание алгоритма	8	30.04.2026– 01.05.2026	
5	Разработка приложения «Программа для публикаций мероприятий дополнительного образования»	68	02.05.2026– 19.05.2026	
6	Оформление пояснительной записки	12	20.05.2026– 21.05.2026	

Дата выдачи задания 07.04.2026 г.

Руководитель бакалаврской работы



(подпись, дата)

Задание к исполнению принял студент



(подпись, дата)

Работу к защите допустить
Декан факультета

 10.06.2026 Л.Р. Рязанова

(подпись, дата)

Реферат

Бакалаврская работа содержит 106 листа, 29 рисунков, 15 таблиц, 16 источников, 3 приложения.

ПУБЛИКАЦИЯ МЕРОПРИЯТИЙ, ВЕБ-ПРИЛОЖЕНИЕ, ДОПОЛНИТЕЛЬНОЕ ОБРАЗОВАНИЕ, REACT, NODE.JS, EXPRESS, POSTGRESQL, PRISMA, REST API, JWT, TAILWIND CSS

Цель работы – разработка программы публикации мероприятий дополнительного образования, обеспечивающей удобный поиск, запись и управление событиями для широкой аудитории пользователей.

Разработанная система предназначена для размещения, поиска и записи на мероприятия дополнительного образования, а также для административного управления публикуемым контентом. Приложение реализовано в виде клиент-серверной системы с трёхуровневой архитектурой, включающей клиентский слой, серверный слой бизнес-логики и слой хранения данных.

Клиентская часть разработана на библиотеке React с использованием инструмента сборки Vite. Пользовательский интерфейс включает страницы просмотра каталога мероприятий, детальной информации о событии, личного кабинета и административного управления. Адаптивная вёрстка реализована с применением CSS-фреймворка Tailwind CSS.

Серверная часть построена на платформе Node.js с использованием фреймворка Express.js. Система аутентификации и авторизации реализована на основе токенов JWT. Для взаимодействия с базой данных PostgreSQL применяется ORM Prisma.

					ПГУ1.090301.10.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Программа для публикаций мероприятий дополнительного образования	Лит.	Лист	Листов
Разроб.		Захаров А. С.		28.05.24				
Провер.		Федюнин Р. Н.		28.05.24			2	86
Н. Контр		Бычкова А. С.		28.05.24		ФВТ, каф. ВТ		
Утверд.		Митрохин М. А.		28.05.24		гр. 22ВВП1		

Содержание

Перечень сокращений и обозначений	5
Введение.....	7
1. Программа публикации мероприятий дополнительного образования.....	9
1.1 Анализ предметной области.....	9
1.1.1 Анализ аналогичных решений	10
1.1.2 Постановка задачи.....	17
1.1.3 Функционал приложения	18
1.2 Выбор технологий и средств разработки приложения.....	21
1.2.1 Выбор целевой платформы	22
1.2.2 Выбор языка программирования	23
1.2.3 Выбор фреймворков и библиотек.....	25
1.2.4 Выбор СУБД и ORM.....	29
1.2.6 Выбор среды разработки	31
1.2.7 Выбор архитектурного решения приложения.....	33
1.3 Реализация приложения.....	37
1.3.1 Описание работы приложения.....	38
1.3.2 Структура базы данных	39
1.3.3 Описание компонентов.....	45
1.3.4 Основные алгоритмы взаимодействия клиента и сервера.....	60
1.4 Тестирование	69
1.4.1 Регистрация пользователя	69
1.4.2 Авторизация пользователя	70
1.4.3 Просмотр каталога мероприятий и фильтрация	70

1.4.4	Запись на мероприятие и добавление в избранное.....	71
1.4.5	Создание мероприятия.....	72
1.4.6	Изменение профиля пользователя.....	73
1.4.7	Функционал администратора.....	74
1.5	Руководство пользователя.....	75
1.5.1	Результат работы приложения.....	75
2.	Безопасность жизнедеятельности.....	80
2.1	Организация рабочего места.....	81
2.2	Микроклимат помещения.....	82
2.3	Режим труда и отдыха	83
	Заключение	84
	Список использованных источников	85
	Приложение А	87
	Приложение Б.....	93
	Приложение В.....	103

Перечень сокращений и обозначений

В этой работе используются следующие сокращения и обозначения

ВКР – выпускная квалификационная работа

СУБД – система управления базами данных

API (Application Programming Interface) – программный интерфейс взаимодействия приложений

bcrypt – библиотека для хэширования паролей

CORS (Cross-Origin Resource Sharing) – механизм управления доступом к ресурсам между источниками

CSS (Cascading Style Sheets) – каскадные таблицы стилей

DOM (Document Object Model) – объектная модель документа

Express.js – минималистичный веб-фреймворк для платформы Node.js

HTML (HyperText Markup Language) – язык гипертекстовой разметки

HTTP (HyperText Transfer Protocol) – протокол передачи гипертекста

ID (Identifier) – идентификатор объекта

JSON (JavaScript Object Notation) – текстовый формат обмена данными

JWT (JSON Web Token) – стандарт токена авторизации на основе JSON

MVC (Model-View-Controller) – архитектурный паттерн разделения приложения на три компонента

MVP (Model-View-Presenter) – архитектурный паттерн, развивающий принципы MVC

MVVM (Model-View-ViewModel) – архитектурный паттерн с двусторонним связыванием данных

Node.js – среда выполнения JavaScript на стороне сервера на базе движка V8

ORM (Object-Relational Mapping) – объектно-реляционное отображение

Prisma – ORM нового поколения для экосистемы Node.js с декларативным описанием схемы данных

React – библиотека JavaScript для построения пользовательских интерфейсов

					ПГУ1.090301.39.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		5

REST (Representational State Transfer) – архитектурный стиль взаимодействия компонентов распределённого приложения

SQL (Structured Query Language) – язык структурированных запросов

Tailwind CSS – утилитарный CSS-фреймворк для стилизации интерфейса

UI (User Interface) – пользовательский интерфейс

UML (Unified Modeling Language) – унифицированный язык моделирования

URL (Uniform Resource Locator) – унифицированный локатор ресурсов

UUID (Universally Unique Identifier) – универсальный уникальный идентификатор

Vite – инструмент сборки и разработки для современных веб-приложений

					ПГУ1.090301.39.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

В современном обществе дополнительное образование приобретает всё большую значимость: кружки, секции, мастер-классы и другие образовательные мероприятия пользуются устойчивым спросом среди широкой аудитории. Вместе с тем поиск подходящих мероприятий нередко затруднён из-за разрозненности информации и отсутствия единого удобного сервиса. Существующие решения либо ориентированы на развлекательный контент, либо характеризуются закрытой моделью публикации, не позволяющей независимым организаторам свободно размещать свои программы. В связи с этим разработка открытой системы для публикации, поиска и записи на мероприятия дополнительного образования является актуальной задачей.

Одним из перспективных подходов к решению данной задачи является построение клиент-серверного веб-приложения, позволяющего организаторам публиковать мероприятия, а пользователям — находить и записываться на них. Применение современных веб-технологий, таких как React, Node.js, Express и PostgreSQL, обеспечивает высокую производительность, удобство использования и надёжное хранение данных. Разграничение прав доступа на основе JWT-аутентификации гарантирует безопасность взаимодействия между пользователями и системой.

Цели выпускной квалификационной работы:

- освоение современных методов разработки клиент-серверных веб-приложений;
- изучение принципов построения REST API и работы с реляционными базами данных;
- реализация системы публикации мероприятий дополнительного образования с использованием React, Node.js, Express и PostgreSQL.
- Задачи, решаемые в ходе выполнения работы:

- проведение анализа предметной области и изучение аналогичных решений;
- определение функциональных требований и проектирование архитектуры системы;
- реализация серверной части приложения на платформе Node.js с фреймворком Express;
- реализация клиентской части приложения на библиотеке React;
- разработка системы аутентификации и авторизации пользователей на основе JWT;
- реализация функционала модерации мероприятий и управления пользователями;
- тестирование системы и оценка качества разработанного приложения.

					ПГУ1.090301.39.001 ПЗ	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

1. Программа публикации мероприятий дополнительного образования

1.1 Анализ предметной области

В настоящее время наблюдается устойчивый интерес к мероприятиям дополнительного образования, к которым относятся кружки, секции, мастер-классы, творческие встречи и другие формы внеформального обучения. [1] Подобные мероприятия позволяют получать новые знания, развивать практические навыки и повышать уровень личной и профессиональной подготовки вне рамок традиционного образовательного процесса. При этом они ориентированы на широкую аудиторию, включая детей, подростков и взрослых пользователей.

Для организации и поиска таких событий уже существуют различные информационные решения. Например, сервис Яндекс.Афиша предоставляет возможность просматривать афишу мероприятий и осуществлять поиск по заданным параметрам. Однако данная платформа в большей степени ориентирована на развлекательные события и не предполагает свободного размещения информации всеми желающими организаторами. Это ограничивает её применение в сфере дополнительного образования, где важны не только обзор мероприятий, но и возможность непосредственного участия различных учреждений и частных лиц в публикации собственных программ.

Наряду с этим существуют специализированные ресурсы, предназначенные для работы в сфере дополнительного образования, в частности Портал «ПФДО». Такие системы обеспечивают доступ к образовательным программам и помогают пользователям находить подходящие варианты обучения. Вместе с тем им присуща централизованная модель наполнения контентом, а также ограниченные возможности для включения частных организаторов и независимых авторов мероприятий. В результате подобные решения не в полной мере закрывают

					ПГУ1.090301.39.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		9

потребности пользователей, которым требуется более гибкий и удобный инструмент для взаимодействия с образовательными событиями.

Кроме того, при поиске мероприятий дополнительного образования пользователи нередко сталкиваются с рядом затруднений. Информация о событиях может быть распределена по разным источникам, иметь неоднородную структуру и не всегда содержать полные сведения о времени проведения, месте, условиях участия и организаторе. Это усложняет процесс выбора подходящего мероприятия и снижает доступность информации для конечного пользователя. Особенно актуальна данная проблема для родителей, студентов и других категорий граждан, заинтересованных в быстром подборе подходящих вариантов обучения и развития.

Таким образом, анализ предметной области показывает, что существует потребность в универсальном информационном решении, которое обеспечивало бы свободную публикацию мероприятий дополнительного образования, удобный поиск и фильтрацию событий, а также эффективное взаимодействие между организаторами и участниками. Разработка такого сервиса является актуальной, поскольку позволит объединить в одной системе функции размещения, поиска и отклика на мероприятия, сделав доступ к ним более простым и удобным.

1.1.1 Анализ аналогичных решений

В ходе анализа существующих решений на отечественном рынке информационных систем, предназначенных для публикации и поиска мероприятий, было выявлено, что полностью идентичных разрабатываемому приложению систем не существует. Однако можно выделить ряд сервисов, обладающих частично схожим функционалом.

Наиболее близким аналогом является платформа Портал «ПФДО». К частично схожим решениям также относятся сервисы Яндекс.Афиша, несмотря на наличие ограничений, связанных с публикацией пользовательских мероприятий.

1.1.1.1 Портал «ПФДО»

На рисунке 1 изображена главная страница Портал «ПФДО»

					ПГУ1.090301.39.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		10

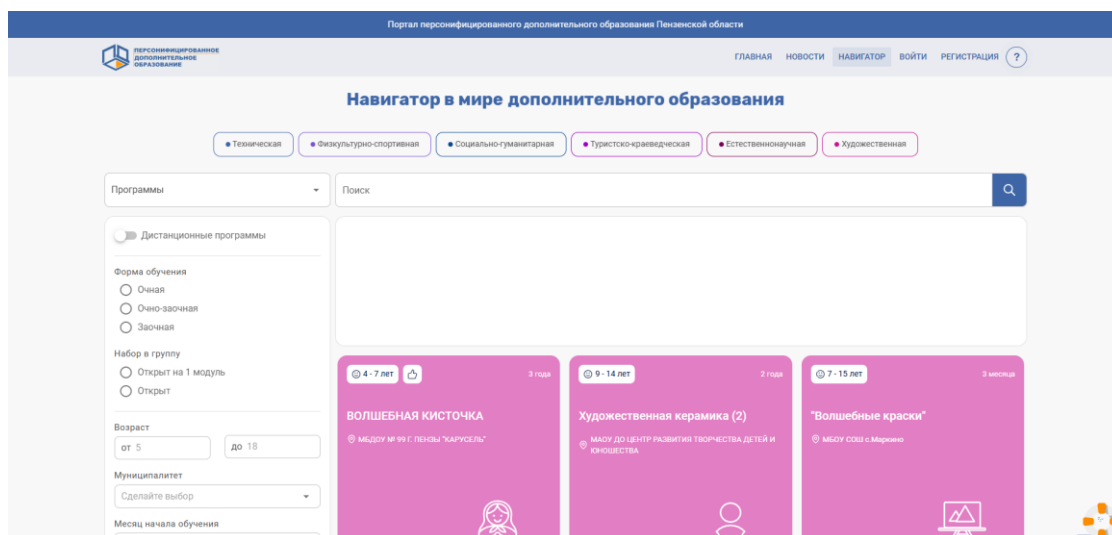


Рисунок 1 – Главная страница сервиса Портал «ПФДО»

На рисунке 2 показан поиск на портале Портал «ПФДО»

Рисунок 2 – Форма для фильтрации и поиска секций на портале Портал «ПФДО»

Результат поиска представлен на рисунке 3

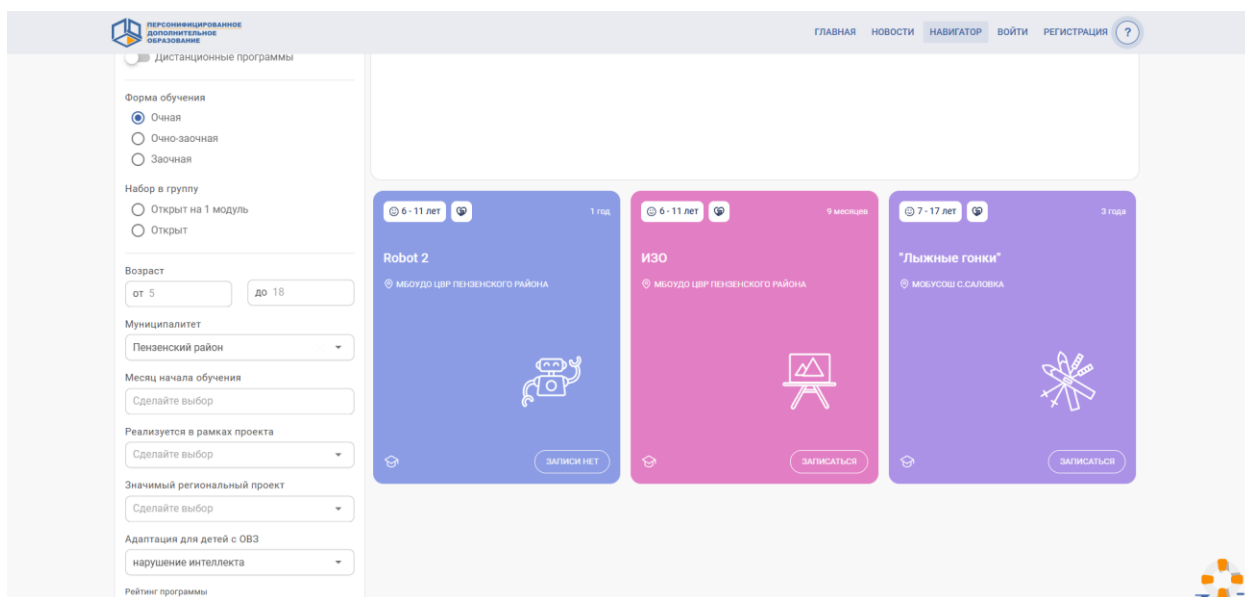


Рисунок 3 – Результаты поиска на портале Портал «ПФДО»

На рисунке 4 представлена страница с информацией о внешкольной секции

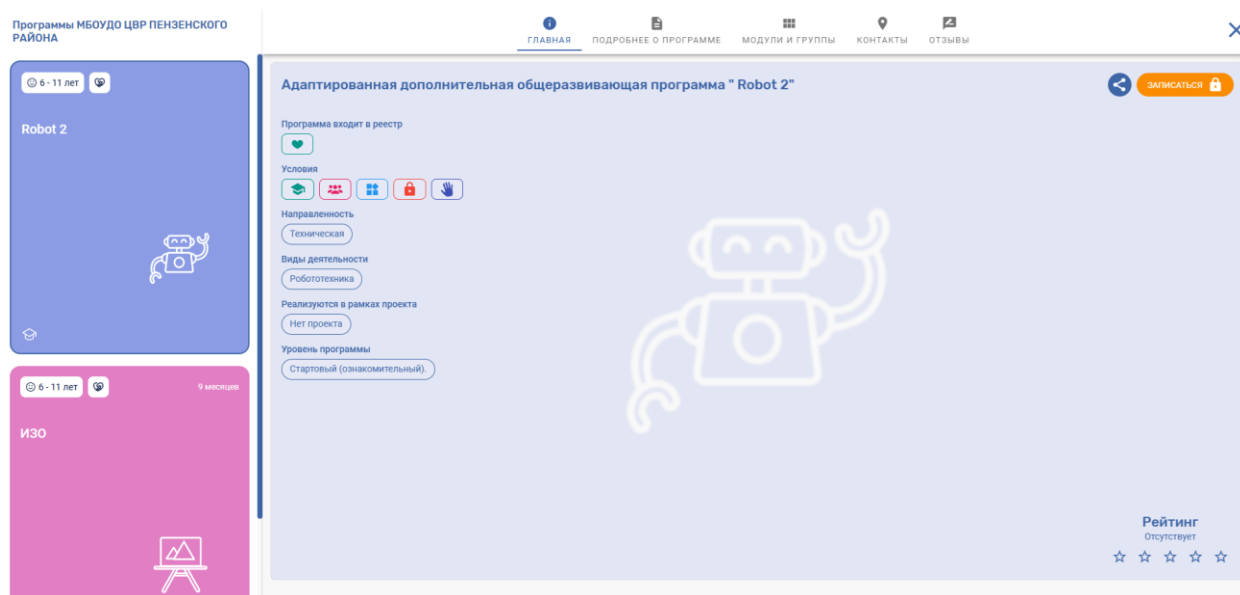


Рисунок 4 – Информация о выбранной внешкольной секции

Платформа «Навигатор дополнительного образования детей» представляет собой специализированную информационную систему, ориентированную на сферу дополнительного (внешкольного) образования. Система предназначена для поиска

и выбора образовательных программ, кружков, секций и иных мероприятий для детей и подростков.

Основной функционал системы включает:

- каталог образовательных программ и мероприятий;
- фильтрацию по возрасту, направлению подготовки, региону и другим параметрам;
- возможность регистрации пользователей;
- подачу заявок на участие в выбранных программах;
- ведение учета образовательных организаций и программ.

Данная система ориентирована на централизованное размещение программ от образовательных учреждений и организаций, прошедших предварительную модерацию.

Преимущества:

- специализация на дополнительном образовании;
- развитая система поиска и фильтрации;
- поддержка регистрации пользователей и подачи заявок.

Недостатки:

- отсутствует возможность свободной публикации мероприятий любыми пользователями;
- ограниченная гибкость для частных организаторов.

1.1.1.2 Сервис «Яндекс.Афиша»

На рисунке 5 представлена главная страница сервиса «Яндекс.Афиша»

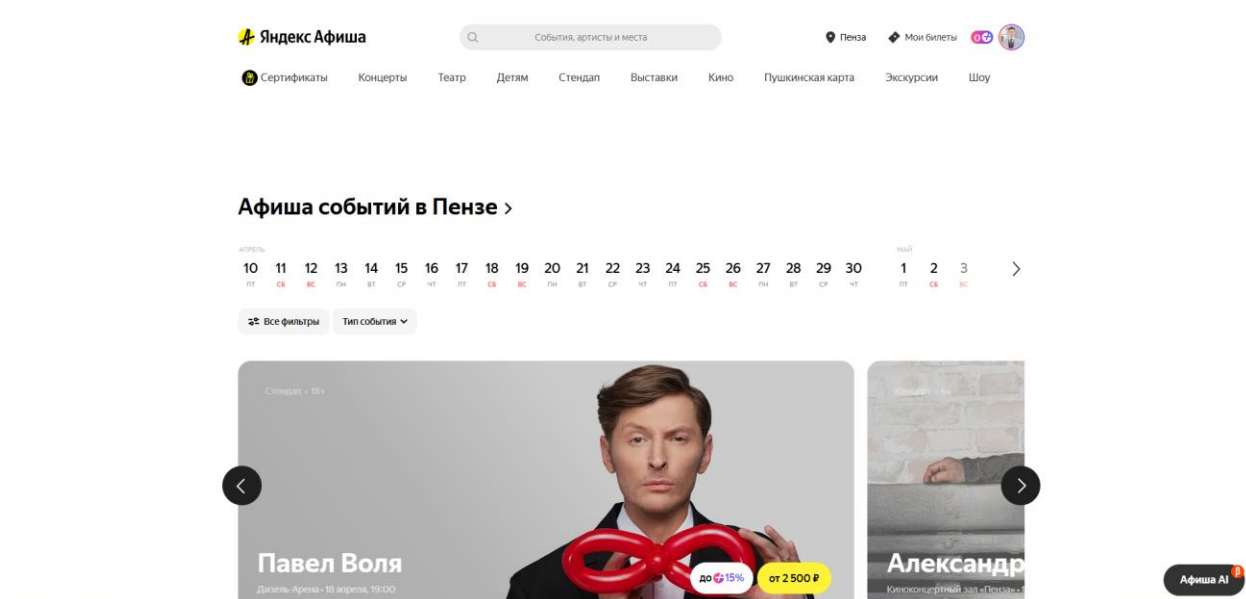


Рисунок 5 – Главная страница сервиса «Яндекс.Афиша»

На рисунке 6 представлено модальное окно для фильтрации мероприятий в общем списке

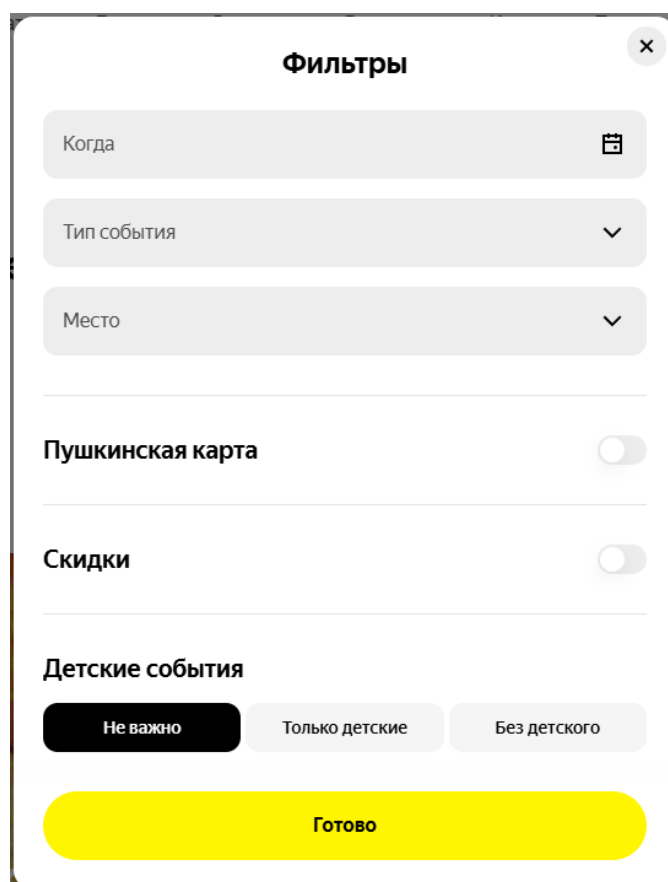


Рисунок 6 – Модальное окно фильтров

На рисунке 7 представлены результаты поиска мероприятий

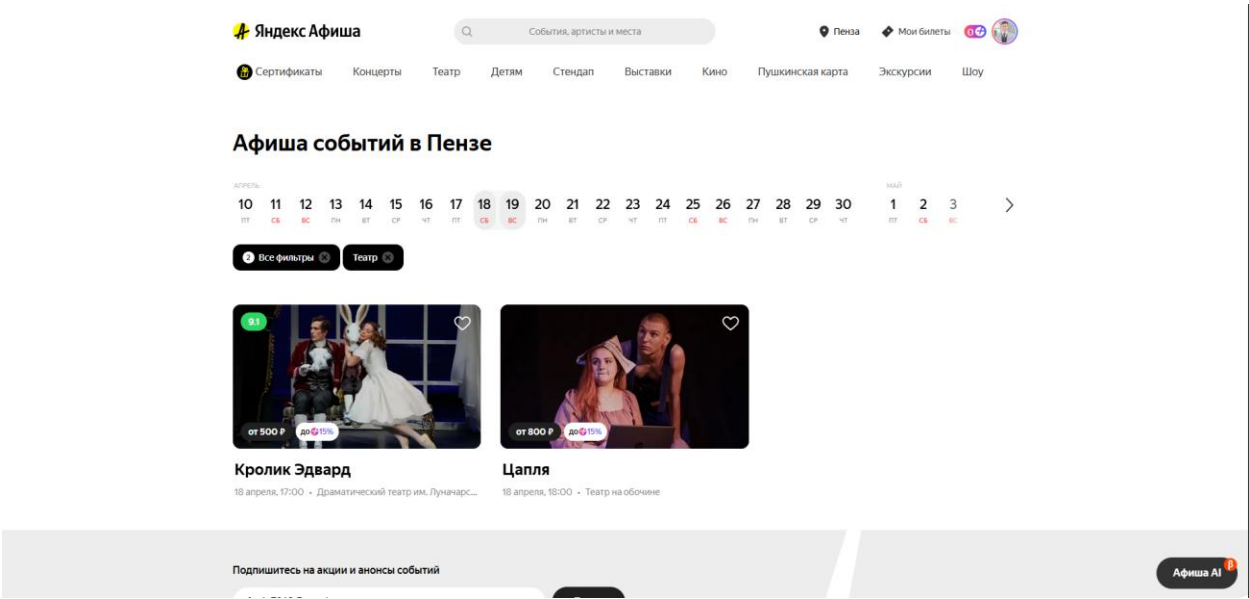


Рисунок 7 – Результат поиска

На рисунке 8 представлена страница с информацией о найденном мероприятии

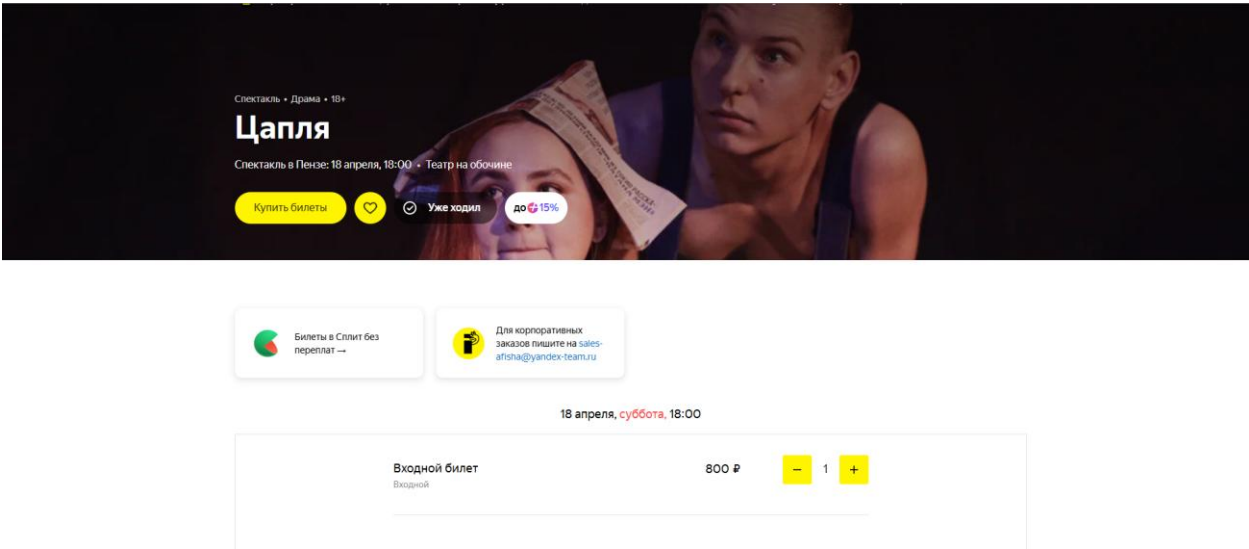


Рисунок 8 – Страница с информацией о мероприятии

Сервис Яндекс.Афиша представляет собой популярную платформу для поиска культурных и развлекательных мероприятий, таких как концерты, театральные постановки, киносеансы и выставки.

Основные функции сервиса:

- поиск мероприятий по различным категориям;
- фильтрация по дате, месту проведения и типу события;
- просмотр подробной информации о мероприятии;
- возможность приобретения билетов через интегрированные сервисы.

Преимущества:

- удобный и интуитивно понятный интерфейс;
- развитая система поиска и рекомендаций;
- высокая популярность и широкая аудитория.

Недостатки:

- публикация мероприятий доступна только партнёрам и организациям;
- отсутствует возможность свободного добавления событий обычными пользователями;
- отсутствует функционал взаимодействия между организатором и участниками внутри системы;
- сервис ориентирован преимущественно на развлекательные мероприятия, а не на дополнительное образование.

1.1.1.3 Вывод по анализу аналогов

В результате проведённого анализа можно сделать вывод, что существующие решения либо ориентированы на агрегирование и распространение информации о мероприятиях, либо на централизованную публикацию образовательных программ от организаций.

При этом в рассмотренных системах отсутствует совокупность следующих функций:

					ПГУ1.090301.39.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		16

- свободная публикация мероприятий любыми зарегистрированными пользователями;
- удобная система записи на мероприятия;
- встроенное взаимодействие между организаторами и участниками;
- специализация на мероприятиях дополнительного образования с возможностью гибкой настройки.

Таким образом, разрабатываемое приложение направлено на устранение выявленных недостатков и объединение функциональных возможностей в рамках единой системы, что подтверждает актуальность и практическую значимость темы выпускной квалификационной работы.

1.1.2 Постановка задачи

Необходимо разработать веб-приложение для публикации, поиска и регистрации на мероприятия дополнительного образования. Приложение должно обеспечивать выполнение следующих функций:

Возможность пользователям с ролью организатора создавать, редактировать и удалять мероприятия, а также публиковать их в системе. При создании мероприятия должны указываться основные сведения: название, описание, дата и время проведения, формат, место проведения, возрастная категория, количество мест, изображение и иные необходимые параметры.

Обработка поисковых запросов пользователей: поиск по названию, тематике, дате, формату проведения, возрастной категории и другим параметрам, а также использование фильтров для удобного отбора мероприятий.

Наличие личного кабинета пользователя с информацией о зарегистрированных на мероприятия, истории записей и данных профиля.

Возможность регистрации пользователей на мероприятия с отображением актуальной информации о доступных местах и статусе записи.

Возможность взаимодействия между участниками и организаторами, включая просмотр списка зарегистрированных пользователей, а также обмен сообщениями или обратную связь в рамках системы.

Для пользователя с ролью администратора необходимо предусмотреть функции модерации публикуемых мероприятий, рассмотрения жалоб и контроля корректности функционирования системы.

Для обеспечения работоспособности приложения необходимо определить следующие минимальные системные требования:

- браузер Google Chrome, Mozilla Firefox, Opera или Microsoft Edge с поддержкой HTML5, CSS3 и JavaScript;

- минимальная скорость интернет-соединения: 2–3 Мбит/с

- минимальный объём оперативной памяти: 1 ГБ;

- рекомендуемый объём оперативной памяти: 2 ГБ и выше;

1.1.3 Функционал приложения

Функции данного веб-приложения можно разделить на три группы:

- для неавторизованного пользователя;

- для авторизованного пользователя;

- для администратора.

Неавторизованный пользователь обладает ограниченным функционалом. Он может найти мероприятие по названию или другим критериям, просмотреть список доступных мероприятий и перейти на страницу выбранного мероприятия для получения дополнительной информации. Кроме того, неавторизованный пользователь может авторизоваться (войти в систему). Диаграмма вариантов использования приложения для неавторизованного пользователя приведена на рисунке 9.

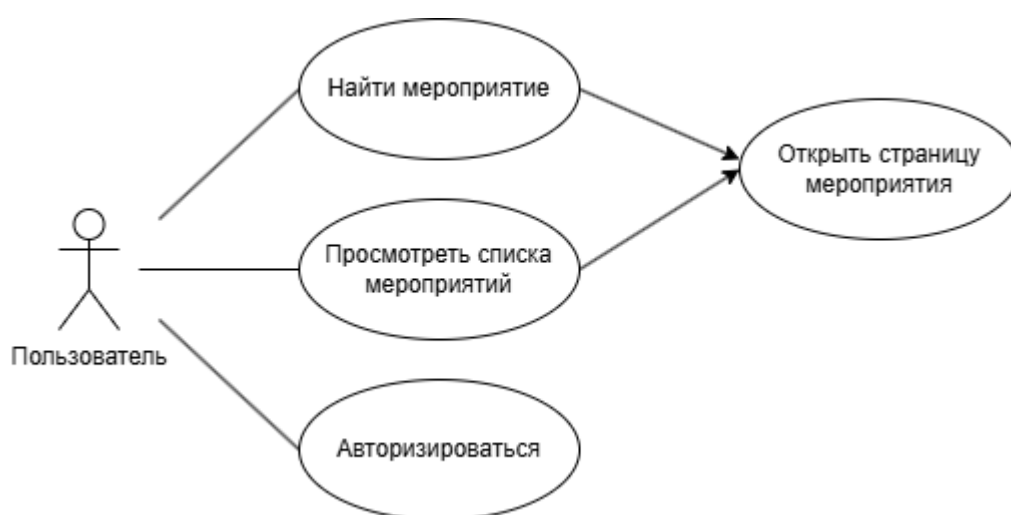


Рисунок 9 – Диаграмма прецедентов неавторизованного пользователя

Авторизованный пользователь обладает расширенным функционалом. Помимо функций, доступных неавторизованному пользователю, он имеет доступ к личному кабинету и может создавать новые мероприятия. Он может редактировать и удалять созданные им мероприятия, записываться на мероприятия других организаторов и отменять свои записи. Кроме того, авторизованный пользователь может добавлять интересующие его мероприятия в список избранного, удалять их из него и просматривать список избранных мероприятий. Личный кабинет пользователя позволяет ему просматривать список собственных мероприятий и информацию о записях на них. Пользователь также может выйти из своей учетной записи.

Диаграмма вариантов использования приложения для авторизованного пользователя приведена на рисунке 10.



Рисунок 10 – Диаграмма прецедентов авторизованного пользователя

Администратор системы обладает наибольшим уровнем доступа. Он может просматривать список всех мероприятий и выполнять поиск по различным критериям, а также открывать страницу выбранного мероприятия. Администратор также может просматривать список неподтверждённых мероприятий и подтверждать или отклонять их для модерации добавляемых пользователями событий. Кроме того, администратор может управлять пользователями системы: просматривать список пользователей, блокировать или удалять их учетные записи

и назначать пользователям различные роли. Наконец, администратор может выйти из своей учетной записи.

Диаграмма вариантов использования приложения для администратора приведена на рисунке 11.

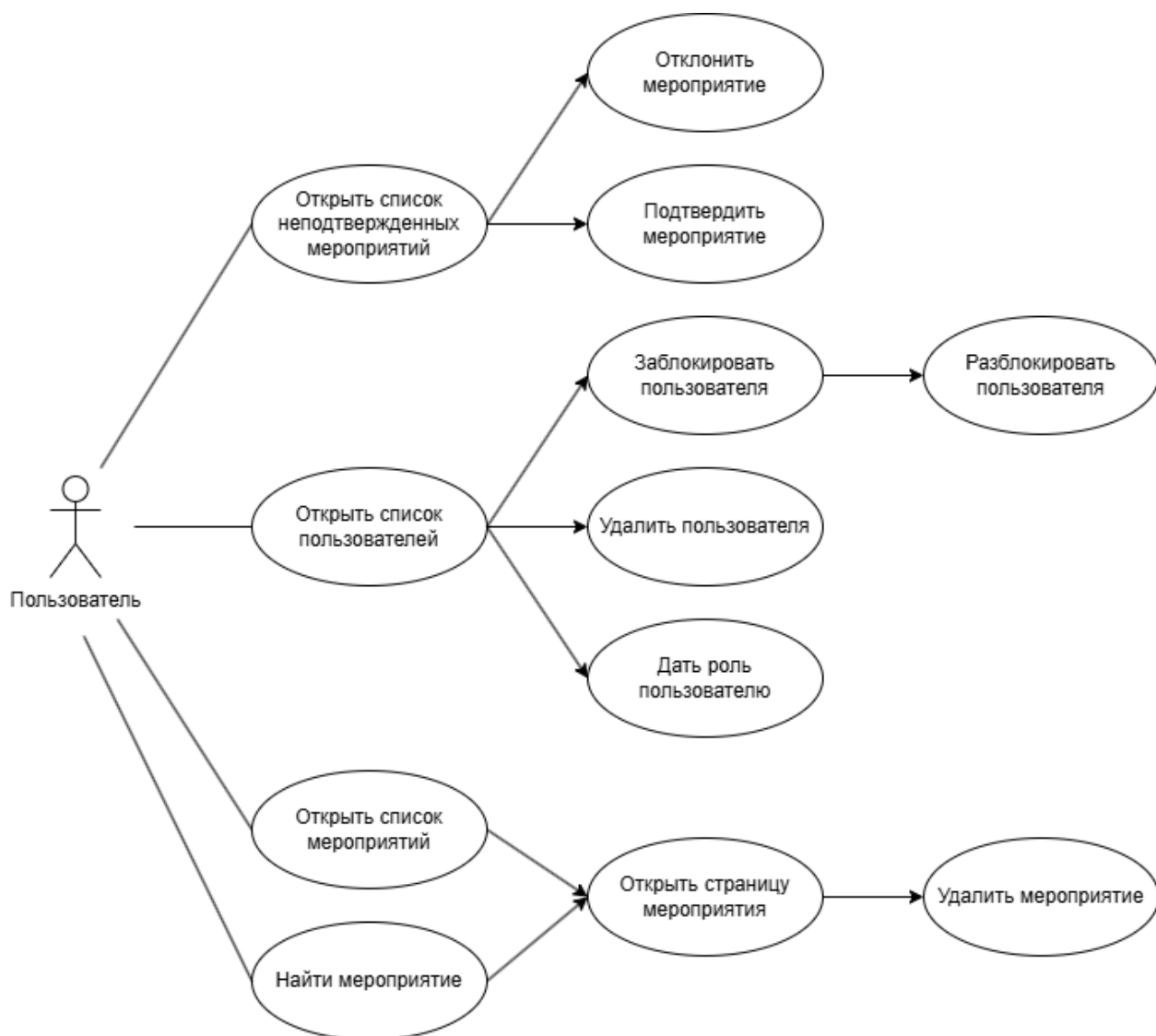


Рисунок 11 – Диаграмма прецедентов администратора

1.2 Выбор технологий и средств разработки приложения

В данном разделе представлен сравнительный анализ языков программирования, фреймворков, систем управления базами данных и интегрированных сред разработки. На основании проведённого анализа

сформирован технологический стек, обеспечивающий оптимальное соотношение скорости разработки, удобства сопровождения кодовой базы и производительности итогового продукта.

1.2.1 Выбор целевой платформы

Определение целевой платформы является первостепенным архитектурным решением, поскольку именно оно задаёт вектор для всех последующих технологических выборов и определяет характер взаимодействия пользователя с системой.

К разрабатываемому приложению для публикации мероприятий дополнительного образования предъявляются два ключевых требования: широкий охват целевой аудитории (педагоги, организаторы, родители и учащиеся) и минимизация трудозатрат на разработку и дальнейшую поддержку.

При выборе платформы рассматривались три основных варианта. Нативное мобильное приложение обеспечивает высокое качество пользовательского опыта и доступ к аппаратным возможностям устройства, однако требует разработки отдельных версий для платформ iOS и Android, что существенно увеличивает стоимость и сроки создания продукта. Desktopное приложение целесообразно при работе со сложными интерфейсами и локальными данными, однако значительно ограничивает аудиторию и не обеспечивает мобильность. Веб-приложение лишено указанных недостатков: оно доступно через браузер с любого устройства без установки, обновляется централизованно на сервере и не требует прохождения модерации магазинов приложений.

Наибольший охват пользователей теоретически достигается при наличии как веб-версии, так и нативного мобильного клиента. Вместе с тем подобный подход влечёт необходимость параллельной разработки и поддержки нескольких кодовых баз, что создаёт избыточную нагрузку в условиях небольшой команды.

Оптимальным решением с учётом сформулированных требований признано создание единого веб-приложения с адаптивным интерфейсом. Такой подход

					ПГУ1.090301.39.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		22

обеспечивает корректное отображение и функционирование системы как на мобильных устройствах, так и на персональных компьютерах без дублирования логики в нескольких проектах.

1.2.2 Выбор языка программирования

Выбор языка программирования определяет экосистему инструментов, доступных разработчику, а также архитектурные возможности будущего приложения. Для сравнения были рассмотрены три наиболее востребованных в сфере веб-разработки языка: Python, C# и JavaScript.

1.2.2.1 Python

Python – интерпретируемый язык программирования общего назначения с лаконичным синтаксисом и обширной экосистемой библиотек. Он занимает лидирующие позиции в индексах популярности (TIOBE, Stack Overflow Developer Survey) и широко применяется в серверной веб-разработке, анализе данных и машинном обучении.

Разработку веб-сервисов на Python поддерживают фреймворки Django, Flask и FastAPI. Django предоставляет готовые механизмы аутентификации, ORM и административную панель, что существенно ускоряет разработку. Flask отличается минималистичностью и гибкостью. FastAPI ориентирован на высокопроизводительные асинхронные API с автоматической генерацией документации.

Вместе с тем применение Python в контексте данного проекта сопряжено с рядом ограничений. Язык не используется в браузере, что при полностековой разработке вынуждает задействовать два разных языка – Python на сервере и JavaScript на клиенте. Это увеличивает когнитивную нагрузку на разработчика и усложняет переиспользование общей логики (валидации, типов данных).

1.2.2.2 C#

C# – строго типизированный объектно-ориентированный язык, разработанный компанией Microsoft и функционирующий в рамках платформы

					ПГУ1.090301.39.001 ПЗ	Лист
						23
Изм.	Лист	№ докум.	Подпись	Дата		

.NET. Область его применения охватывает корпоративные веб-сервисы, настольные и мобильные приложения, а также системы машинного обучения на базе ML.NET.

Веб-разработка на C# строится преимущественно вокруг платформы ASP.NET Core – кроссплатформенного фреймворка для создания высокопроизводительных HTTP-сервисов. К ключевым характеристикам платформы относятся: общезыковая среда выполнения (CLR), богатая стандартная библиотека классов, встроенные средства внедрения зависимостей и высокие показатели производительности.

Недостатком применения C# в рамках данного проекта является сложность экосистемы: для небольшого веб-приложения инфраструктура .NET оказывается избыточной. Кроме того, как и в случае с Python, разработчику потребовался бы отдельный язык на стороне клиента, что нарушает единообразие стека.

1.2.2.3 JavaScript

JavaScript – мультипарадигменный интерпретируемый язык, изначально созданный для обеспечения интерактивности веб-страниц в браузере. Сегодня он является де-факто стандартом клиентской разработки и поддерживается всеми актуальными браузерными платформами без каких-либо дополнительных зависимостей.

На стороне клиента JavaScript усиливается экосистемой фреймворков – React, Angular, Vue.js – предоставляющих инструментарий для компонентного построения интерфейса, управления состоянием и клиентской маршрутизации [2]. На стороне сервера JavaScript функционирует благодаря среде выполнения Node.js, превращающей его в язык общего назначения, пригодный для построения полноценных серверных приложений на базе фреймворков Express, Fastify и Hono.

Ключевым преимуществом JavaScript является возможность использования единого языка во всём стеке разработки – от клиентской части до серверной логики. Это существенно снижает когнитивную нагрузку на разработчика, упрощает

					ПГУ1.090301.39.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		24

переиспользование кода (например, функций валидации) и унифицирует процессы сопровождения проекта.

На основании проведённого анализа JavaScript был выбран в качестве единственного языка программирования для реализации данного проекта. Указанное решение обусловлено его уникальной способностью охватывать весь стек разработки, богатой экосистемой инструментов, а также широкой поддержкой со стороны профессионального сообщества.

1.2.3 Выбор фреймворков и библиотек

При проектировании архитектуры веб-приложения существенную роль играет выбор фреймворков, определяющих структуру кода и предоставляющих готовые инструменты для решения типовых задач. В рамках данного проекта рассматривались фреймворки для клиентской и серверной частей приложения.

1.2.3.1 React

React – библиотека JavaScript, разработанная компанией Meta (ранее Facebook), предназначенная для построения пользовательских интерфейсов. В основе библиотеки лежит компонентный подход: интерфейс разбивается на независимые, логически завершённые компоненты, каждый из которых инкапсулирует собственную разметку, стили и поведение. Реактивность обеспечивается механизмом виртуального DOM [4]: при изменении состояния React вычисляет разницу между предыдущим и актуальным деревом элементов и обновляет только те части реального DOM, которые действительно изменились, что существенно повышает производительность при работе со сложными интерфейсами.

К ключевым достоинствам React применительно к задаче разработки приложения для публикации мероприятий дополнительного образования относятся [3]:

– компонентная структура – возможность переиспользования готовых компонентов (карточек мероприятий, фильтров, форм) в различных частях приложения без дублирования кода;

– эффективная работа с динамическими данными – встроенные механизмы управления состоянием (useState, useReducer, useContext) [12] позволяют реализовывать интерактивные фильтры, формы регистрации и личный кабинет пользователя;

– развитая экосистема – наличие готовых решений для маршрутизации (React Router), управления состоянием (Redux, MobX) и взаимодействия с API.

В данном проекте React используется совместно с библиотекой компонентов Radix UI [6] и утилитарным CSS-фреймворком Tailwind CSS [5], что позволяет создавать доступный и стилистически согласованный интерфейс при минимальных затратах на написание базовых UI-элементов.

1.2.3.2 Vue.js

Vue.js – прогрессивный JavaScript-фреймворк для разработки пользовательских интерфейсов, построенный на принципах реактивности и декларативного связывания данных. Представление автоматически обновляется при изменении состояния модели, что избавляет разработчика от ручного управления DOM. Фреймворк поддерживает компонентный подход, аналогичный React, и располагает собственной экосистемой: маршрутизатор Vue Router и менеджер состояния Pinia обеспечивают построение полноценных одностраничных приложений.

Отличительной особенностью Vue является низкий порог входа: синтаксис шаблонов близок к стандартному HTML, что упрощает освоение фреймворка. Вместе с тем Vue исторически применяется реже при разработке крупных корпоративных приложений с высокой степенью переиспользования компонентов, чем React. Принимая во внимание масштаб и характер разрабатываемой системы,

а также более широкое распространение React в профессиональном сообществе, выбор был сделан в пользу последнего.

1.2.3.3 Angular

Angular – полнофункциональный клиентский фреймворк, разрабатываемый компанией Google. Он реализован на TypeScript и предоставляет исчерпывающий набор встроенных инструментов: систему маршрутизации, механизмы работы с формами, HTTP-клиент и внедрение зависимостей. Angular предполагает строгую структуру проекта с разделением на модули, компоненты, сервисы и директивы, что делает его особенно подходящим для крупных корпоративных приложений с обширным объёмом бизнес-логики.

Вместе с тем применение Angular в рамках данного проекта представляется нецелесообразным по ряду причин. Высокий порог входа и объёмная инфраструктура фреймворка создают избыточную сложность для проекта среднего масштаба. Строгие архитектурные требования Angular увеличивают время начальной настройки и усложняют разработку в условиях ограниченных сроков. Таким образом, Angular был исключён из рассмотрения как инструмент, не соответствующий требованиям проекта по критерию соразмерности решения задаче.

Результаты сравнительного анализа рассмотренных фронтенд-фреймворков позволяют выделить следующие ключевые различия. React, разработанный компанией Meta, отличается обширной экосистемой и средним порогом входа; маршрутизация подключается через стороннюю библиотеку React Router, однако богатое сообщество и зрелость инструментария компенсируют этот недостаток. Vue.js, поддерживаемый независимым сообществом, характеризуется наиболее низким порогом входа и интуитивным синтаксисом шаблонов, однако уступает React по распространённости в крупных проектах и размеру экосистемы. Angular, развиваемый компанией Google, предоставляет полный набор встроенных инструментов, включая маршрутизацию и HTTP-клиент, однако требует

					ПГУ1.090301.39.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		27

обязательного использования TypeScript и отличается наиболее высоким порогом входа, что делает его избыточным для проектов среднего масштаба.

Таким образом, по совокупности критериев -соразмерности задаче и распространённости в профессиональном сообществе – наиболее целесообразным выбором для данного проекта признан React.

1.2.3.4 Node.js и Express

Для реализации серверной части приложения выбрана платформа Node.js [7] совместно с фреймворком Express. Node.js представляет собой среду выполнения JavaScript на стороне сервера, основанную на высокопроизводительном движке V8. Асинхронная событийно-ориентированная модель Node.js обеспечивает эффективную обработку множества одновременных соединений без блокировки потока выполнения, что особенно актуально при построении REST API.

Express – минималистичный и гибкий веб-фреймворк для Node.js, предоставляющий базовые средства маршрутизации HTTP-запросов и обработки middleware. В отличие от Angular и подобных ему решений, Express не навязывает жёстких архитектурных шаблонов, предоставляя разработчику свободу в организации кода [8]. В рамках данного проекта реализован подход, близкий к MVC: маршруты (routes) делегируют обработку запросов контроллерам (controllers), которые взаимодействуют с базой данных через Prisma ORM.

Ключевым преимуществом связки Node.js + Express в контексте проекта является единый язык на всех уровнях стека: JavaScript используется как во фронтенд-части (React), так и в серверной логике, что существенно упрощает разработку и сопровождение кодовой базы.

На основании проведённого анализа для реализации проекта были выбраны React в качестве клиентского фреймворка и Node.js / Express в качестве серверной платформы. Данное сочетание обеспечивает единообразие технологического стека, высокую скорость разработки и достаточную масштабируемость для решаемой задачи.

					ПГУ1.090301.39.001 ПЗ	Лист
						28
Изм.	Лист	№ докум.	Подпись	Дата		

1.2.4 Выбор СУБД и ORM

Для надёжного хранения информации о пользователях, мероприятиях, регистрациях и жалобах необходима система управления базами данных, а для удобного взаимодействия с ней из серверного кода – инструмент объектно-реляционного отображения (ORM). В рамках данного раздела рассмотрены две реляционные СУБД – MySQL и PostgreSQL, – а также три ORM: Sequelize, TypeORM и Prisma.

1.2.4.1 MySQL

MySQL – реляционная СУБД с открытым исходным кодом, получившая широкое распространение в веб-разработке благодаря простоте развёртывания и высокой скорости выполнения типовых операций чтения. Она хорошо подходит для классических задач: хранения пользовательских данных, ведения журналов, работы с табличными структурами.

Вместе с тем MySQL уступает альтернативам по ряду характеристик: поддержка расширенных типов данных ограничена, полнотекстовый поиск менее развит, а ряд возможностей стандарта SQL реализован не в полной мере. Для приложения с разветвлёнными связями между сущностями (мероприятия, организаторы, возрастные категории, теги, пользователи) указанные ограничения приобретают существенное значение.

1.2.4.2 PostgreSQL

PostgreSQL – объектно-реляционная СУБД с открытым исходным кодом, отличающаяся высокой надёжностью, строгим соответствием стандарту SQL и широкими возможностями расширения. Система обеспечивает полную поддержку транзакций в соответствии с принципами ACID, располагает богатым набором встроенных типов данных, в том числе JSONB для хранения полуструктурированных данных, и поддерживает полнотекстовый поиск на уровне ядра СУБД.

Применительно к задаче хранения данных разрабатываемого приложения PostgreSQL демонстрирует очевидные преимущества перед MySQL. Схема базы данных проекта включает взаимосвязанные сущности – User, Event, Organizer, Tag, Age, UserEvent, Complaint – с множеством внешних ключей и составных ограничений уникальности. PostgreSQL обеспечивает целостность таких структур на уровне СУБД и эффективно обрабатывает сложные запросы с несколькими соединениями таблиц, что делает её предпочтительным выбором [9].

1.2.5 Выбор ORM

Для работы с базой данных из серверного кода на Node.js рассматривались три инструмента объектно-реляционного отображения: Sequelize, TypeORM и Prisma.

Sequelize – один из наиболее зрелых ORM для Node.js, существующий с 2011 года. Он поддерживает множество СУБД, в том числе PostgreSQL и MySQL, и предоставляет широкие возможности для определения моделей, ассоциаций и миграций. Вместе с тем Sequelize основан на подходе с экземплярами моделей, что усложняет работу с вложенными связями и нередко порождает избыточный по объёму и сложности код запросов. Отсутствие встроенной строгой типизации снижает надёжность при работе в JavaScript-проектах.

TypeORM – ORM с поддержкой TypeScript и декораторного стиля описания сущностей, близкого к подходам Hibernate (Java) и Entity Framework (.NET). TypeORM поддерживает паттерны Active Record и Data Mapper, обеспечивает гибкую настройку запросов через QueryBuilder. Однако инструмент имеет репутацию нестабильного API: в крупных проектах нередко фиксируются проблемы с производительностью миграций и непредсказуемым поведением при сложных связях. Для проекта на чистом JavaScript, без TypeScript, преимущества TypeORM оказываются менее выраженными.

Prisma – ORM нового поколения для экосистемы Node.js, существенно упрощающий взаимодействие с реляционными базами данных. Схема базы данных

					ПГУ1.090301.39.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		30

описывается в декларативном формате Prisma Schema, на основе которого автоматически генерируется типобезопасный клиентский API. Ключевым преимуществом Prisma Client является то, что он позволяет разработчику работать с данными в терминах объектов: запросы формируются через лаконичный API и всегда возвращают простые JavaScript-объекты без необходимости оперировать экземплярами моделей. Инструмент Prisma Migrate обеспечивает декларативное управление миграциями схемы с версионированием изменений.

Сравнительный анализ рассмотренных ORM представлен ниже. Sequelize обладает наибольшей зрелостью, однако уступает по удобству работы с типами и связанными объектами. TypeORM предлагает более современный подход, но демонстрирует нестабильность в ряде сценариев. Prisma выгодно отличается чистым декларативным синтаксисом, автогенерацией клиента и предсказуемым поведением запросов, что особенно важно при работе с разветвлённой схемой данных.

Ключевым преимуществом Prisma Client является то, что он позволяет разработчику работать с данными в терминах объектов, формируя запросы к базе данных через безопасный API и получая в ответ простые JavaScript-объекты без необходимости оперировать экземплярами моделей. Это существенно упрощает серверный код и снижает вероятность ошибок при работе со связанными сущностями. На основании изложенного для реализации проекта были выбраны СУБД PostgreSQL и ORM Prisma: данное сочетание обеспечивает надёжное хранение взаимосвязанных данных, декларативное управление миграциями схемы и удобную интеграцию с серверной частью на Node.js.

1.2.6 Выбор среды разработки

Среда разработки должна обеспечить комфорт написания кода, отладки и управления проектом. Рассмотрены следующие варианты:

1.2.6.1 WebStorm

WebStorm – коммерческая IDE от JetBrains, ориентированная на JavaScript и веб-технологии. Она предлагает сильные инструменты анализа кода, рефакторинг, автодополнение и удобную навигацию. Среди минусов – необходимость лицензии и относительно большие системные требования, что может быть избыточно для небольшого проекта.

1.2.6.2 IntelliJ IDEA

IntelliJ IDEA – универсальная IDE от JetBrains с поддержкой множества языков. Она включает в себя всё, что и WebStorm, плюс возможности для Java, Kotlin и т.д. Несмотря на мощь, IntelliJ IDEA также является коммерческой и требует лицензии для полного набора функций. Кроме того, для чисто JavaScript-проекта её функционал избыточен.

1.2.6.3 Visual Studio Code

Visual Studio Code – бесплатный, кроссплатформенный редактор кода от Microsoft. VS Code пользуется огромной популярностью среди веб-разработчиков. Основные преимущества:

- Кроссплатформенность: работает на Windows, macOS, Linux;
- Расширяемость: большой каталог расширений (для JavaScript, React, Node.js, Git, Docker и др.);
- Интеллектуальное автодополнение: IntelliSense, переход к определению, рефакторинг;
- Лёгковесность: быстрая работа на большинстве машин;
- Встроенный отладчик и Git: встроенная поддержка отладки JavaScript/Node.js и контроля версий.

Для разработки приложения была выбрана среда Visual Studio Code. Она предоставляет широкий набор инструментов и расширений для работы с JavaScript, React и Node.js, встроенную поддержку Git, интеллектуальное автодополнение на основе IntelliSense, а также средства отладки серверного и клиентского кода.

Немаловажным фактором является бесплатное распространение редактора, исключая необходимость приобретения лицензии. Совокупность указанных преимуществ делает Visual Studio Code оптимальным выбором среды разработки для данного проекта.

Таким образом, для разработки приложения для публикации мероприятий дополнительного образования был сформирован следующий технологический стек:

- язык программирования – JavaScript;
- клиентский фреймворк – React совместно с библиотекой компонентов Radix UI и утилитарным CSS-фреймворком Tailwind CSS;
- серверная платформа – Node.js с фреймворком Express;
- система управления базами данных – PostgreSQL;
- инструмент объектно-реляционного отображения – Prisma ORM;
- среда разработки – Visual Studio Code.

1.2.7 Выбор архитектурного решения приложения

При проектировании веб-приложения одним из ключевых решений является определение архитектурной организации системы. Среди наиболее распространённых подходов на сегодняшний день выделяются паттерны MVC, MVP и MVVM. Выбор конкретного паттерна обусловлен как характером решаемой задачи, так и применяемым технологическим стеком. Общей целью всех перечисленных шаблонов служит декомпозиция приложения на независимые слои, каждый из которых несёт свою зону ответственности.

1.2.7.1 Архитектурный паттерн MVC

MVC (Model-View-Controller) – один из базовых архитектурных шаблонов, предполагающий деление системы на три взаимодействующих уровня: модель данных (Model), слой отображения (View) и управляющий компонент (Controller). Ключевая идея состоит в том, что контроллер и представление зависят от модели, однако сама модель не имеет зависимостей от остальных частей системы.

Пользователь взаимодействует с элементами интерфейса, передавая тем самым управление контроллеру. Контроллер модифицирует состояние модели, после чего обновлённые данные передаются в представление для визуализации.

Характерные особенности данного паттерна:

- представление осведомлено о модели, поскольку контроллер явно передаёт её в слой отображения;
- представление не имеет прямой связи с контроллером и не содержит ссылок на него;
- между контроллером и представлением существует отношение «многие к одному»: один контроллер способен делегировать отрисовку различным представлениям в зависимости от контекста запроса.

1.2.7.2 Архитектурный паттерн MVP

MVP (Model-View-Presenter) концептуально близок к MVC, однако роль управляющего компонента здесь выполняет презентер. Данный шаблон широко применяется при создании настольных приложений с развитым пользовательским интерфейсом. Характерной чертой является двустороннее взаимодействие между представлением и презентером.

Отличительные свойства MVP:

- представление не имеет сведений о модели – актуализация слоя отображения выполняется презентером на основе полученных данных;
- представление и презентер взаимно осведомлены друг о друге и общаются через явные интерфейсы;
- между презентером и представлением устанавливается строгое отношение «один к одному».

1.2.7.3 Архитектурный паттерн MVVM

MVVM (Model-View-ViewModel) представляет собой дальнейшее развитие идей MVC. Его отличительными особенностями являются введение промежуточного компонента – модели представления (ViewModel) – и механизм

двустороннего связывания данных. Шаблон обеспечивает фундамент для построения клиентских приложений с высокой степенью отзывчивости, исключая при этом избыточное дублирование кода. Помимо этого, MVVM существенно упрощает модульное тестирование компонентов.

Основные характеристики паттерна:

- представление не осведомлено о модели данных;
- модель представления не содержит ссылок на конкретный слой отображения;
- представление отслеживает изменения в модели представления посредством механизмов реактивной подписки;
- отношение между моделью представления и представлением – «многие ко многим».

Наглядное сравнение схем взаимодействия компонентов в каждом из рассмотренных паттернов представлено на рисунке 12.

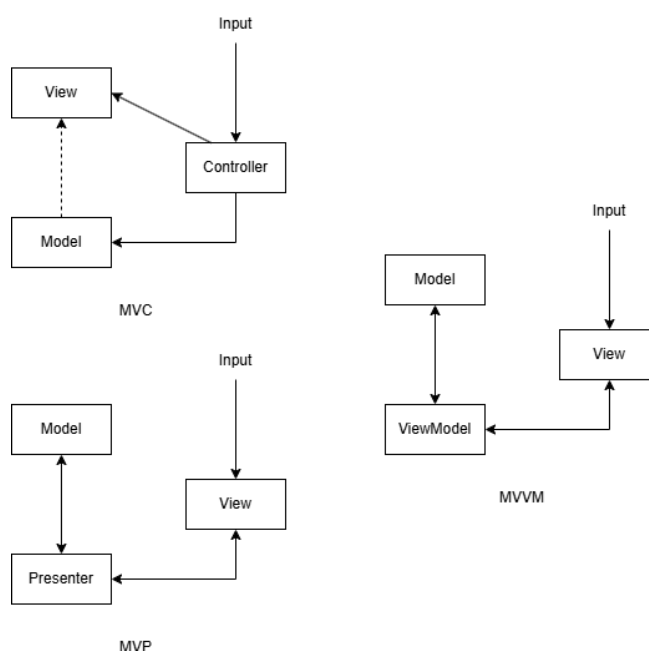


Рисунок 12 – Схемы взаимодействия компонентов в архитектурных паттернах MVC, MVP и MVVM

1.2.7.4 Обоснование выбора архитектурного паттерна

Для реализации данного приложения был выбран паттерн MVVM, поскольку его концепции органично соответствуют парадигме современных реактивных фреймворков – в первую очередь React, применяемого в качестве основы клиентской части. Современные фреймворки для разработки веб-приложений функционируют в парадигме реактивного программирования: они отслеживают изменения внутреннего состояния, оповещают все зависимые компоненты и автоматически перерисовывают интерфейс в ответ на эти изменения. Реализация подобного поведения опирается именно на принципы MVVM, асинхронные подходы, исключающие блокировку пользовательского взаимодействия, а также элементы функционального программирования.

MVVM и реактивное программирование не тождественны, однако преследуют единую цель – повышение отзывчивости и надёжности приложения с точки зрения конечного пользователя.

В результате архитектура приложения организована следующим образом.

Модель (Model). Уровень данных реализован средствами СУБД PostgreSQL с использованием ORM-инструмента Prisma. Схема данных охватывает сущности: пользователи (users), мероприятия (events), организаторы (organizers), категории (tags), возрастные группы (ages), записи на участие (user_event) и жалобы (complaints). Доступ к базе данных осуществляется исключительно через серверный уровень приложения (Node.js / Express), что исключает прямое обращение клиента к данным.

Представление (View). Слой отображения реализован на React 19 с применением TailwindCSS [5] и компонентной библиотеки Radix UI [6]. Страницы приложения – лента мероприятий, детальная страница события, личный кабинет пользователя, административная панель – реагируют на действия пользователя, подписываются на изменения состояния и реактивно обновляют отображаемые

данные. Навигация между представлениями обеспечивается средствами React Router DOM.

Модель представления (ViewModel). Роль ViewModel в приложении выполняют хуки React (useState, useEffect, useMemo) [12], инкапсулирующие логику получения данных, их преобразования и управления локальным состоянием компонентов. Взаимодействие с сервером осуществляется через единый API-клиент (apiRequest), который формирует HTTP-запросы к REST API бэкенда, автоматически прикрепляет JWT-токен авторизации из локального хранилища и обрабатывает ошибочные ответы. Все операции выполняются асинхронно, что позволяет интерфейсу оставаться интерактивным во время загрузки данных.

REST API (связующий уровень). Серверная часть построена на Express.js и предоставляет клиенту набор REST-эндпоинтов, сгруппированных по доменным областям: /api/auth, /api/events, /api/users и связанные маршруты. Каждый маршрут проходит через цепочку middleware-компонентов, обеспечивающих аутентификацию (JWT), авторизацию (проверка роли и права владения ресурсом), валидацию входных данных [13], ограничение частоты запросов и централизованную обработку ошибок. Бизнес-логика сосредоточена в контроллерах, работающих с базой данных исключительно через Prisma Client.

Таким образом, трёхуровневая архитектура приложения – клиентская часть (React), серверный API (Express / Node.js) и слой данных (PostgreSQL / Prisma) – реализует принципы паттерна MVVM: чёткое разделение ответственности, реактивное обновление интерфейса и асинхронное взаимодействие уровней через REST-интерфейс.

1.3 Реализация приложения

В данном разделе рассматривается процесс разработки клиент-серверного приложения, реализованного в рамках представленного проекта. Описываются ключевые составляющие системы, логика их взаимодействия, а также организация

					ПГУ1.090301.39.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		37

хранения данных. Основная задача данного раздела – подробно раскрыть особенности реализации приложения и принятые архитектурные решения.

1.3.1 Описание работы приложения

Структура системы и распределение ее компонентов по вычислительным узлам представлены на рисунке 13.

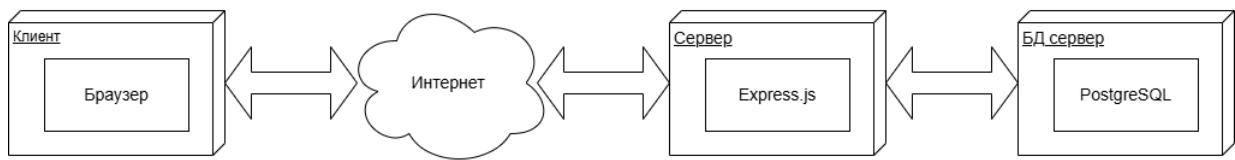


Рисунок 13 – UML-диаграмма развёртывания системы

Логика работы приложения построена на взаимодействии пользователя с клиентской частью, реализованной на React. В зависимости от выполняемого действия интерфейс формирует HTTP-запрос и передает его на сервер. Серверная часть, разработанная на Node.js и Express.js, принимает запрос, проходит через набор промежуточных обработчиков и выполняет необходимые проверки. Если в ходе обработки возникает ошибка, сервер возвращает соответствующее сообщение клиенту. При успешном выполнении запроса сервер обращается к базе данных PostgreSQL через Prisma ORM [10] и выполняет операции, связанные с получением, созданием, изменением или удалением данных. После завершения обработки клиент получает результат операции либо подтверждение ее выполнения.

Последовательность этих действий показана на рисунке 14.

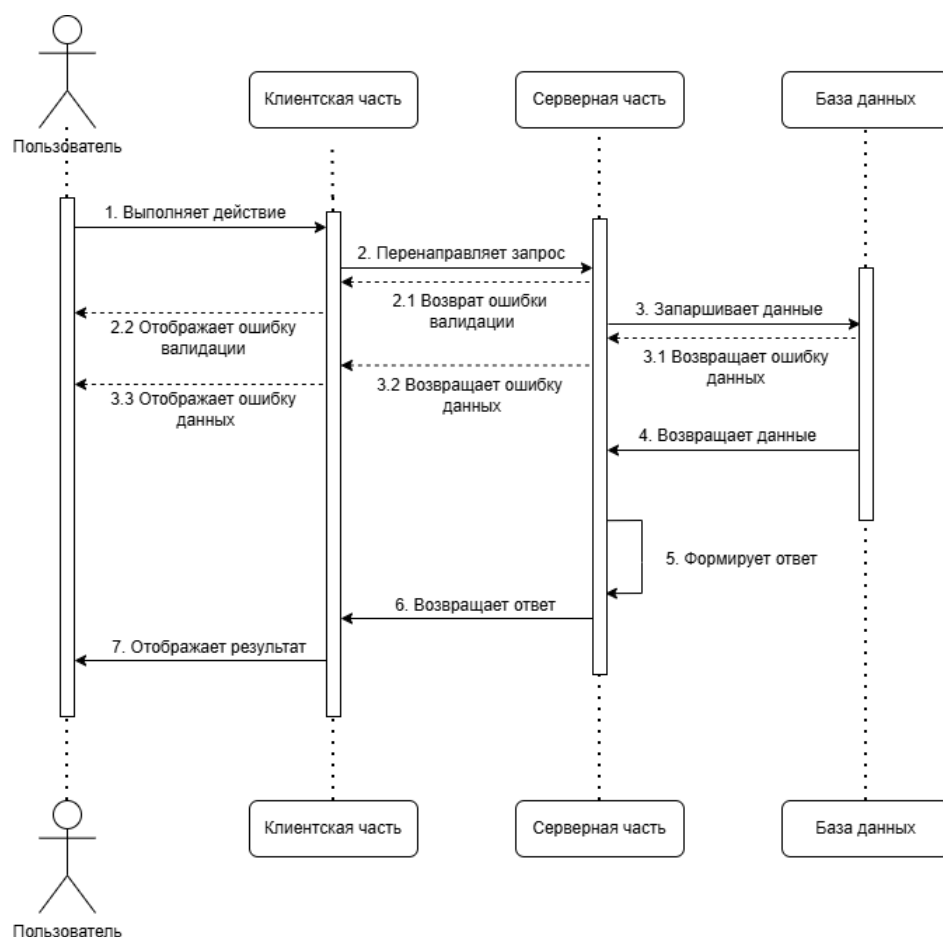


Рисунок 14 – UML-диаграмма последовательности

1.3.2 Структура базы данных

Для реализации клиент-серверного веб-приложения, предназначенного для работы с учетными записями пользователей и данными о мероприятиях, необходимо заранее определить состав сущностей и характер их взаимодействия в базе данных. Поскольку грамотно спроектированная структура хранения напрямую влияет на производительность, удобство сопровождения и дальнейшее развитие системы, в данном разделе рассматриваются основные таблицы, их атрибуты и связи, реализованные в рамках проекта.

Приложение должно обеспечивать возможность оперативного добавления новых мероприятий, а также управления сведениями о пользователях, организаторах, избранном и жалобах. Проведенный анализ показал, что в разрабатываемой системе целесообразно использовать следующие таблицы:

Изм.	Лист	№ докум.	Подпись	Дата

Таблица 1 (users) является центральной в схеме данных и хранит сведения обо всех зарегистрированных в системе пользователях. Каждая запись однозначно идентифицируется полем id типа UUID, что исключает коллизии при масштабировании системы. Поля first_name и second_name хранят имя и фамилию пользователя и используются при отображении профиля. Поле name содержит логин – уникальное имя пользователя внутри системы. Поле email служит основным идентификатором при аутентификации, а password хранит хешированное значение пароля, что обеспечивает безопасность учётных данных. Поле role определяет уровень доступа пользователя в системе и может принимать значения, соответствующие ролям неавторизованного пользователя, организатора и администратора. Внешний ключ organizer_id связывает пользователя с записью в таблице организаторов в том случае, если пользователь выступает официальным представителем учреждения.

Таблица 1 – users

Поле	Тип данных	Описание	Ключ
id	UUID	Идентификатор пользователя	РК
name	TEXT	Логин пользователя	
first_name	TEXT	Имя пользователя	
second_name	TEXT	Фамилия пользователя	
email	TEXT	Почта пользователя	
password	TEXT	Пароль	
organizer_id	UUID	Идентификатор организатора	FK
role	TEXT	Роль пользователя в системе	

Таблица 2 (events) является основной сущностью приложения и хранит полную информацию о каждом мероприятии. Поле id типа UUID обеспечивает уникальную идентификацию каждой записи. Поле name содержит название мероприятия, отображаемое в каталоге и на странице события. Поле description хранит развёрнутое описание, позволяющее пользователю ознакомиться с

содержанием мероприятия до регистрации. Поля date и time фиксируют дату и время проведения, а created – момент создания записи в системе, что необходимо для сортировки и аудита. Поле format указывает на форму проведения: очную, онлайн или гибридную. Поле address содержит место проведения для очных мероприятий. Поле count_members задаёт максимально допустимое число участников, что позволяет системе контролировать доступность мест при регистрации. Поле is_recurring указывает, является ли мероприятие разовым или повторяющимся. Поле author фиксирует, кем была создана запись о мероприятии. Внешние ключи tag_id, age_id и organizer_id связывают мероприятие соответственно с тематической категорией, возрастной группой и организатором. Булевы поля approved, active и deleted образуют систему статусов мероприятия: approved отражает факт прохождения модерации администратором, active – доступность мероприятия для просмотра в каталоге, deleted – признак мягкого удаления, позволяющий скрывать запись без физического удаления из базы данных.

Таблица 2 – events

Поле	Тип данных	Описание	Ключ
id	UUID	Идентификатор мероприятия	РК
name	TEXT	Название мероприятия	
is_recurring	BOOL	Мероприятие проходит один раз или повторяется	
format	TEXT	формат мероприятия онлайн или оффлайн	
description	TEXT	описание мероприятия	
created	DATE	дата создания записи	
date	DATE	дата проведения	
time	TIME	время проведения	
count_members	INT	максимальное кол-во участников	
address	TEXT	место проведения	

Поле	Тип данных	Описание	Ключ
author	TEXT	Кем создано мероприятие	
tag_id	TEXT	Идентификатор тега мероприятия	FK
age_id	TEXT	Идентификатор возрастной категории	FK
organizer_id	TEXT	Идентификатор организатора	FK
approved	BOOL	Одобрено ли мероприятие	
active	BOOL	Активно ли мероприятие	
deleted	BOOL	Удалено ли мероприятие	

Таблица 3 (user_event) реализует связь многие-ко-многим между пользователями и мероприятиями и является ключевым элементом для отслеживания взаимодействия пользователей с контентом платформы. Одна запись в этой таблице соответствует паре «конкретный пользователь – конкретное мероприятие». Поле member фиксирует факт регистрации пользователя на мероприятие, а поле favorites – добавление его в список избранного. Такая структура позволяет в одном запросе к базе данных получить одновременно статус участия и статус избранного для любой пары, не дублируя записи.

Таблица 3 – user_event

Поле	Тип данных	Описание	Ключ
id	UUID	Идентификатор записи	PK
id_user	TEXT	Идентификатор пользователя	FK
id_event	TEXT	Идентификатор мероприятия	FK
favorites	BOOL	Добавлено ли в избранное	
member	BOOL	Записан ли на мероприятие	

Таблица 4 (tags) содержит справочник тематических категорий, по которым классифицируются мероприятия. Каждая категория имеет уникальный

идентификатор – поле id типа UUID – и текстовое наименование, хранящееся в поле tag. Наличие отдельной таблицы для тегов обеспечивает гибкость: администратор может добавлять новые категории без изменения структуры базы данных, а фильтрация мероприятий по тематике выполняется через простое соединение таблиц.

Таблица 4 – tags

Поле	Тип данных	Описание	Ключ
id	UUID	Идентификатор тега	РК
tag	TEXT	Название тега	

Таблица 5 представляет собой справочник возрастных групп, к которым может быть отнесено каждое мероприятие. Уникальный идентификатор хранится в поле id, а сама возрастная категория – в поле age_category. Отдельная таблица для возрастных категорий позволяет унифицировать их перечень по всей системе и использовать их в качестве фильтра при поиске, обеспечивая подбор мероприятий, соответствующих возрасту участника.

Таблица 5 – ages

Поле	Тип данных	Описание	Ключ
id	UUID	Идентификатор категории	РК
age_category	TEXT	Возрастная категория	

Таблица 6 organizers хранит сведения об официальных учреждениях и организациях, от имени которых публикуются мероприятия. Каждый организатор идентифицируется полем id типа UUID, а его наименование хранится в поле name. Связь с таблицей users позволяет привязать конкретного пользователя к учреждению-организатору, а связь с таблицей events – отобразить на странице мероприятия информацию о проводящей его организации. Вынесение

организаторов в отдельную таблицу исключает дублирование данных об учреждении в каждой записи о мероприятии.

Таблица 6 – organizers

Поле	Тип данных	Описание	Ключ
id	UUID	Идентификатор организатора	РК
name	TEXT	Название учреждения организатора	

Диаграмма изображенная на рисунке 15 отражает все шесть таблиц и связи между ними. Центральное место занимает таблица events, к которой через внешние ключи подключены таблицы tags, ages и organizers – это позволяет классифицировать каждое мероприятие по тематике, возрастной группе и организатору. Таблица users связана с таблицей organizers через поле organizer_id, что обеспечивает привязку пользователя к представляемому им учреждению. Связь между пользователями и мероприятиями реализована через промежуточную таблицу user_event, которая фиксирует как факт регистрации на мероприятие, так и добавление его в избранное. Подобная схема обеспечивает нормализацию данных, исключает избыточность и позволяет эффективно выполнять запросы любой сложности – от получения списка мероприятий по фильтру до формирования личного кабинета пользователя с полной историей его активности на платформе. Диаграмма отражает все шесть таблиц и связи между ними. Центральное место занимает таблица events, к которой через внешние ключи подключены таблицы tags, ages и organizers – это позволяет классифицировать каждое мероприятие по тематике, возрастной группе и организатору. Таблица users связана с таблицей organizers через поле organizer_id, что обеспечивает привязку пользователя к представляемому им учреждению. Связь между пользователями и мероприятиями реализована через промежуточную таблицу user_event, которая фиксирует как факт регистрации на мероприятие, так и добавление его в избранное.

Подобная схема обеспечивает нормализацию данных, исключает избыточность и позволяет эффективно выполнять запросы любой сложности – от получения списка мероприятий по фильтру до формирования личного кабинета пользователя с полной историей его активности на платформе.

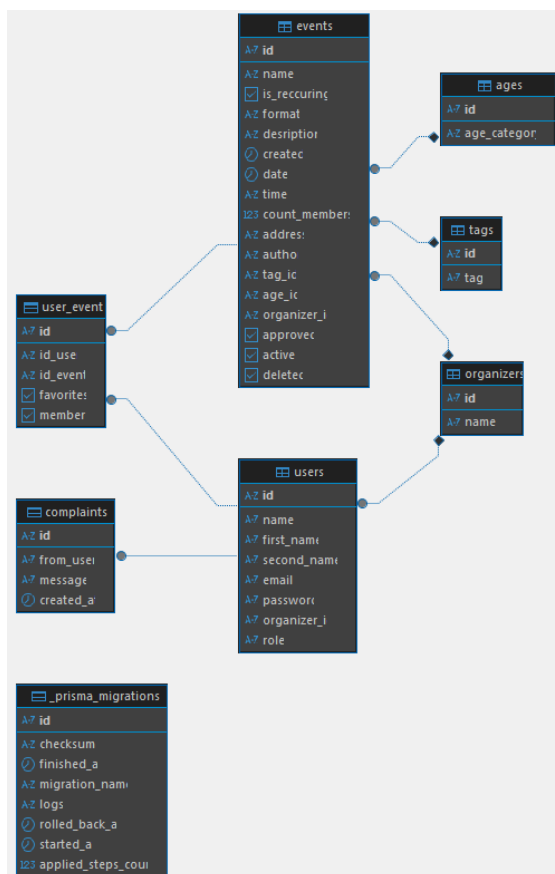


Рисунок 15 – Структура базы данных

1.3.3 Описание компонентов

В соответствии с выбранной трёхуровневой архитектурой клиент-серверного приложения, система разделена на три самостоятельных функциональных слоя:

- клиентский слой (Presentation Layer);
- серверный слой (Business Logic Layer);
- слой хранения данных (Data Layer).

Подобное разграничение позволяет независимо разрабатывать, тестировать и модифицировать каждый из уровней, не затрагивая логику остальных. UML-

диаграмма компонентов приложения и взаимосвязи между уровнями представлены на рисунке 16

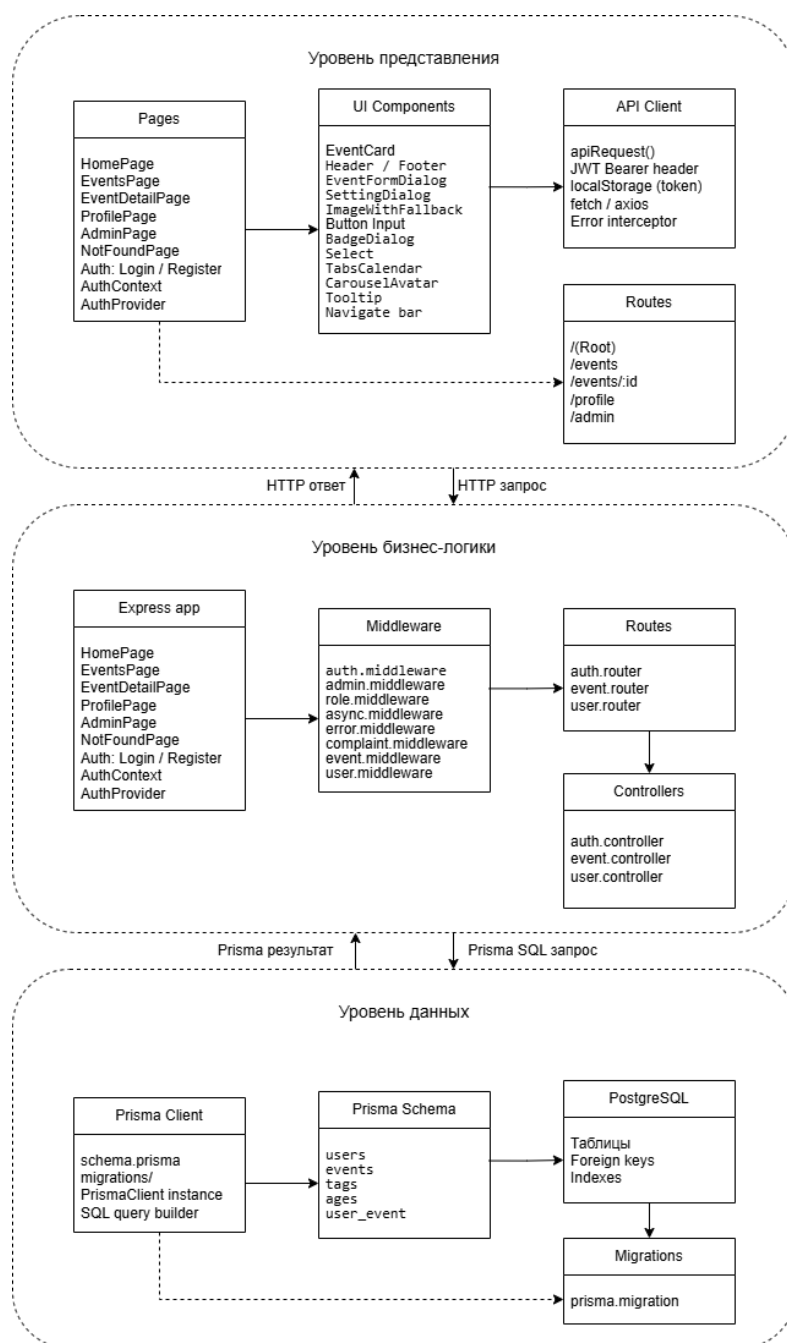


Рисунок 16 – UML-диаграмма компонентов

Клиентский слой отвечает за формирование пользовательского интерфейса и организацию взаимодействия с пользователем. Именно на этом уровне осуществляется отображение перечня мероприятий, просмотр детальной

информации о конкретном событии, управление профилем, выполнение административных функций, а также процедуры входа и регистрации. Клиентская часть реализована на библиотеке React с использованием инструмента сборки Vite [4]. Приложение построено исключительно на функциональных компонентах с применением React Hooks, что соответствует современным практикам разработки на React. Каждый компонент инкапсулирует собственную логику отображения и поведения, что обеспечивает его независимость и возможность повторного применения в различных частях интерфейса.

Страницы приложения сосредоточены в каталоге pages и представляют собой верхнеуровневые компоненты, каждый из которых соответствует отдельному разделу сервиса:

- HomePage – корневая страница, отображающая общую информацию о платформе и предоставляющая точку входа для навигации по разделам;
- EventsPage – страница каталога мероприятий, предоставляющая пользователю список доступных событий с возможностью фильтрации по тегам, возрастным категориям и другим параметрам;
- EventDetailPage – страница детального просмотра выбранного события, отображающая полное описание, дату, место проведения, изображение и количество доступных мест;
- ProfilePage – личный кабинет авторизованного пользователя, где доступны сведения об аккаунте, список записанных и добавленных в избранное мероприятий, а также форма редактирования персональных данных;
- AdminPage – страница административного управления, предназначенная для модерации новых событий, управления пользователями и просмотра поступивших жалоб;
- NotFoundPage – страница, отображаемая при обращении к несуществующему маршруту и информирующая пользователя об ошибке навигации.

Для обеспечения доступа к защищённым разделам приложения реализована система аутентификации. Диалоговые окна `LoginDialog` и `RegisterDialog` предоставляют пользователю формы входа и создания нового аккаунта соответственно. Они открываются поверх текущей страницы, не прерывая навигацию, что улучшает пользовательский опыт. Управление состоянием авторизации на уровне всего приложения организовано посредством механизма `React Context`. `AuthContext` определяет контекст, хранящий информацию о текущем сеансе, а `AuthProvider` оборачивает корневой компонент приложения, обеспечивая глобальную доступность этих данных из любого вложенного компонента без необходимости явной передачи через `props`. Пользовательский хук `useAuth` предоставляет удобный программный интерфейс для получения данных контекста аутентификации, скрывая детали реализации от потребляющих компонентов.

Библиотека переиспользуемых UI-компонентов формирует визуальную основу всего интерфейса. Компоненты можно разделить на несколько функциональных групп.

Компоненты отображения контента:

- `EventCard` – карточка мероприятия с кратким описанием, датой, тегами и кнопкой действия, используемая для вывода событий в списке;
- `ImageWithFallback` – компонент изображения с автоматической подстановкой заглушки при ошибке загрузки, обеспечивающий визуальную стабильность интерфейса;
- `Avatar` – отображение аватара пользователя с поддержкой резервного изображения при отсутствии загруженной фотографии;
- `Badge` – небольшая метка для отображения тегов, возрастных ограничений и статусов мероприятий.

Компоненты навигации и структуры страницы:

– Header и корневой компонент Root – задают общую структуру страницы: навигационную панель, шапку с элементами управления сеансом и область основного контента;

– Tabs – компонент переключения между вкладками, применяемый на страницах с несколькими режимами отображения;

– Carousel – горизонтальная прокручиваемая лента для отображения нескольких элементов в ограниченном пространстве.

Компоненты взаимодействия и ввода:

– Button – кнопки с различными вариантами визуального оформления и состояниями (активное, неактивное, загрузка);

– Input – поля ввода текста с поддержкой валидации и отображения ошибок;

– Select – выпадающий список для выбора значения из заданного набора вариантов;

– Calendar – компонент выбора даты, применяемый при создании мероприятий и настройке фильтров;

– Tooltip – всплывающие подсказки, добавляемые к элементам интерфейса для пояснения их назначения.

Компоненты диалогов:

– Dialog – базовый компонент модального окна, используемый как основа для специализированных диалогов;

– EventFormDialog – модальное окно для создания и редактирования мероприятий, доступное организаторам и администраторам;

– SettingDialog – диалог настроек пользовательского профиля с формой изменения персональных данных.

Навигация между разделами приложения организована посредством React Router. Конфигурация маршрутов определена в файле routes.jsx, где каждому пути сопоставлен соответствующий компонент страницы:

– / – корневой маршрут, отображающий HomePage;

- /events – страница каталога мероприятий EventsPage;
- /events/:id – динамический маршрут с сегментом-идентификатором, открывающий EventDetailPage для конкретного события и позволяющий формировать постоянные прямые ссылки на отдельные мероприятия;
- /profile – защищённый маршрут личного кабинета, доступный только авторизованным пользователям;
- /admin – защищённый административный раздел, доступный пользователям с соответствующим уровнем прав.

При обращении к любому незарегистрированному пути система перенаправляет пользователя на компонент NotFoundPage.

Взаимодействие клиентского слоя с серверной частью осуществляется через централизованную функцию apiRequest(), которая инкапсулирует логику отправки HTTP-запросов к REST API. [14] Функция принимает метод запроса, адрес конечной точки и тело запроса, формируя унифицированный вызов к серверу. Все запросы передаются в формате JSON, ответы сервера также десериализуются из JSON перед передачей в вызывающий код. [11] Организация клиент-серверного взаимодействия предполагает следующий порядок операций:

- клиентский компонент инициирует вызов apiRequest() с указанием метода, маршрута и данных;
- функция извлекает токен доступа из localStorage и прикрепляет его к исходящему запросу в заголовке Authorization в формате Bearer;
- сформированный HTTP-запрос направляется на соответствующую конечную точку серверного API;
- сервер обрабатывает запрос, выполняет необходимые операции и возвращает ответ в формате JSON;
- при получении ответа с кодом ошибки функция формирует исключение, которое перехватывается вызывающим компонентом для информирования пользователя.

Такой подход позволяет серверу идентифицировать и верифицировать пользователя без хранения состояния сеанса на стороне сервера, что соответствует принципам REST-архитектуры. Обмен данными между клиентским и серверным уровнями производится исключительно посредством HTTP-запросов и ответов в формате JSON, что обеспечивает строгое разделение слоёв и независимость их реализации друг от друга.

Точкой входа является файл `index.js`, в котором инициализируется экземпляр Express-приложения и выполняется настройка глобальных компонентов. Файл `index.js` представлен на листинге 1.

Листинг 1 – Точка входа сервера (`index.js`)

```
const app = express();
const PORT = process.env.PORT || 5000;

app.use(cors({ origin: 'http://localhost:5173', credentials:
true }));
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

app.use('/api/auth', authRoutes);
app.use('/api/events', eventRoutes);
app.use('/api/users', userRoutes);
app.use('/api/', userEventsRoutes);
app.use('/api', complaintsRoutes);

app.use((req, res) => res.status(404).json({ message: 'Маршрут не
найден' }));
app.use(errorMiddleware);

app.listen(PORT, () => console.log(`Сервер запущен на
http://localhost:\${PORT}`));
```

В `index.js` выполняются следующие действия:

- настройка CORS с разрешением запросов от клиентского адреса `http://localhost:5173` с передачей учётных данных (`credentials: true`);
- подключение встроенных парсеров тела запроса: `express.json()` для обработки JSON и `express.urlencoded()` для форм;
- монтирование пяти групп маршрутов по соответствующим префиксам;

– регистрация обработчика несуществующих маршрутов, возвращающего статус 404;

– подключение глобального обработчика ошибок `errorMiddleware` последним в цепочке, что является требованием Express для перехвата ошибок.

Система промежуточной обработки запросов является ключевым элементом обеспечения безопасности и корректности работы серверного слоя. Каждый `middleware` решает строго одну задачу и может подключаться к отдельным маршрутам независимо:

– `authMiddleware` – извлекает JWT из заголовка `Authorization`, верифицирует его с помощью `jwt.verify` и секрета `JWT_ACCESS_SECRET`, находит пользователя в базе данных по идентификатору из токена, проверяет флаг блокировки и записывает объект `{id, role}` в `req.user` для использования последующими обработчиками [13];

– `adminOnly` – проверяет наличие `req.user` и соответствие поля `role` значению `'admin'`, при несоответствии возвращает статус 403;

– `checkRole(...roles)` – обобщённая версия проверки роли, принимает произвольный список допустимых ролей через `rest`-параметр и возвращает функцию-`middleware`;

– `isEventOwner` – загружает мероприятие по идентификатору из параметров маршрута и сравнивает поле `author` с `req.user.id`, запрещая изменение чужих событий;

– `eventExists` и `eventOwnerOrAdmin` – два варианта проверки существования мероприятия: первый лишь убеждается в наличии записи, второй дополнительно допускает к операции либо владельца, либо администратора;

– `complaintExists`, `complaintOwnerOrAdmin`, `complaintAdminOnly` – аналогичный набор для сущности жалобы;

– checkAlreadyJoined (flood.middleware) – предотвращает повторную запись пользователя на одно и то же мероприятие, проверяя наличие записи в таблице user_event с флагом member: true;

– pagination – извлекает из строки запроса параметры page и limit, вычисляет значения skip и take для Prisma и записывает их в req.pagination;

– validate(schema) – принимает схему Zod, вызывает safeParse для тела запроса и при наличии ошибок возвращает статус 400 с массивом нарушений;

– asyncHandler(fn) – оборачивает асинхронную функцию-обработчик в Promise.resolve().catch(next), перенаправляя необработанные отклонения в глобальный обработчик ошибок;

– errorMiddleware – глобальный обработчик ошибок Express, принимает четыре аргумента (err, req, res, next) и возвращает JSON с полем message и статусом из err.status либо 500 по умолчанию;

– logger – выводит в консоль метод, URL и временную метку каждого входящего запроса.

Маршрутизация разделена на пять независимых роутеров:

– auth.router – регистрирует четыре POST-маршрута: /register, /login, /logout и /refresh, связывая каждый с соответствующей функцией auth.controller;

– event.router – обслуживает операции над мероприятиями: получение списка, детальный просмотр, создание, редактирование, удаление и модерацию;

– user-events.router – отвечает за пользовательские действия с мероприятиями: запись на событие и добавление в избранное;

– user.router – предоставляет маршруты для работы с профилем пользователя;

– complaints.router – регистрирует четыре маршрута для операций над жалобами: POST /complaints, GET /complaints, PATCH /complaints/:id и DELETE /complaints/:id.

Бизнес-логика каждой группы маршрутов сосредоточена в соответствующих контроллерах. Рассмотрим наиболее значимые из них.

Функция `events` из `event.controller`, представленная на листинге 2, выбирает из базы данных только активные, не удалённые и прошедшие модерацию события. С помощью агрегации `_count` она одновременно подсчитывает количество участников – записей в таблице `user_event` с флагом `member: true` – и возвращает это значение в поле `members` результирующего объекта.

Листинг 2. Функция `events` (`event.controller.js`)

```
export const events = async (req, res) => {
  const events = await prisma.event.findMany({
    where: { active: true, deleted: false, approved: true },
    include: {
      _count: {
        select: { user_event: { where: { member: true } } },
      },
    },
  });
  const result = events.map(({ _count, ...event }) => ({
    ...event,
    members: _count.user_event,
  }));
  res.json(result);
};
```

Функция `newEvent`, представленная на листинге 3, создаёт новое мероприятие. Независимо от переданных данных поле `approved` принудительно устанавливается в `false`, что обязывает каждое новое событие проходить модерацию администратором. Идентификатор автора извлекается из `req.user.id`, сформированного `authMiddleware`. Числовые поля `tag_id`, `age_id`, `organizer_id` и `count_members` приводятся к типу `Number` во избежание ошибок типизации Prisma. Дата преобразуется в объект `Date` с явным указанием нулевого времени в UTC.

Листинг 3. Функция `newEvent` (`event.controller.js`)

```
export const newEvent = async (req, res) => {
  try {
    const { date, tag_id, age_id, organizer_id, count_members,
      approved, ...rest } = req.body;
```

```

const event = await prisma.event.create({
  data: {
    ...rest,
    time: rest.time?.slice(0, 5) ?? null,
    date: date ? new Date(date + 'T00:00:00.000Z') : null,
    tag_id: tag_id ? Number(tag_id) : null,
    age_id: age_id ? Number(age_id) : null,
    organizer_id: organizer_id ? Number(organizer_id) : null,
    count_members: count_members ? Number(count_members) :
null,
    author: req.user.id,
    approved: false,
  },
});
res.json(event);
} catch (e) {
  res.status(500).json({ message: e.message });
}
};

```

Функция join, представленная на листинге 4, реализует запись пользователя на мероприятие. Используется составной уникальный ключ id_user_id_event для поиска существующей связи. Если запись уже существует (например, пользователь ранее добавил событие в избранное), выполняется обновление флага member в true. Если записи нет – создаётся новая. Такой подход позволяет совмещать статусы «участник» и «в избранном» в одной строке таблицы.

Листинг 4. Функция join (event.controller.js)

```

export const join = async (req, res) => {
  const id_event = parseInt(req.params.id);
  const id_user = req.user.id;
  const existing = await prisma.userEvent.findUnique({
    where: { id_user_id_event: { id_user, id_event } },
  });
  const record = existing
    ? await prisma.userEvent.update({
        where: { id_user_id_event: { id_user, id_event } },
        data: { member: true },
      })
    : await prisma.userEvent.create({
        data: { id_user, id_event, member: true },
      });
  res.json(record);
};

```

Функция editMe из user.controller, представленная на листинге 5, позволяет пользователю обновить собственные данные профиля, включая смену пароля. При наличии поля newPassword в теле запроса выполняется дополнительная верификация: текущий пароль сравнивается с хранимым хэшем посредством bcrypt.compare [15]. Только при успешной проверке новый пароль хэшируется и включается в объект обновления. Ошибка с кодом P2002, генерируемая Prisma при нарушении ограничения уникальности, перехватывается отдельно и возвращается клиенту с понятным сообщением.

Листинг 5. Функция editMe (user.controller.js)

```
export const editMe = async (req, res) => {
  try {
    const { first_name, second_name, name, email,
currentPassword, newPassword } = req.body;
    if (newPassword) {
      const user = await prisma.user.findUnique({ where: { id:
req.user.id } });
      const valid = await bcrypt.compare(currentPassword,
user.password);
      if (!valid) return res.status(400).json({ message: 'Wrong
current password' });
    }
    const data = {
      ...(first_name !== undefined && { first_name }),
      ...(second_name !== undefined && { second_name }),
      ...(name !== undefined && { name }),
      ...(email !== undefined && { email }),
      ...(newPassword && { password: await
bcrypt.hash(newPassword, 10) }),
    };
    const updated = await prisma.user.update({
      where: { id: req.user.id },
      select: { id: true, name: true, first_name: true,
second_name: true, email: true, role: true },
      data,
    });
    res.json(updated);
  } catch (e) {
    if (e.code === 'P2002') return res.status(400).json({
message: 'Email already taken' });
    res.status(500).json({ message: e.message });
  }
}
```

} ;

Слой хранения данных построен на основе реляционной СУБД PostgreSQL с применением ORM Prisma. Схема базы данных описана в файле `schema.prisma`, история изменений структуры фиксируется в директории `migrations/` в виде последовательных SQL-файлов. Взаимодействие сервера с базой данных осуществляется исключительно через singleton-экземпляр `PrismaClient`, что исключает создание избыточных подключений.

Схема данных включает шесть моделей:

- `User` – хранит идентификатор (`id`, автоинкремент), системное имя (`name`), имя и фамилию (`first_name`, `second_name`), уникальный адрес электронной почты (`email`), хэш пароля (`password`), роль (`role`, по умолчанию `'user'`), флаг блокировки (`blocked`, по умолчанию `false`) и внешний ключ `organizer_id`;

- `Organizer` – содержит идентификатор и уникальное название организатора (`name`);

- `Event` – включает уникальное название (`name`), формат проведения (`format`), описание (`description`), дату (`date`) и время (`time`), максимальное число участников (`count_members`), адрес (`address`), идентификатор автора (`author`), внешние ключи `tag_id`, `age_id` и `organizer_id`, а также три булевых флага: `approved` (статус модерации, по умолчанию `false`), `active` (активность, по умолчанию `true`) и `deleted` (мягкое удаление, по умолчанию `false`);

- `Tag` и `Age` – справочные модели, содержащие соответственно поля `tag` и `age_category` для классификации мероприятий;

- `UserEvent` – связующая модель, объединяющая пользователя и мероприятие через поля `id_user` и `id_event`, дополненные двумя независимыми флагами: `member` (участие) и `favorites` (избранное). Составной уникальный индекс `@@unique([id_user, id_event])` гарантирует наличие не более одной связи между конкретным пользователем и мероприятием, позволяя при этом хранить оба статуса в одной строке.

Для реализации адаптивной вёрстки в клиентской части приложения применён CSS-фреймворк Tailwind CSS. Его ключевая особенность состоит в атомарном подходе к стилизации: вместо написания отдельных CSS-правил разработчик применяет к элементам заранее определённые утилитарные классы непосредственно в JSX-разметке [16]. Tailwind CSS реализует парадигму «mobile-first», при которой стили без префикса действуют на экранах любой ширины начиная с мобильных, а стили с префиксами sm:, lg:, 2xl: и другими активируются только при достижении соответствующей ширины экрана. Это позволяет проектировать интерфейс последовательно от наименьшего размера к наибольшему.

Адаптивная сетка мероприятий на странице EventsPage представлена на листинге 6. На мобильных устройствах карточки выстраиваются в одну колонку, при ширине от 1024px (брейкпоинт lg) сетка переключается на двухколоночный режим, а на экранах от 1536px (брейкпоинт 2xl) отображаются три колонки.

Листинг 6. Адаптивная сетка мероприятий EventsPage.jsx

```
<div className="grid gap-4 lg:grid-cols-2 2xl:grid-cols-3">
  {filteredEvents.map((event) => (
    <EventCard key={event.id} event={event} tagsMap={tagsMap}
agesMap={agesMap} />
  ))}
</div>
```

Боковая панель фильтров на странице EventsPage также имеет адаптивное поведение: на мобильных экранах она полностью скрыта с помощью класса hidden, а на экранах от 1024px отображается посредством lg:block. Общий контейнер страницы использует flex-раскладку: панель фильтров фиксируется по ширине (w-[280px] flex-shrink-0), а основная область мероприятий (flex-1) адаптивно заполняет оставшееся пространство.

Листинг 7. Адаптивная боковая панель фильтров (EventsPage.jsx)

```
<div className="max-w-7xl mx-auto px-6 py-8">
```

```

<div className="flex gap-8">
  <aside className="w-[280px] flex-shrink-0 hidden lg:block">
    <div className="bg-card border border-border rounded-lg p-
6 space-y-6">
      ...
    </div>
  </aside>
  <div className="flex-1 space-y-4">
    ...
  </div>
</div>
</div>

```

На листинге 8 представлен блок деталей карточки мероприятия, расположенный внутри компонента ProfilePage. Сетка деталей события (дата, время, формат) на мобильных устройствах отображается в одну колонку, а начиная с брейкпоинта sm переключается на трёхколоночный режим посредством класса sm:grid-cols-3. Текстовые элементы используют классы text-xs для подписей и font-medium для значений, что создаёт визуальную иерархию без применения жёстких размеров.

Листинг 8. Адаптивная сетка деталей карточки мероприятия (ProfilePage.jsx)

```

<div className="grid sm:grid-cols-3 gap-4 text-sm pt-2">
  <div>
    <p className="text-muted-foreground text-xs mb-1">Дата</p>
    <p className="text-foreground font-medium">{date}</p>
  </div>
  <div>
    <p className="text-muted-foreground text-xs mb-1">Время</p>
    <p className="text-foreground font-medium">{time}</p>
  </div>
  <div>
    <p className="text-muted-foreground text-xs mb-1">Формат</p>
    <p className="text-foreground font-medium">{format}</p>
  </div>
</div>

```

Применение CSS-переменных через семантические токены Tailwind (bg-card, border-border, text-foreground, text-muted-foreground, bg-primary/10 и другие) обеспечивает единообразие цветовой схемы по всему приложению [16]. Все цветовые значения определены в одном месте конфигурации и автоматически

применяются ко всем компонентам, что упрощает поддержку и потенциальное переключение тем оформления. Классы переходов `transition-colors` на интерактивных элементах и состояние `hover:border-primary/50` на карточках мероприятий обеспечивают плавные визуальные отклики без написания дополнительного CSS-кода.

Таким образом, сочетание Tailwind CSS с компонентной архитектурой React позволило реализовать адаптивный интерфейс, корректно отображающийся на устройствах с различной шириной экрана – от мобильных телефонов до широкоформатных мониторов – без дублирования стилей и с минимальным объёмом дополнительного кода.

1.3.4 Основные алгоритмы взаимодействия клиента и сервера

Для наглядного рассмотрения алгоритмов взаимодействия клиента и сервера были разработаны UML-диаграммы деятельности, отображающие последовательности действий между пользователем, клиентским приложением и сервером.

Первым рассматриваемым алгоритмом является процесс регистрации и авторизации. Для использования всех функций приложения необходимо пройти регистрацию и авторизоваться. Если неавторизованный пользователь пытается воспользоваться защищённой функцией – например, записаться на мероприятие или перейти в личный кабинет – клиентское приложение отображает диалоговое окно авторизации поверх текущей страницы.

При регистрации пользователь заполняет поля имени, фамилии, адреса электронной почты и пароля. Введённые данные отправляются на сервер, который проверяет уникальность email в базе данных. Если такой адрес уже зарегистрирован, клиенту возвращается соответствующая ошибка. В противном случае пароль хэшируется с помощью bcrypt, автоматически формируется системное имя пользователя, и запись сохраняется в базе данных.

При авторизации сервер проверяет существование пользователя по email, сравнивает введенный пароль с хранимым хэшем и в случае успеха возвращает пару токенов: access-токен со сроком действия 15 минут и refresh-токен сроком 7 дней. Access-токен сохраняется в localStorage и прикрепляется к каждому последующему запросу в заголовке Authorization. Если введенные данные некорректны, пользователю отображается сообщение об ошибке. Последовательность описанных действий представлена на рисунке 17.

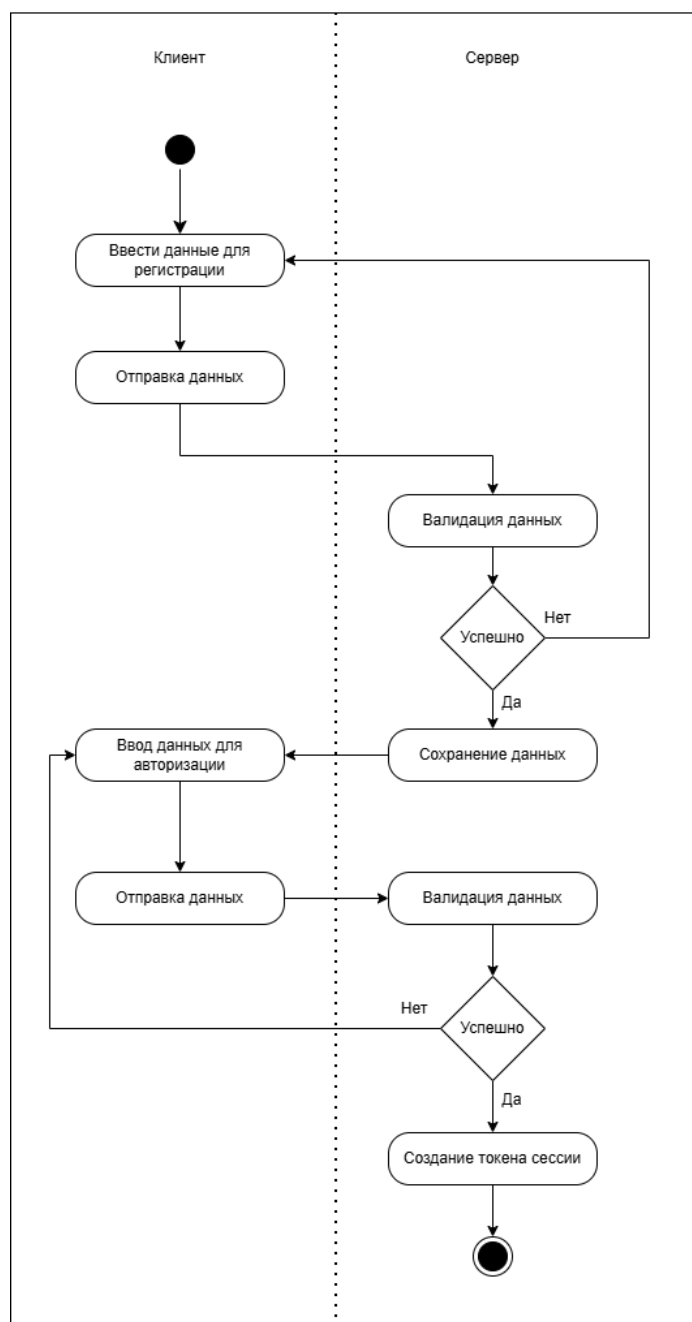


Рисунок 17 – UML-диаграмма деятельности регистрации и авторизации

Следующим рассматривается алгоритм создания и редактирования мероприятия. Создание мероприятия доступно авторизованным пользователям через соответствующую форму. Пользователь заполняет обязательные поля – название, формат, дату – и при необходимости указывает описание, адрес, тег, возрастную категорию, организатора и максимальное число участников. Заполненные данные передаются на сервер, который принудительно устанавливает поле `approved` в значение `false` независимо от переданных данных, что обязывает каждое новое мероприятие проходить модерацию администратором. До момента одобрения мероприятие не отображается в общем каталоге.

Редактирование мероприятия доступно его автору или администратору. Сервер проверяет принадлежность записи через `isEventOwner` либо роль пользователя, и только при успешной проверке применяет изменения. Последовательность описанных действий представлена на рисунке 18.

					ПГУ1.090301.39.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		62

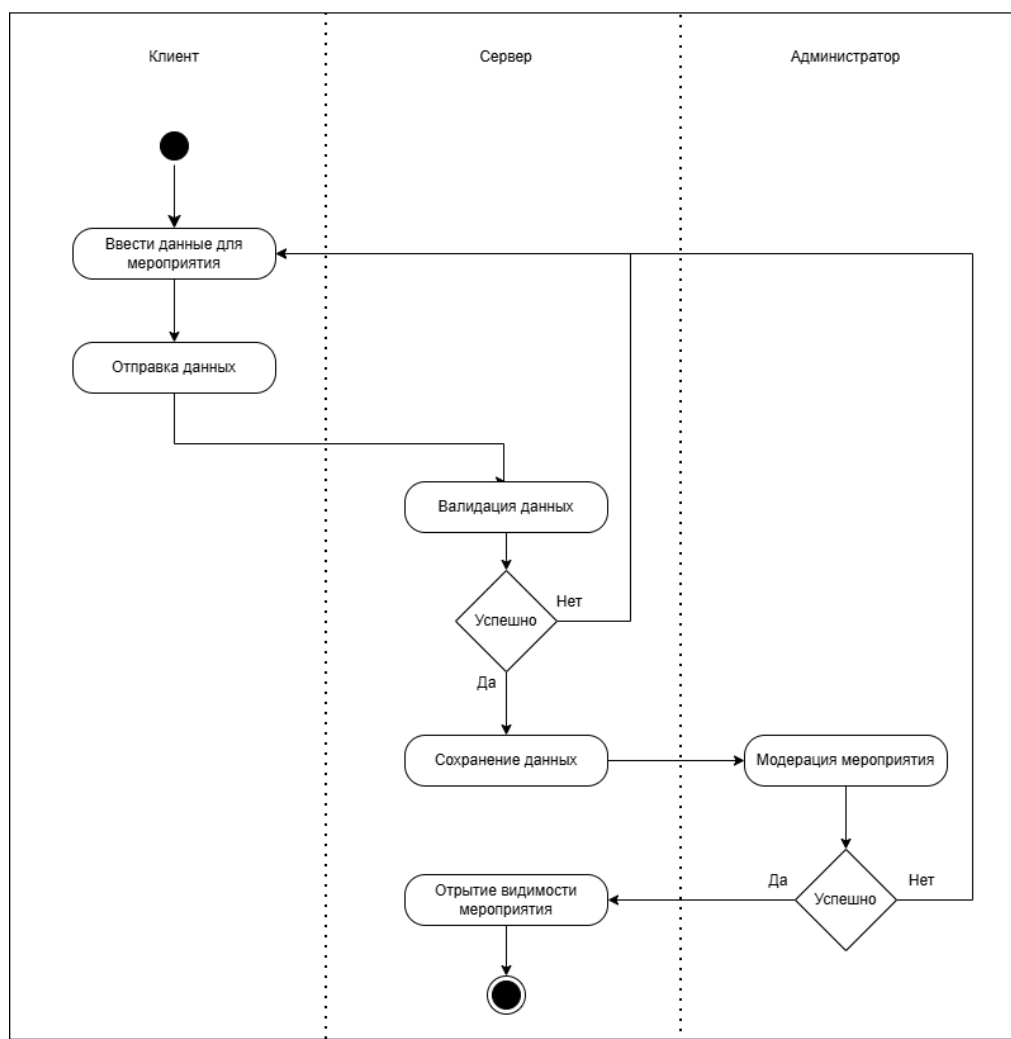


Рисунок 18 – UML-диаграмма деятельности создания и редактирования мероприятия

Следующим рассматривается алгоритм взаимодействия пользователя с мероприятием – записи на событие и добавления его в избранное. Оба действия доступны только авторизованным пользователям. При выполнении любого из них сервер обращается к таблице `user_event` по составному ключу `id_user + id_event`. Если запись уже существует – выполняется обновление соответствующего флага (`member` или `favorites`), если нет – создаётся новая. Такой подход позволяет хранить оба статуса в одной строке таблицы.

При записи на мероприятие дополнительно срабатывает `checkAlreadyJoined` (`flood.middleware`), предотвращающий повторную регистрацию пользователя на

одно и то же событие. Отмена записи и удаление из избранного выполняются симметрично – через DELETE-запрос с обнулением соответствующего флага. Последовательность описанных действий представлена на рисунке 19.

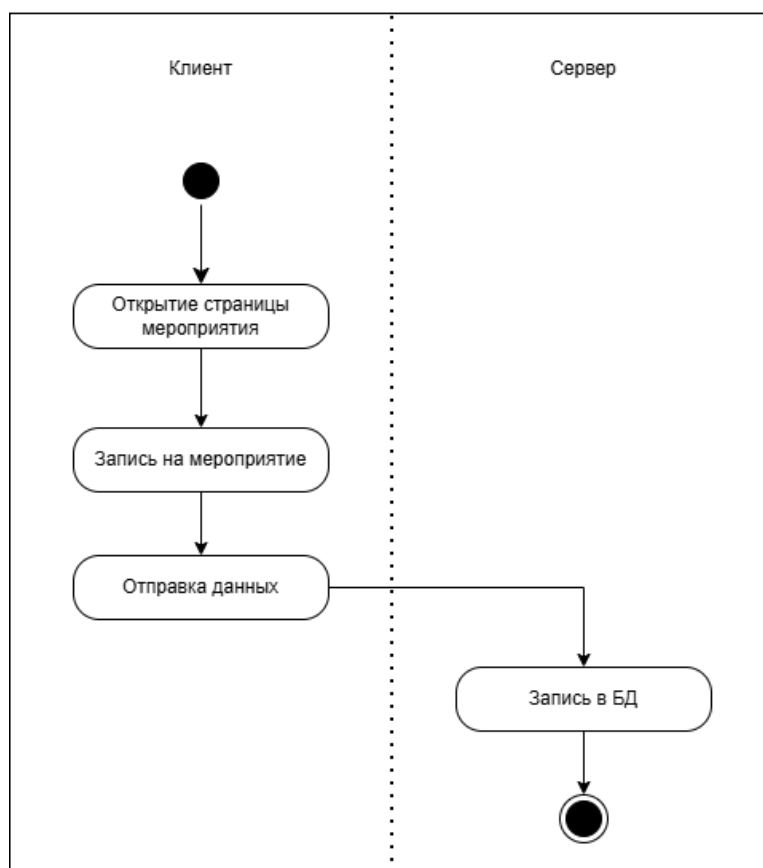


Рисунок 19 – UML-диаграмма деятельности записи пользователя на мероприятие

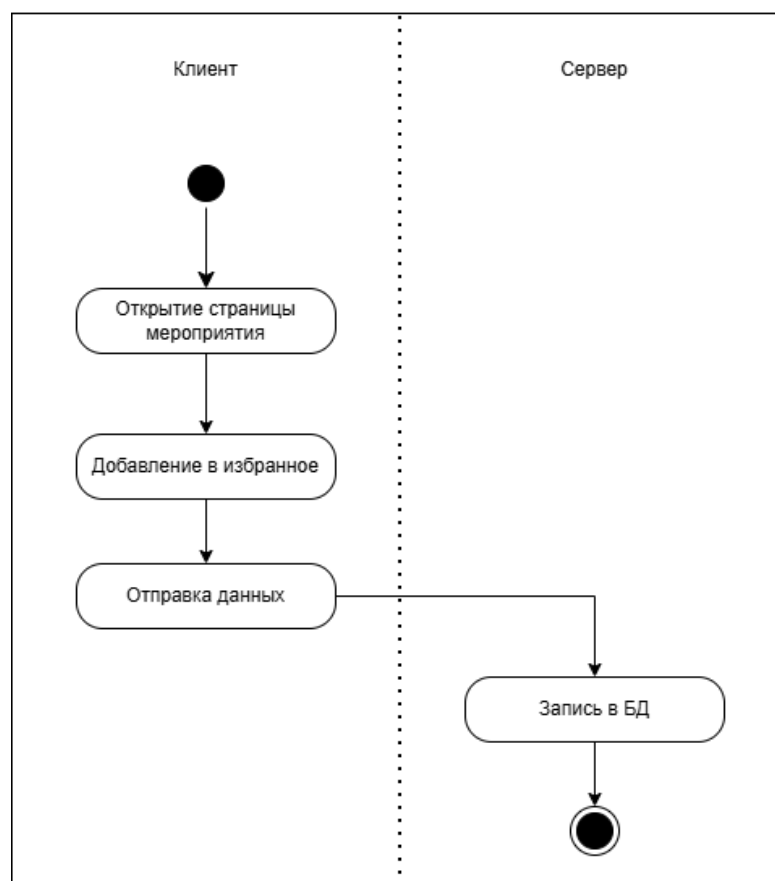


Рисунок 20 – UML-диаграмма деятельности добавления мероприятия в избранное

Алгоритм поиска и фильтрации мероприятий реализован на стороне клиента. Пользователь задаёт параметры фильтрации – тег, возрастную категорию или текстовый поиск по названию – и клиентское приложение немедленно применяет их к уже загруженному списку мероприятий без повторного обращения к серверу. В каталоге отображаются только активные, не удалённые и прошедшие модерацию события. Последовательность описанных действий представлена на рисунке 21.

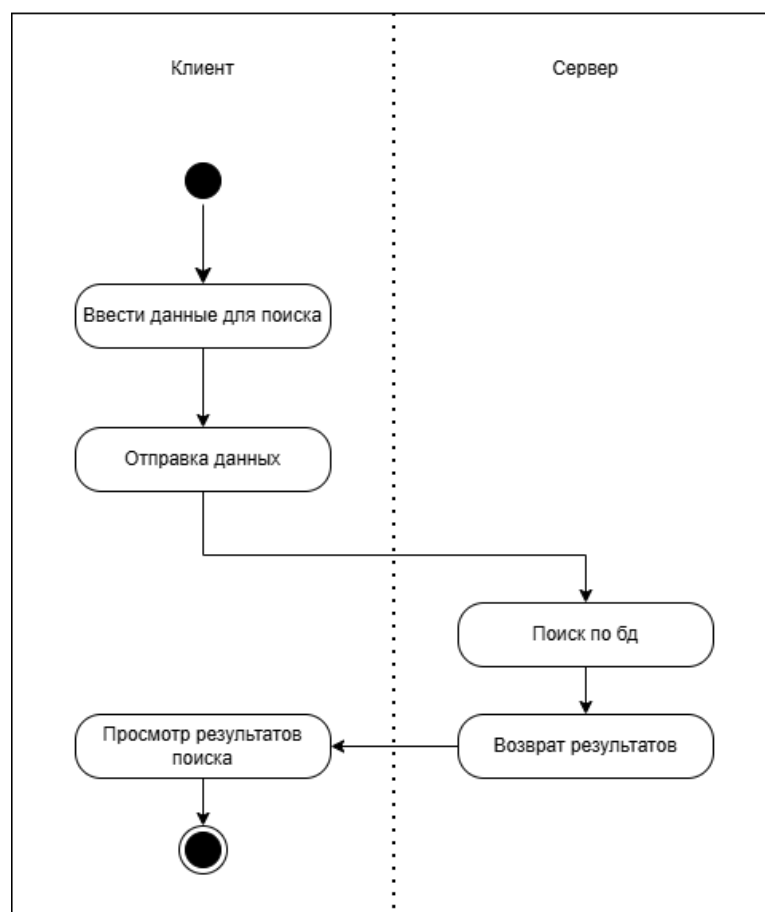


Рисунок 21 – UML-диаграмма деятельности поиска и фильтрации мероприятий

Алгоритм изменения профиля позволяет авторизованному пользователю обновить персональные данные: имя, фамилию, системное имя и адрес электронной почты. При смене пароля выполняется дополнительная проверка: сервер сравнивает введенный текущий пароль с хранимым хэшем посредством `bcrypt.compare`, и только при успешной верификации новый пароль хэшируется и сохраняется. При нарушении ограничения уникальности email сервер возвращает клиенту понятное сообщение об ошибке. Последовательность описанных действий представлена на рисунке 22.

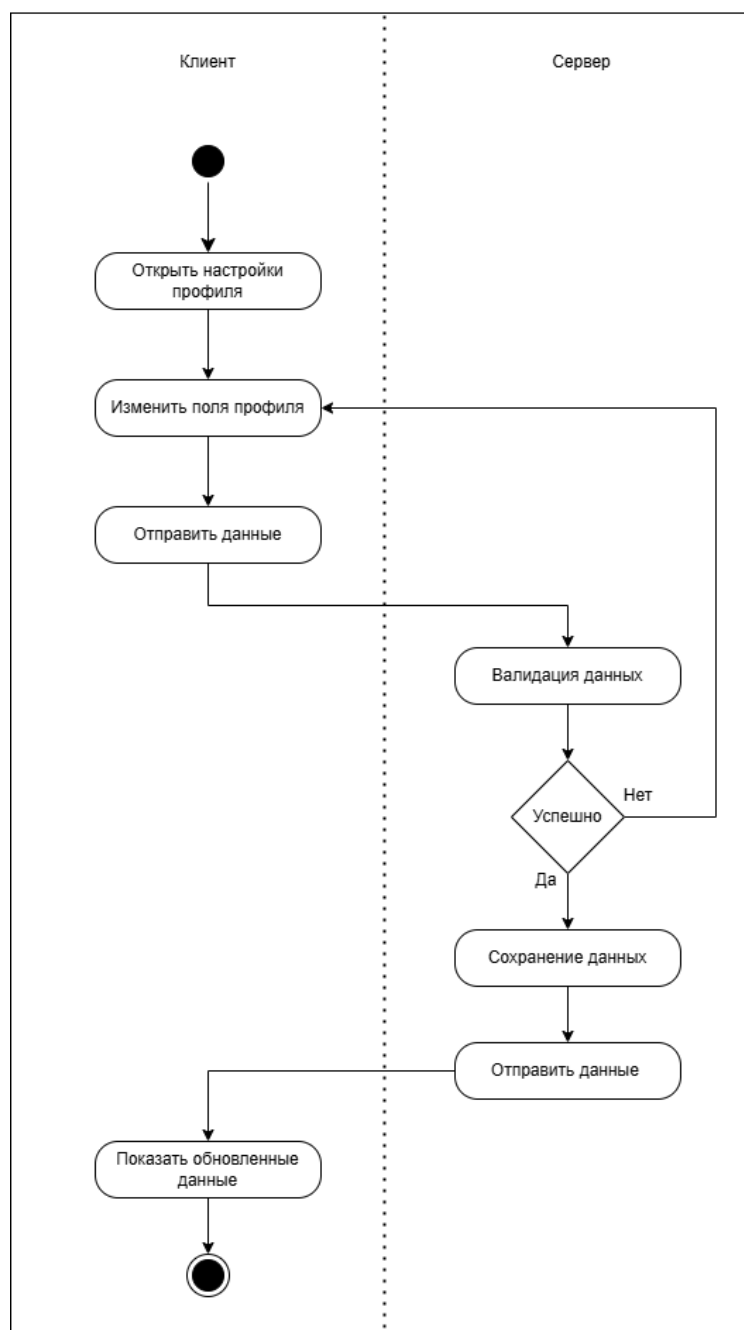


Рисунок 22 – UML-диаграмма деятельности изменения профиля

Заключительным рассматривается алгоритм административных функций. В данном сценарии взаимодействуют два участника – администратор и система. Администратор имеет доступ к специальному разделу AdminPage, недоступному обычным пользователям. В рамках этого раздела реализованы три группы операций:

– модерация мероприятий – администратор просматривает список событий с флагом approved: false и принимает решение об одобрении или удалении каждого из них;

– удаление мероприятий – выполняется мягкое удаление через установку флага deleted: true, что исключает событие из каталога без физического удаления записи из базы данных;

– управление пользователями – администратор может изменить роль пользователя или заблокировать его аккаунт; заблокированный пользователь получает отказ в доступе на уровне authMiddleware при каждом последующем запросе.

Последовательность описанных действий представлена на рисунке 23.

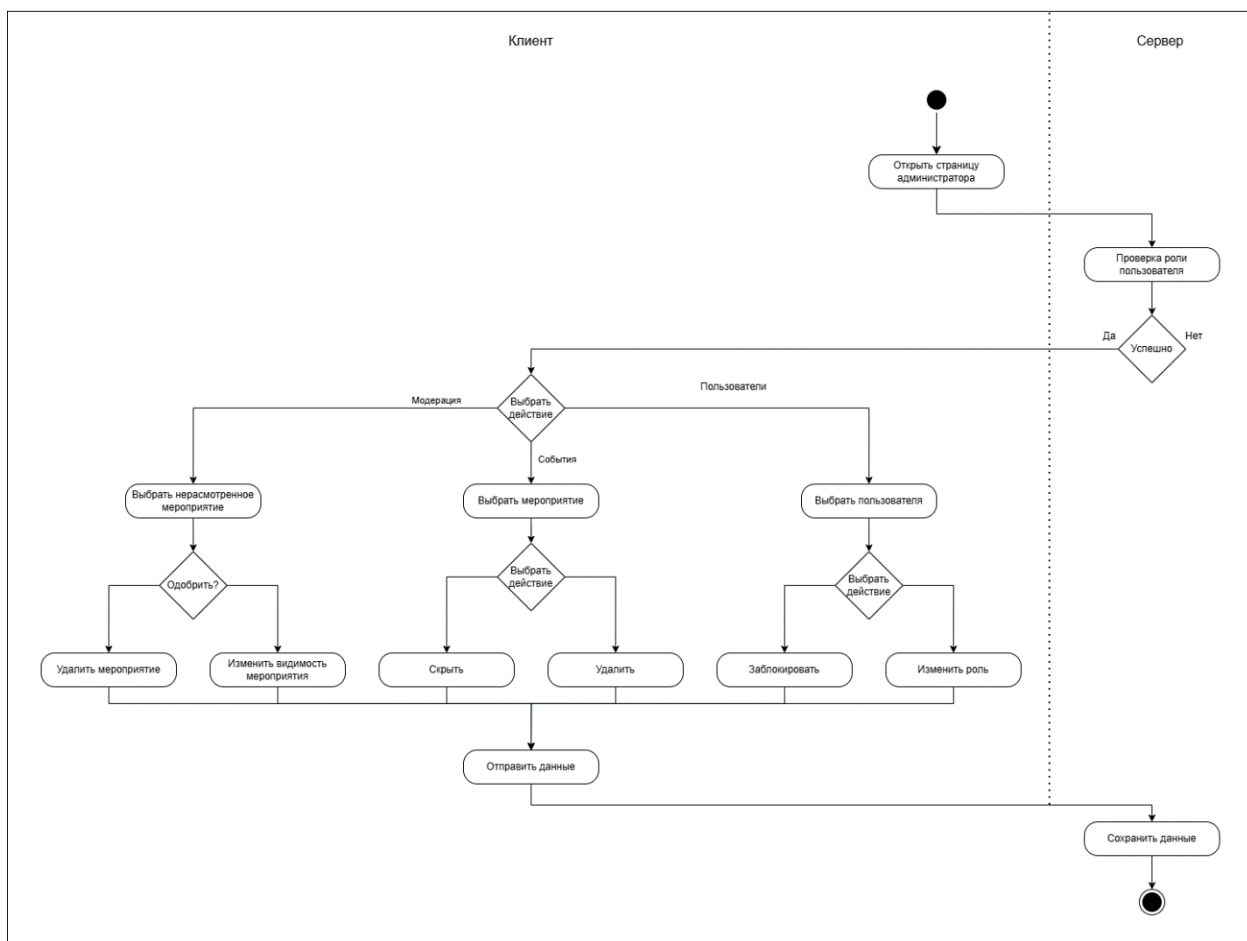


Рисунок 23 – UML-диаграмма деятельности административных функций

1.4 Тестирование

Для проверки работоспособности клиент-серверного приложения «Платформа для поиска и записи на мероприятия» необходимо провести его тестирование. Для тестирования были составлены тест-кейсы. Тест-кейсы составлены для следующих действий:

- регистрация пользователя;
- авторизация пользователя;
- просмотр каталога мероприятий и фильтрация;
- запись на мероприятие;
- добавление мероприятия в избранное;
- создание мероприятия;
- изменение профиля пользователя;
- модерация мероприятий администратором;
- управление пользователями администратором.

1.4.1 Регистрация пользователя

В таблице 7 представлен тест-кейс регистрации пользователя.

Таблица 7– тест-кейс регистрации пользователя

Шаги	Действия	Данные	Ожидаемый результат	Фактический результат
1	Нажать на кнопку «Зарегистрироваться»		На экране отобразилось модальное окно регистрации	На экране отобразилось модальное окно регистрации
2	Оставить поля пустыми, нажать кнопку «Зарегистрироваться»		Поля выделены красным, отображены сообщения об обязательных полях	Поля выделены красным, отображены сообщения об обязательных полях
3	Ввести email, уже существующий в системе, заполнить остальные поля,	Email: test@test.ru, Пароль: test, Имя: Иван,	Отображено сообщение об ошибке «Email уже занят»	Отображено сообщение об ошибке «Email уже занят»

Шаги	Действия	Данные	Ожидаемый результат	Фактический результат
	нажать «Зарегистрироваться»	Фамилия: Иванов		
4	Ввести корректные данные нового пользователя, нажать «Зарегистрироваться»	Email: newuser@test.ru, Пароль: test, Имя: Иван, Фамилия: Иван	Модальное окно закрылось, пользователь авторизован, кнопка «Войти» заменена на логин пользователя	Модальное окно закрылось, пользователь авторизован, кнопка «Войти» заменена на логин пользователя

1.4.2 Авторизация пользователя

В таблице 8 представлен тест-кейс авторизации пользователя.

Таблица 8 – тест-кейс авторизации пользователя

Шаги	Действия	Данные	Ожидаемый результат	Фактический результат
1	Нажать на кнопку «Войти»		На экране отобразилось модальное окно для авторизации пользователя	На экране отобразилось модальное окно для авторизации пользователя
2	Ввести в поля ввода тестовые данные, нажать кнопку «Войти»	Email: test@test.ru Пароль: 111111	Поля очистились и стали выделены красным, под ними отобразилось сообщение неправильный email или пароль	Поля очистились и стали выделены красным, под ними отобразилось сообщение неправильный email или пароль
3	Ввести в поля ввода тестовые данные, нажать кнопку «Войти»	Email: test@test.ru Пароль: test	Модальное окно закрылось, кнопка «Войти» была заменена на логин пользователя, при нажатии на которое открывается профиль пользователя	Модальное окно закрылось, кнопка «Войти» была заменена на логин пользователя, при нажатии на которое открывается профиль пользователя

1.4.3 Просмотр каталога мероприятий и фильтрация

В таблице 9 представлен тест-кейс просмотра каталога мероприятий и применения фильтров по тегу и возрастной категории.

Таблица 9 – тест-кейс просмотра каталога мероприятий

Шаги	Действия	Данные	Ожидаемый результат	Фактический результат
1	Нажать на кнопку «Войти»		На экране отобразилось модальное окно для авторизации пользователя	На экране отобразилось модальное окно для авторизации пользователя
2	Ввести в поля ввода тестовые данные, нажать кнопку «Войти»	Email: test@test.ru Пароль: 111111	Поля очистились и стали выделены красным, под ними отобразилось сообщение неправильный email или пароль	Поля очистились и стали выделены красным, под ними отобразилось сообщение неправильный email или пароль
3	Ввести в поля ввода тестовые данные, нажать кнопку «Войти»	Email: test@test.ru Пароль: test	Модальное окно закрылось, кнопка «Войти» была заменена на логин пользователя, при нажатии на которое открывается профиль пользователя	Модальное окно закрылось, кнопка «Войти» была заменена на логин пользователя, при нажатии на которое открывается профиль пользователя

1.4.4 Запись на мероприятие и добавление в избранное

В таблице 10 представлен тест-кейс записи пользователя на мероприятие. В таблице 11 представлен тест-кейс добавления мероприятия в избранное.

Таблица 10 – тест-кейс записи пользователя на мероприятие

Шаги	Действия	Данные	Ожидаемый результат	Фактический результат
1	Нажать кнопку «Записаться» на странице мероприятия		Кнопка изменила состояние на «Отменить запись», счётчик участников увеличился на 1	Кнопка изменила состояние на «Отменить запись», счётчик участников увеличился на 1
2	Перейти в личный кабинет, открыть вкладку «Мои записи»		Мероприятие отображается в списке записей	Мероприятие отображается в списке записей
3	Вернуться на страницу		Кнопка вернулась в состояние «Записаться»,	Кнопка вернулась в состояние «Записаться»,

Шаги	Действия	Данные	Ожидаемый результат	Фактический результат
	мероприятия, нажать «Отменить запись»		счётчик участников уменьшился на 1	счётчик участников уменьшился на 1
4	Выйти из аккаунта, попытаться нажать кнопку «Записаться»		Отобразилось модальное окно «Вход в личный кабинет»	Отобразилось модальное окно «Вход в личный кабинет»

Таблица 11 – тест-кейс добавления мероприятия в избранное

Шаги	Действия	Данные	Ожидаемый результат	Фактический результат
1	Нажать кнопку «В избранное» на странице мероприятия		Кнопка изменила состояние на «В избранном»	Кнопка изменила состояние на «В избранном»
2	Перейти в личный кабинет, открыть вкладку «Избранное»		Мероприятие отображается в списке избранных	Мероприятие отображается в списке избранных
3	Вернуться на страницу мероприятия, нажать «В избранном»		Кнопка вернулась в состояние «В избранное», мероприятие удалено из избранного	Кнопка вернулась в состояние «В избранное», мероприятие удалено из избранного

1.4.5 Создание мероприятия

В таблице 12 представлен тест-кейс создания нового мероприятия авторизованным пользователем.

Таблица 12 – тест-кейс создания нового мероприятия авторизованным пользователем

Шаги	Действия	Данные	Ожидаемый результат	Фактический результат
1	Открыть страницу каталога мероприятий		Отображается список активных одобренных мероприятий	Отображается список активных одобренных мероприятий
2	Ввести текст в поле «Поиск по названию, описанию или организатору...»	«основы»	Список мероприятий обновился, отображаются только события,	Список мероприятий обновился, отображаются только события,

Шаги	Действия	Данные	Ожидаемый результат	Фактический результат
			содержащие введенное слово	содержащие введенное слово
3	Установить флажок у одной из тем в боковой панели фильтров	Тема: «Биология»	Список дополнительно отфильтрован по выбранной теме	Список дополнительно отфильтрован по выбранной теме
4	Установить флажок у возрастной категории	Возраст: «14+»	Список дополнительно отфильтрован по возрастной категории	Список дополнительно отфильтрован по возрастной категории

1.4.6 Изменение профиля пользователя

В таблице 13 представлен тест-кейс изменения персональных данных пользователя, включая смену пароля.

Таблица 13 – тест-кейс изменения персональных данных пользователя

Шаги	Действия	Данные	Ожидаемый результат	Фактический результат
1	Перейти на вкладку «Настройки» в личном кабинете		Открылось модальное окно «Настройки профиля» с заполненными текущими данными	Открылось модальное окно «Настройки профиля» с заполненными текущими данными
2	Изменить поля «Имя» и «Фамилия», нажать «Сохранить»	Имя: Алексей, Фамилия: Смирнов	Кнопка изменила текст на «Сохранение...», данные профиля обновились	Кнопка изменила текст на «Сохранение...», данные профиля обновились
3	Ввести в поле «Email» уже занятый адрес, нажать «Сохранить»	Email: existing@test.ru	Под полем email отображено сообщение «Этот email уже занят»	Под полем email отображено сообщение «Этот email уже занят»
4	Заполнить поле «Текущий пароль» неверным значением, ввести новый пароль, нажать «Сохранить»	Текущий пароль: wrongpass, Новый пароль: newpass123, Повторите: newpass123	Под полем «Текущий пароль» отображено сообщение «Неверный текущий пароль»	Под полем «Текущий пароль» отображено сообщение «Неверный текущий пароль»

1.4.7 Функционал администратора

В таблице 14 представлен тест-кейс модерации мероприятий администратором. В таблице 15 представлен тест-кейс управления пользователями.

Таблица 14 – тест-кейс модерации мероприятий администратором

Шаги	Действия	Данные	Ожидаемый результат	Фактический результат
1	Открыть вкладку «Модерация» на странице AdminPage		Отображается список мероприятий, ожидающих одобрения, с указанием их количества в заголовке вкладки	Отображается список мероприятий, ожидающих одобрения, с указанием их количества в заголовке вкладки
2	Нажать кнопку «Одобрить» у одного из мероприятий		Мероприятие исчезло из списка на модерации, стало доступно в общем каталоге	Мероприятие исчезло из списка на модерации, стало доступно в общем каталоге
3	Нажать кнопку «Удалить» у другого мероприятия из списка на модерации		Мероприятие удалено из списка, в общем каталоге не отображается	Мероприятие удалено из списка, в общем каталоге не отображается
4	Нажать кнопку «Снять» у уже одобренного мероприятия		Мероприятие скрыто из каталога, статус одобрения сброшен	Мероприятие скрыто из каталога, статус одобрения сброшен

Таблица 15 – тест-кейс управления пользователями

Шаги	Действия	Данные	Ожидаемый результат	Фактический результат
1	Открыть вкладку «Пользователи» на странице AdminPage			Отображается список всех пользователей с их ролями и статусами
2	Изменить роль пользователя через выпадающий список		Роль: «Администратор»	Роль пользователя обновлена в списке
3	Нажать кнопку «Заблокировать» у выбранного пользователя			Кнопка изменила текст на «Уже заблокирован» и стала неактивной,

Шаги	Действия	Данные	Ожидаемый результат	Фактический результат
				статус пользователя изменился
4	Попытаться войти в систему от имени заблокированного пользователя		Email: blocked@test.ru, Пароль: test123	Вход не выполнен, отображено сообщение об ошибке

1.5 Руководство пользователя

1.5.1 Результат работы приложения

Платформа образовательных мероприятий представляет собой многостраничное веб-приложение. Шапка сайта присутствует на всех страницах и содержит логотип с переходом на главную, навигационные ссылки, кнопку «Войти» и кнопку создания мероприятия для авторизованных пользователей.

Главная страница состоит из четырёх секций. В верхней – приветственный блок с кратким описанием платформы и кнопками «Найти мероприятие» и «Узнать больше». Ниже расположен блок с четырьмя карточками преимуществ: «Удобный поиск», «Простая регистрация», «Для организаторов» и «Надёжность». Третья секция – «Популярные мероприятия» – отображает актуальные образовательные программы. Завершает страницу призыв к действию с кнопкой «Посмотреть все мероприятия». Главная страница представлена на рисунке 24.

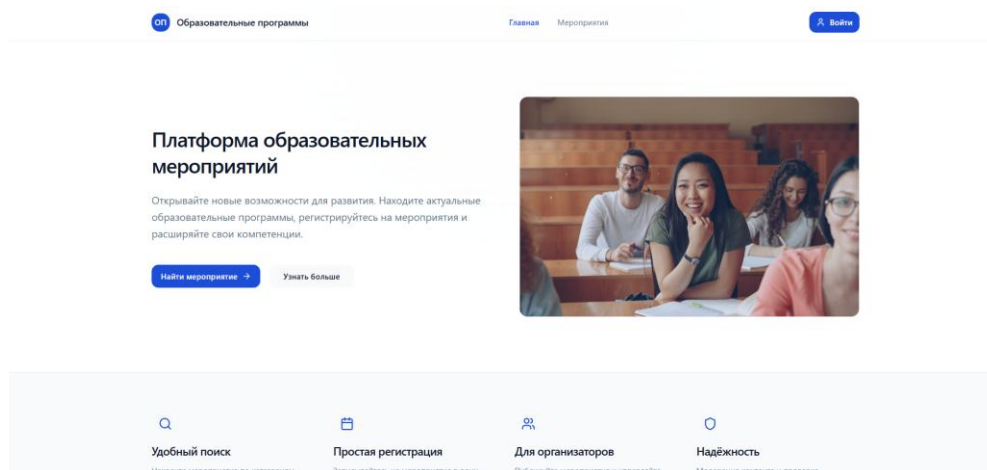


Рисунок 24 – Главная страница

Страница каталога мероприятий разделена на два блока. В левой части расположена боковая панель фильтров с полем текстового поиска, выбором тематических категорий, возрастных ограничений и формата проведения, а также кнопкой «Сбросить фильтры». В основной области отображаются карточки мероприятий, каждая из которых содержит название, формат (Онлайн / Оффлайн / Гибрид), возрастную категорию и количество доступных мест. Фильтрация применяется мгновенно без перезагрузки страницы. Каталог мероприятий представлен на рисунке 25.

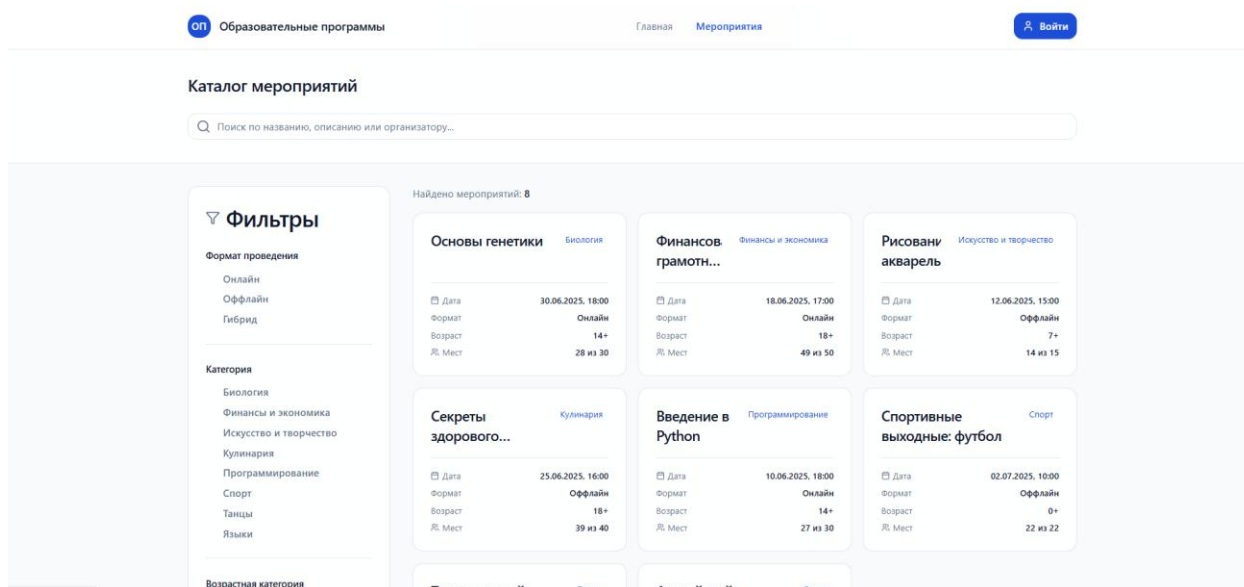


Рисунок 25 – Страница каталога мероприятий

При переходе на страницу конкретного мероприятия отображается его полное описание: название, формат, дата и время, адрес проведения, организатор, возрастная категория и счётчик занятых мест. Авторизованному пользователю доступны две кнопки действий: «Записаться» / «Отменить запись» и «В избранное» / «В избранном». Кнопка «Записаться» становится недоступной при отсутствии свободных мест и отображает текст «Мест нет». Страница мероприятия показана на рисунке 26.

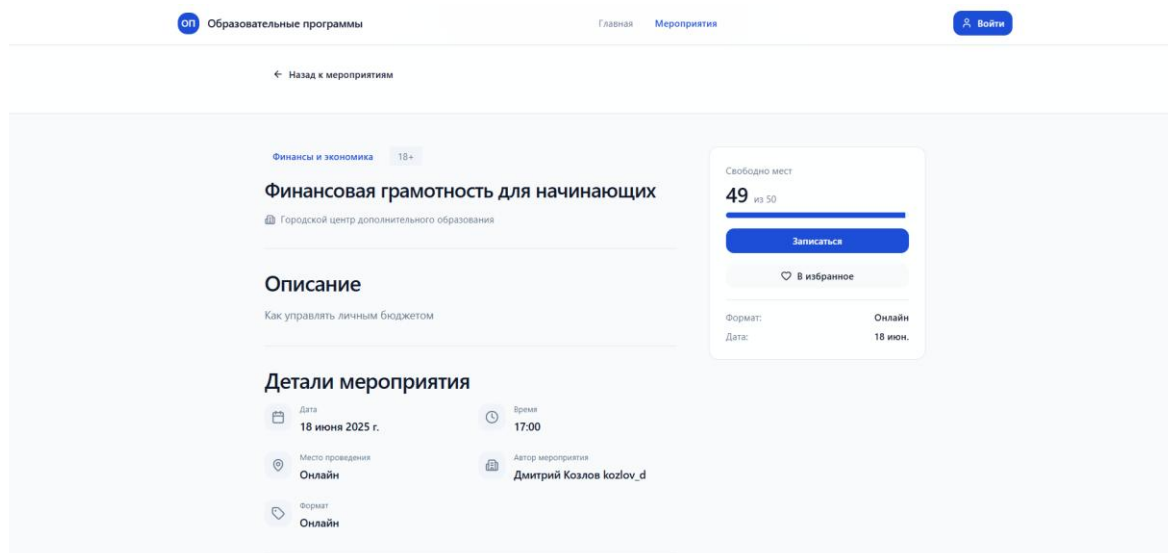


Рисунок 26 – Страница мероприятия

Личный кабинет авторизованного пользователя содержит блок с информацией об аккаунте и четыре вкладки. Вкладка «Мои записи» отображает мероприятия, на которые пользователь зарегистрирован. Вкладка «Избранное» – события, добавленные в избранное. Вкладка «Мои мероприятия» содержит список созданных пользователем событий с указанием статуса модерации. Вкладка «Настройки» открывает модальное окно «Настройки профиля», в котором можно изменить имя, фамилию, имя пользователя, email, а также сменить пароль. Личный кабинет представлен на рисунке 27.

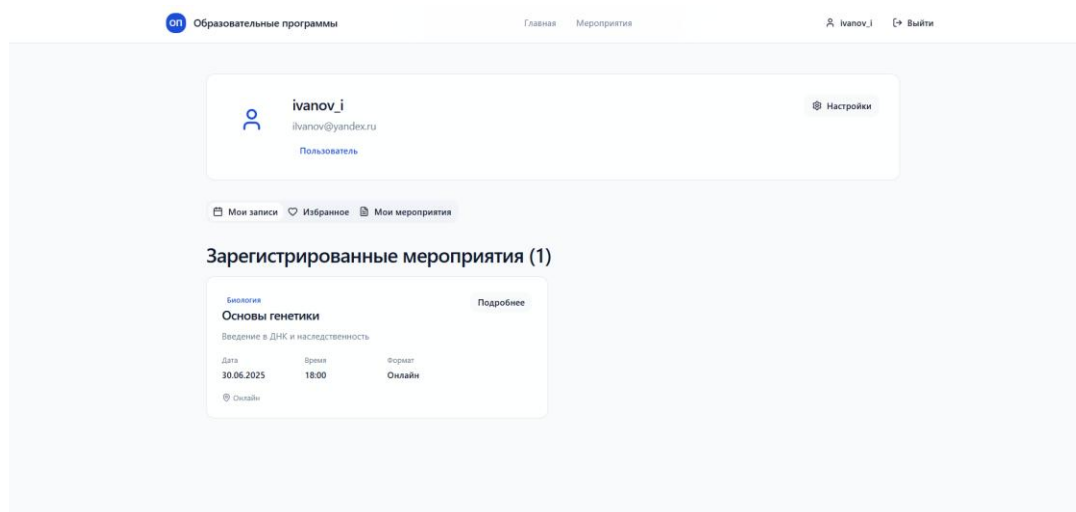


Рисунок 27 – Личный кабинет пользователя

Для создания нового мероприятия авторизованный пользователь нажимает кнопку «Создать мероприятие» в шапке. Открывается модальное окно с формой, в которой заполняются обязательные поля – название, формат и дата – а также необязательные: описание, время, адрес, максимальное число участников, тема, возрастная категория и организатор. После сохранения мероприятие получает статус «На модерации» и не отображается в общем каталоге до одобрения администратором. Форма создания мероприятия представлена на рисунке 28.

Рисунок 28 – Форма создания мероприятия

Административная панель доступна пользователям с ролью admin и содержит сводку статистики – карточки с общим количеством мероприятий, числом ожидающих модерации, опубликованных, скрытых, удалённых и

зарегистрированных пользователей. Ниже расположены три вкладки. Вкладка «Модерация» отображает мероприятия со статусом «На модерации»: для каждого доступны кнопки «Одобрить», «Снять» и «Удалить». Вкладка «Все мероприятия» предоставляет полный список событий с возможностью поиска и управления их статусом. Вкладка «Пользователи» позволяет изменить роль пользователя через выпадающий список – «Пользователь», «Организатор» или «Администратор» – а также заблокировать аккаунт кнопкой «Заблокировать». Административная панель показана на рисунке 29.

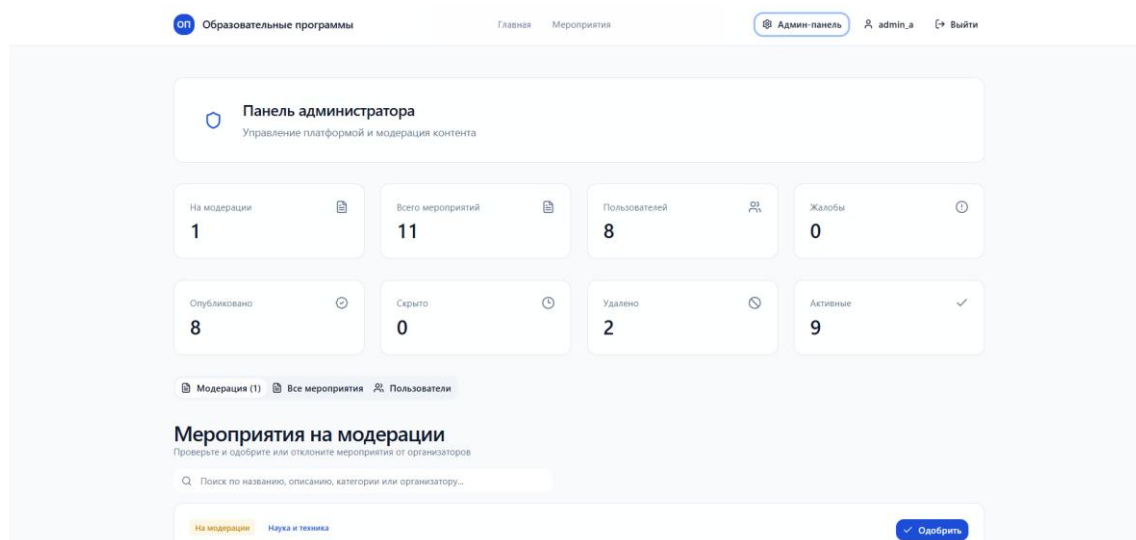


Рисунок 29 – Административная панель

2. Безопасность жизнедеятельности

Разработка веб-приложения предполагает продолжительную работу специалиста за персональным компьютером в офисном помещении. Подобный характер труда сопряжён с рядом производственных факторов, способных негативно сказаться на состоянии здоровья: зрительным перенапряжением, статической нагрузкой на опорно-двигательный аппарат, воздействием электромагнитного излучения.

Требования к освещению рабочего места

Правильно организованное освещение является одним из ключевых условий сохранения зрения сотрудника, работающего с ПЭВМ. Нормативные документы устанавливают следующий перечень требований к световой среде на подобных рабочих местах.

Уровень освещённости горизонтальной рабочей поверхности должен находиться в диапазоне 300–500 лк. Как недостаточный, так и избыточный световой поток одинаково вреден для зрения, поэтому отклонение от указанного диапазона в любую сторону недопустимо.

Световой поток в помещении необходимо распределять равномерно, исключая резкие перепады яркости между рабочей зоной и окружающим пространством; соотношение яркостей экрана и прилегающего фона не должно превышать 3:1.

Источники естественного освещения следует располагать таким образом, чтобы прямой солнечный свет падал преимущественно с левой стороны от рабочего стола. Окна и светильники не должны попадать в поле зрения оператора и не должны располагаться непосредственно позади него.

Яркость бликовых отражений на экране монитора ограничивается величиной 40 кд/м², яркость потолочного перекрытия – 200 кд/м². Превышение этих значений создаёт дискомфорт и ускоряет зрительное утомление.

					ПГУ1.090301.39.001 ПЗ	Лист
						80
Изм.	Лист	№ докум.	Подпись	Дата		

Для общего искусственного освещения предпочтительно применять люминесцентные лампы серии ЛБ в комплекте с рассеивающими экранами или решётчатыми экранами, обеспечивающими рассеянный световой поток без прямых бликов на рабочих поверхностях. Дополнительно на каждом рабочем месте рекомендуется устанавливать настольный светильник с регулируемым положением плафона и ступенчатой регулировкой яркости.

Пренебрежение приведёнными нормами влечёт постепенное снижение остроты зрения, хроническое утомление зрительного анализатора и, как следствие, снижение эффективности профессиональной деятельности разработчика.

2.1 Организация рабочего места

Эргономически грамотно обустроенное рабочее место снижает статическую нагрузку на мышцы шеи, плечевого пояса и нижнего отдела позвоночника, а также уменьшает вероятность развития туннельного синдрома запястья. К организации рабочего пространства пользователя ПЭВМ предъявляются следующие требования.

Минимально допустимое расстояние между рабочими столами, на которых установлены видеомониторы, составляет 2,0 м по фронту и 1,2 м – между боковыми поверхностями соседних мониторов. Такое расстояние обеспечивает защиту от взаимного электромагнитного воздействия оборудования.

Оптимальное удаление экрана от глаз оператора – от 600 до 700 мм; абсолютный минимум составляет 500 мм. Данное расстояние выбирается с учётом размера шрифта и символов, используемых в рабочих программах.

Высота рабочей поверхности стола регулируется в диапазоне 680–800 мм; конструкция столешницы должна позволять разместить на ней всё необходимое оборудование без нагромождения, оставляя свободное пространство для предплечий.

Рабочее кресло должно обеспечивать независимую регулировку высоты сиденья, угла наклона спинки и расстояния спинки от переднего края сиденья;

					ПГУ1.090301.39.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		81

механизм фиксации каждого из параметров должен надёжно удерживать выбранное положение в течение всей рабочей смены.

Обивка сиденья и спинки кресла выполняется из полумягких материалов, не накапливающих статическое электричество и допускающих влажную уборку дезинфицирующими составами.

Для снижения влияния электромагнитного излучения и зрительного дискомфорта рекомендуется использовать мониторы с заводским антибликовым и антирадиационным покрытием матрицы либо оснащать экраны дополнительными защитными фильтрами.

Совокупность перечисленных мер позволяет сформировать рабочее пространство, при котором нагрузка на тело разработчика в течение смены остаётся в физиологически допустимых пределах.

2.2 Микроклимат помещения

Параметры воздушной среды в рабочем помещении оказывают прямое влияние на самочувствие и работоспособность сотрудника. В кабинетах с ПЭВМ необходимо обеспечивать следующие условия.

Температура воздуха в рабочей зоне поддерживается на уровне 19–23 °С в холодный сезон и 20–25 °С – в тёплый; колебания температуры на протяжении смены не должны превышать 4 °С.

Относительная влажность воздуха поддерживается в диапазоне 40–60%, при целевом значении 55%. При снижении влажности ниже 40% рекомендуется использование бытовых или промышленных увлажнителей воздуха, поскольку сухой воздух ускоряет пересыхание слизистых оболочек глаз и дыхательных путей.

Помещение должно быть оборудовано естественной или принудительной вентиляцией, обеспечивающей не менее однократного воздухообмена в час; прямые потоки кондиционированного воздуха не должны быть направлены на рабочие места.

Поддержание стабильного микроклимата в указанных параметрах снижает нагрузку на терморегуляторную систему организма и создаёт благоприятный фон для продуктивной умственной деятельности.

2.3 Режим труда и отдыха

Высокая информационная насыщенность работы программиста требует грамотного чередования периодов активной работы с ПЭВМ и восстановительных пауз.

Непрерывная работа за монитором не должна превышать одного астрономического часа; по истечении этого времени необходимо сделать регламентированный перерыв вне зависимости от субъективного ощущения усталости.

Продолжительность каждого регламентированного перерыва составляет 10–15 минут; в этот период рекомендуется переключиться на деятельность, не связанную со зрительным напряжением: выполнить физические упражнения, пройтись или ознакомиться с бумажными документами.

В ночную смену (22:00–06:00) хронобиологические ритмы организма существенно замедляют восстановление, поэтому продолжительность регламентированных перерывов увеличивается на 30% относительно дневной нормы.

Суммарное время работы непосредственно с компьютером не должно превышать 6 часов за смену. Задачи, которые позволяют обойтись без монитора – совещания, работа с печатными материалами, обсуждения – целесообразно планировать в качестве «разгрузочных» промежутков.

Строгое соблюдение режима труда и отдыха позволяет сохранять концентрацию внимания на протяжении всей рабочей смены и препятствует накоплению хронического утомления.

Заключение

В рамках выпускной квалификационной работы разработано клиент-серверное веб-приложение для публикации и поиска мероприятий дополнительного образования с разграничением ролей участников: рядовой пользователь получает доступ к каталогу мероприятий и инструментам записи на них, организатор публикует собственные события и управляет ими, администратор осуществляет общий контроль над контентом платформы.

В процессе разработки применён актуальный технологический стек: клиентская часть построена на библиотеке React с использованием Vite, Tailwind CSS и Radix UI; серверная логика реализована на платформе Node.js с фреймворком Express; хранение данных обеспечивает СУБД PostgreSQL с ORM Prisma. Разработка велась в среде Visual Studio Code.

Реализованный функционал охватывает полный цикл взаимодействия с платформой: публикацию и просмотр мероприятий, фильтрацию по категориям, тегам, возрасту и формату, аутентификацию на основе JWT, личный кабинет с избранным и записью на мероприятия, а также административную панель с инструментами модерации и обработки жалоб. Интерфейс адаптирован для мобильных и десктопных устройств.

Проведённое тестирование подтвердило соответствие приложения всем поставленным требованиям. Задание на дипломное проектирование выполнено в полном объёме.

В числе перспективных направлений дальнейшего развития выделяются: интеграция с картографическими сервисами, расширение аналитики для организаторов и поддержка многоязычного интерфейса. Практическая значимость работы определяется тем, что созданная платформа объединяет организаторов и участников мероприятий, снижает барьеры при поиске образовательных возможностей и способствует повышению доступности дополнительного образования.

					ПГУ1.090301.39.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		84

Список использованных источников

1. Концепция развития дополнительного образования детей до 2030 года [Электронный ресурс] / Режим доступа: <https://docs.edu.gov.ru/document/b8a5dc6a5c2cfe975bd4c6b24523e318/>, свободный. (Дата обращения: 05.04.2026)
2. Флэнаган Д. JavaScript: подробное руководство / Пер. с англ. – 7-е изд. – СПб.: «Питер», 2023. – 688 с.
3. Документация React [Электронный ресурс] / Режим доступа: <https://react.dev/learn>, свободный. (Дата обращения: 10.04.2026)
4. Документация Vite [Электронный ресурс] / Режим доступа: <https://vitejs.dev/guide/>, свободный. (Дата обращения: 12.04.2026)
5. Документация Tailwind CSS [Электронный ресурс] / Режим доступа: <https://tailwindcss.com/docs/installation>, свободный. (Дата обращения: 14.04.2026)
6. Документация Radix UI [Электронный ресурс] / Режим доступа: <https://www.radix-ui.com/docs/primitives/overview/introduction>, свободный. (Дата обращения: 18.04.2026)
7. Документация Node.js [Электронный ресурс] / Режим доступа: <https://nodejs.org/en/docs>, свободный. (Дата обращения: 22.04.2026)
8. Документация Express.js [Электронный ресурс] / Режим доступа: <https://expressjs.com/en/guide/routing.html>, свободный. (Дата обращения: 25.04.2026)
9. Документация PostgreSQL [Электронный ресурс] / Режим доступа: <https://www.postgresql.org/docs/current/>, свободный. (Дата обращения: 29.04.2026)
10. Документация Prisma ORM [Электронный ресурс] / Режим доступа: <https://www.prisma.io/docs/orm>, свободный. (Дата обращения: 02.04.2026)
11. Документация JSON Web Token (JWT) [Электронный ресурс] / Режим доступа: <https://jwt.io/introduction>, свободный. (Дата обращения: 20.04.2026)

12. Лукас Д. Разработка веб-приложений с применением Node.js / Пер. с англ. – М.: «ДМК Пресс», 2020. – 392 с.

13. Бэнкс Алекс, Порселло Ева. React и Redux: функциональная веб-разработка. – СПб.: «Питер», 2018. – 336 с.

14. REST API Design Best Practices [Электронный ресурс] / Режим доступа: <https://restfulapi.net/rest-api-design-tutorial-with-example/>, свободный. (Дата обращения: 10.04.2026)

15. Документация bcrypt (хеширование паролей) [Электронный ресурс] / Режим доступа: <https://www.npmjs.com/package/bcrypt>, свободный. (Дата обращения: 17.04.2026)

16. Введение в адаптивный веб-дизайн [Электронный ресурс] / Режим доступа: https://developer.mozilla.org/ru/docs/Learn/CSS/CSS_layout/Responsive_Design, свободный. (Дата обращения: 24.04.2026)

Приложение А

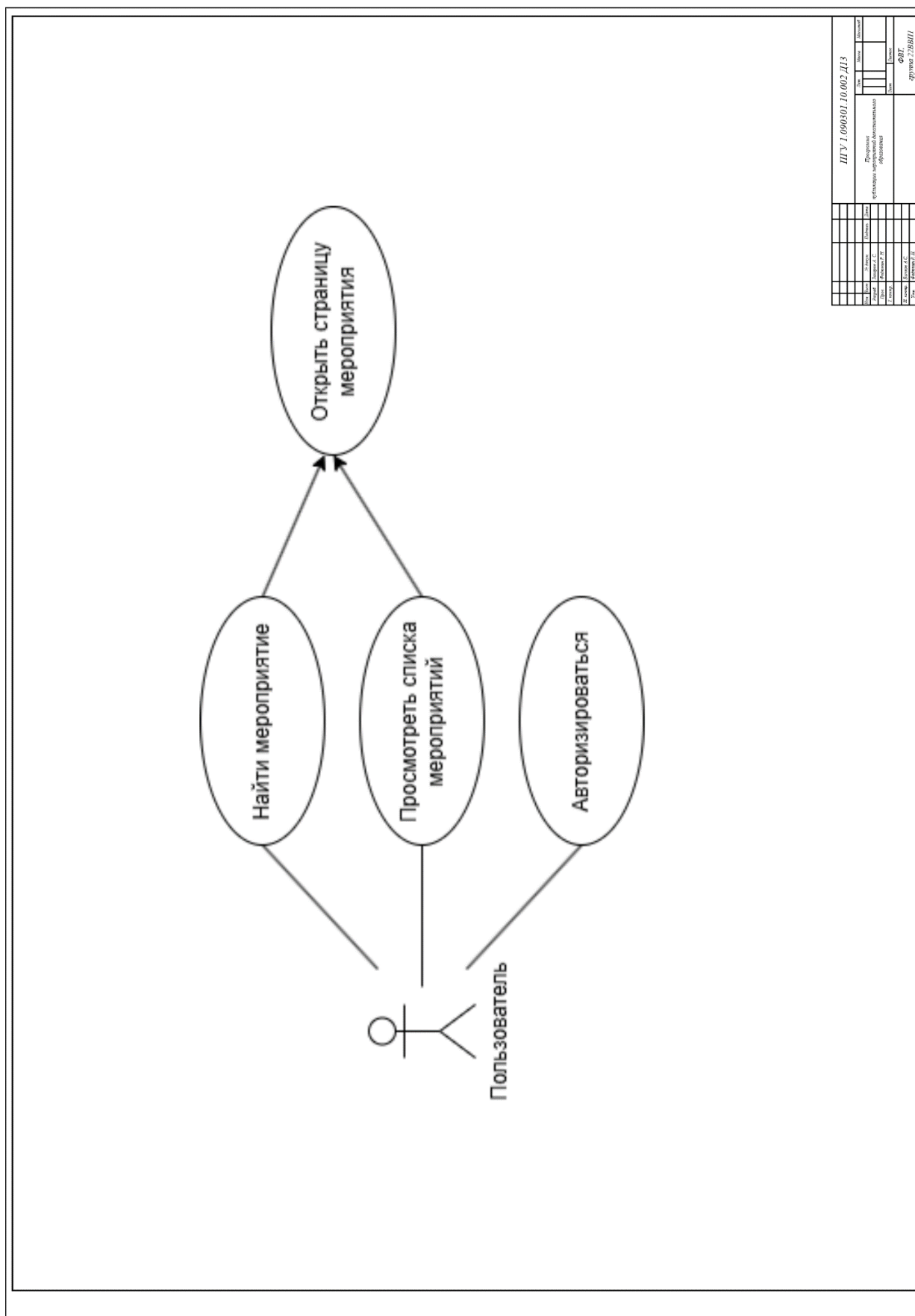
(обязательное)

Программа публикации мероприятий дополнительного образования

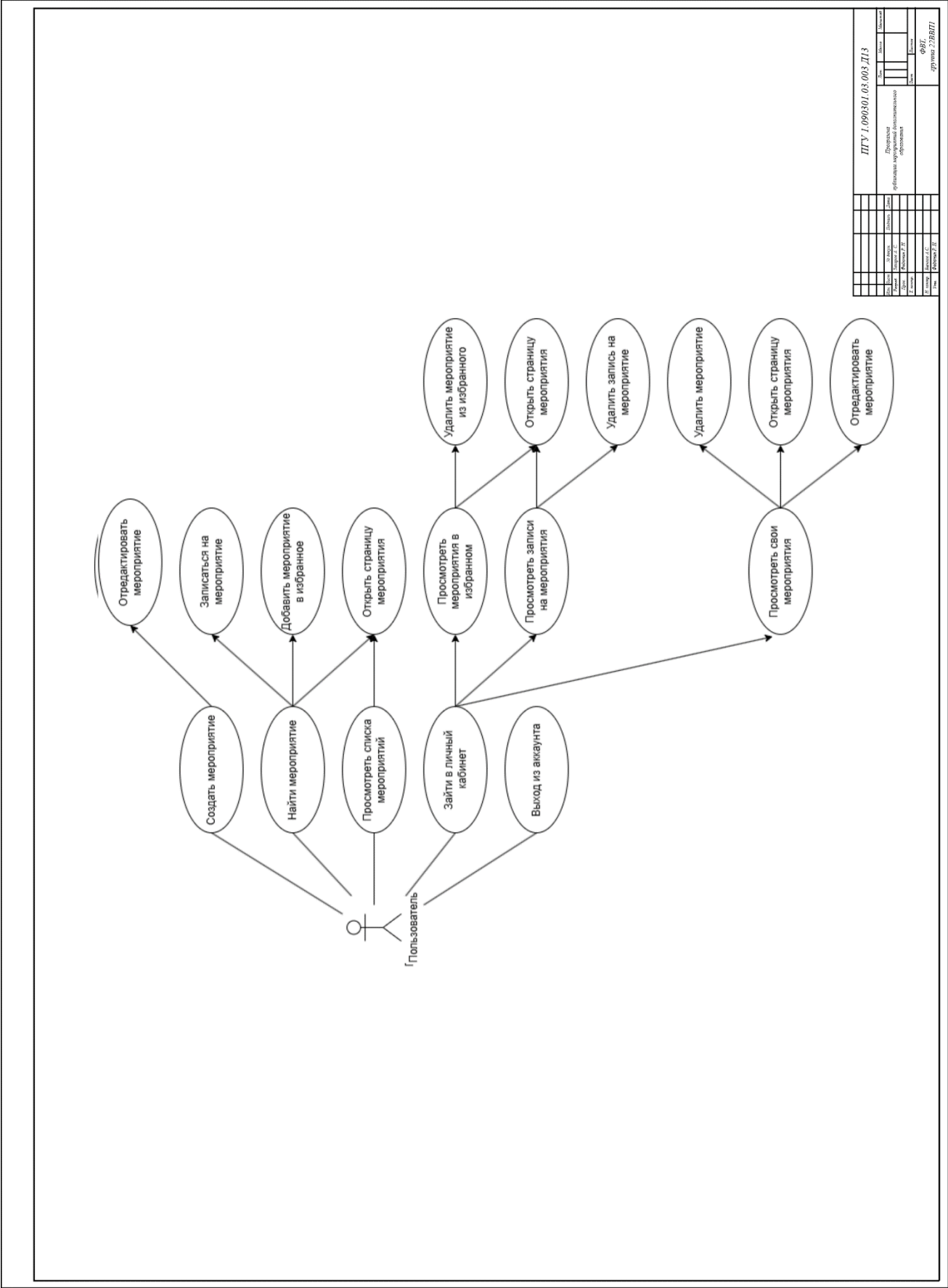
UML-диаграммы

					ПГУ1.090301.39.001 ПЗ	Лист
						87
Изм.	Лист	№ докум.	Подпись	Дата		

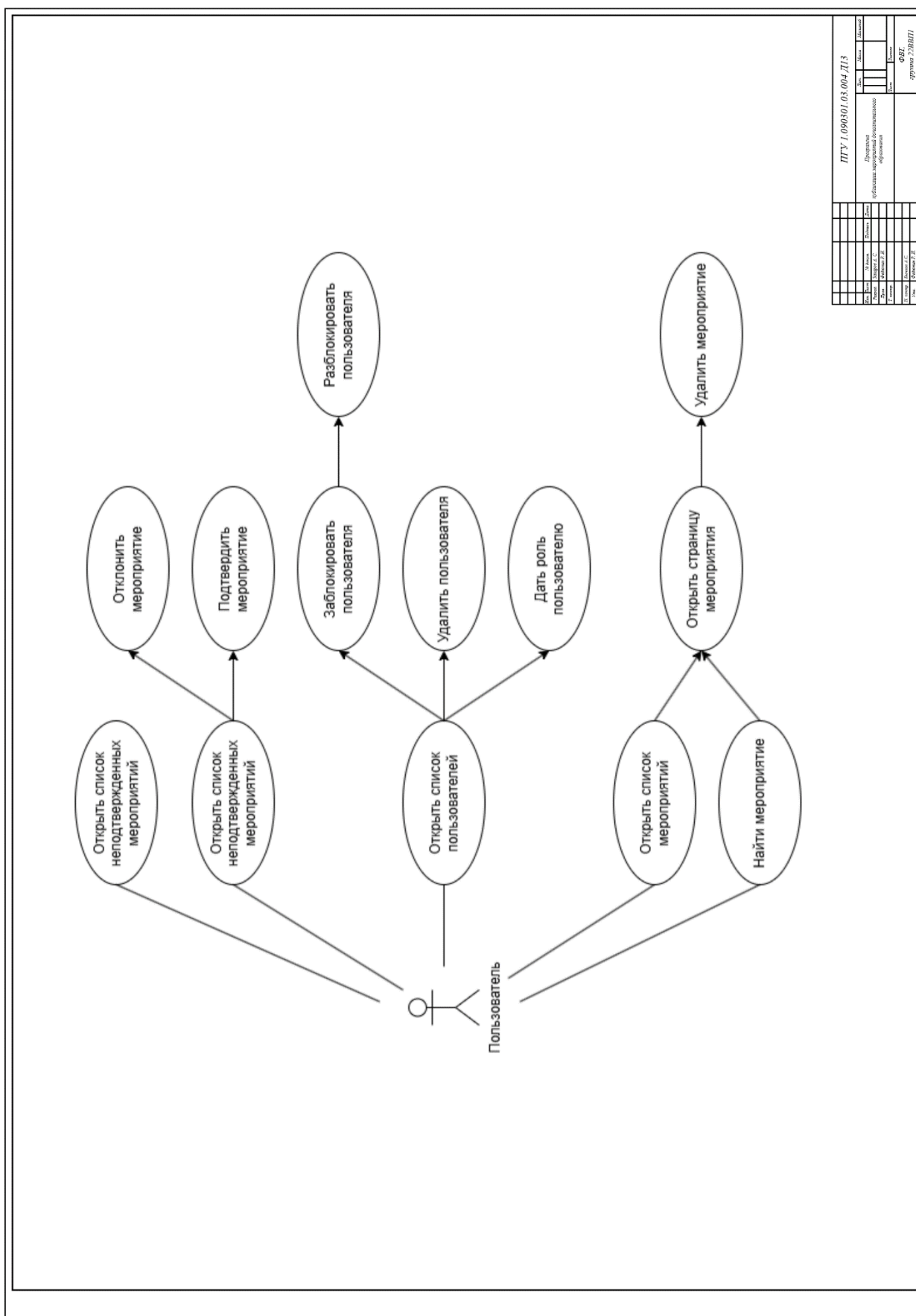
Приложение А.1 - Диаграмма прецедентов неавторизованного пользователя



Приложение А.2 - Диаграмма прецедентов авторизованного пользователя



Приложение А.3 - Диаграмма прецедентов администратора



ПГУ 1.090301.03.004 ПЗ									
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия

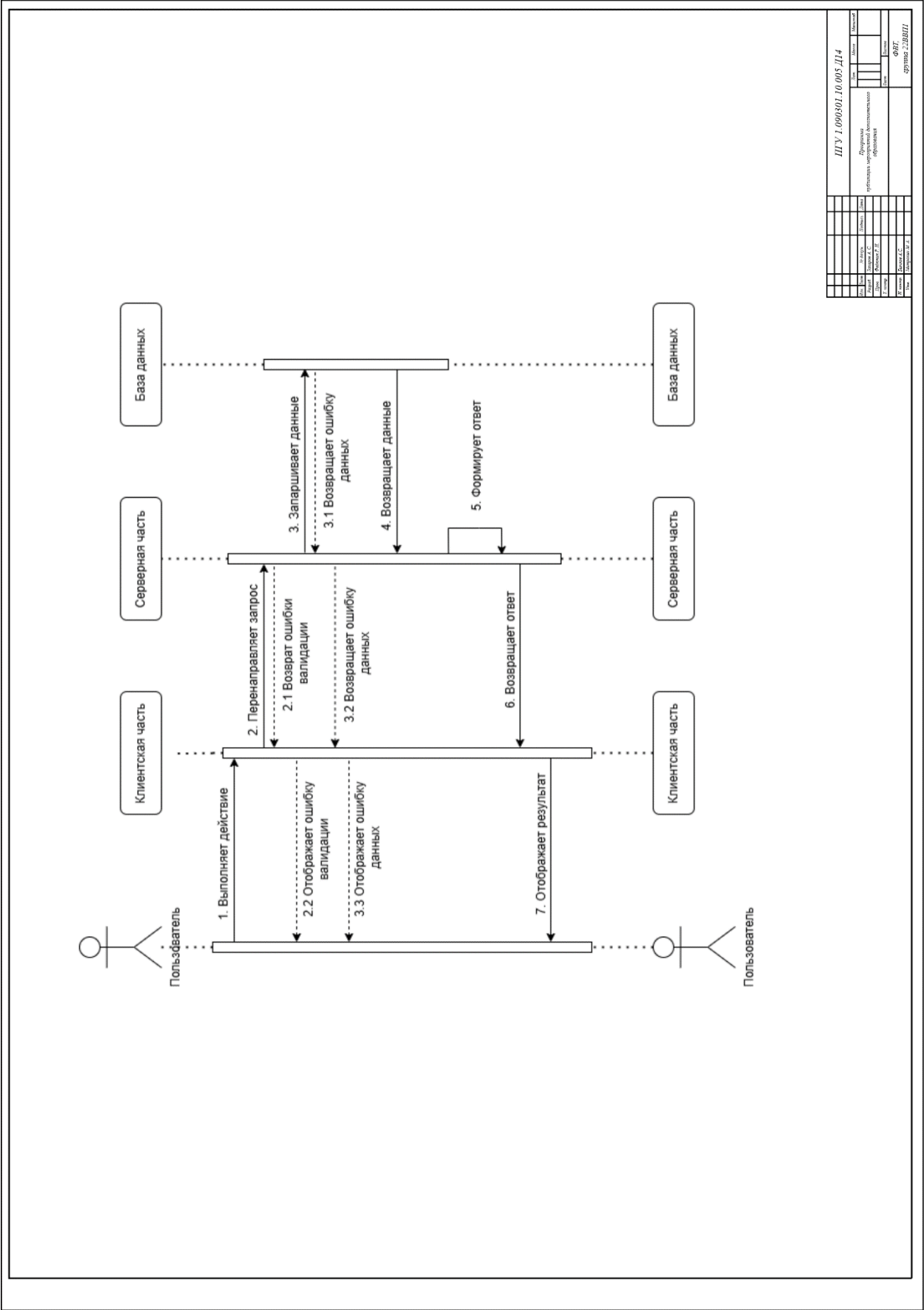
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.39.001 ПЗ

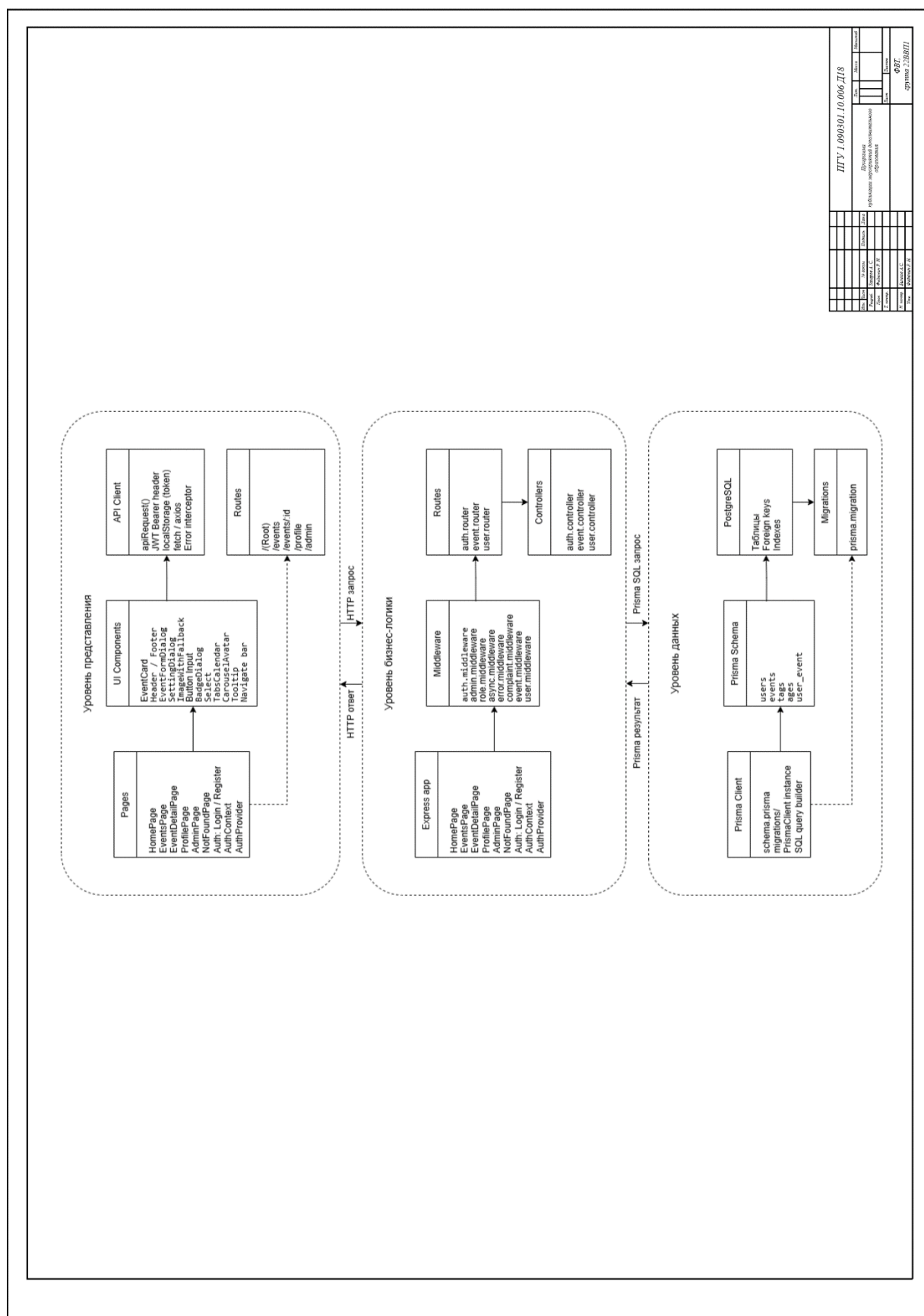
Лист

90

Приложение А.4 - Диаграмма последовательности функционирования



Приложение А.5 - Диаграмма компонентов приложения



Приложение Б

(обязательное)

Программа публикации мероприятий дополнительного образования

Листинг программы

Файл «HomePage.jsx»

```
import { useEffect, useState } from 'react';
import { Link } from 'react-router-dom';
import { Button } from '../ui/button';
import { Search, Calendar, Users, Shield, ArrowRight } from
' lucide-react';
import { apiRequest } from '../../api/client';
import { EventCard } from '../ui/EventCard';
import photo from '../../assets/photo-1758270704524-
596810e891b5.jpg';

export function HomePage() {
  const [featuredEvents, setFeaturedEvents] = useState([]);
  const [loading, setLoading] = useState(true);
  const [tagsMap, setTagsMap] = useState({});
  const [agesMap, setAgesMap] = useState({});

  useEffect(() => {
    const loadData = async () => {
      try {
        const [tagsData, agesData, eventsData] = await
Promise.all([
          apiRequest('/api/events/tags'),
          apiRequest('/api/events/ages'),
          apiRequest('/api/events'),
        ]);

        const tagsObj = Object.fromEntries(tagsData.map(t =>
[t.id, t.tag]));
        const agesObj = Object.fromEntries(agesData.map(a =>
[a.id, a.age_category]));

        setTagsMap(tagsObj);
        setAgesMap(agesObj);

        const events = Array.isArray(eventsData)
          ? eventsData
          : eventsData.events || [];

        const shuffled = [...events];
        for (let i = shuffled.length - 1; i > 0; i--) {
          const j = Math.floor(Math.random() * (i + 1));
```

					ПГУ1.090301.39.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		93

```

        [shuffled[i], shuffled[j]] = [shuffled[j],
shuffled[i]];
    }

    setFeaturedEvents(shuffled.slice(0, 3));
  } catch (error) {
    console.error('Ошибка загрузки данных:', error);
    setFeaturedEvents([]);
  } finally {
    setLoading(false);
  }
};

loadData();
}, []);

const scrollToEvents = () => {
  const eventsSection = document.getElementById('events-
preview');
  eventsSection?.scrollIntoView({ behavior: 'smooth' });
};

return (
  <div className="flex flex-col">
    <section className="relative bg-card border-b border-
border">
      <div className="max-w-7xl mx-auto px-6 py-24">
        <div className="grid lg:grid-cols-2 gap-12 items-
center">
          <div className="space-y-6">
            <h1 className="text-4xl font-semibold text-
foreground leading-tight">
              Платформа образовательных мероприятий
            </h1>
            <p className="text-lg text-muted-foreground
leading-relaxed">
              Открывайте новые возможности для развития.
Находите актуальные образовательные программы,
              регистрируйтесь на мероприятия и расширяйте
свои компетенции.
            </p>
          </div>
          <div className="flex gap-4 pt-4">
            <Link to="/events">
              <Button size="lg" className="gap-2">
                Найти мероприятие
                <ArrowRight className="w-4 h-4" />
              </Button>
            </Link>
          </div>
        </div>
      </section>
    </div>
  );
};

```

```

                                <Button size="lg" variant="outline"
onClick={scrollToEvents}>
                                Узнать больше
                                </Button>
                                </div>
                                </div>

                                <div className="relative h-96 rounded-lg overflow-
hidden border border-border">
                                <img
                                src={photo}
                                alt="Образовательные программы"
                                className="w-full h-full object-cover"
                                />
                                </div>
                                </div>
                                </div>
                                </section>

                                <section className="py-16 bg-background">
                                <div className="max-w-7xl mx-auto px-6">
                                <div className="grid md:grid-cols-4 gap-8">
                                <div className="space-y-3">
                                <div className="w-12 h-12 bg-primary/10 rounded-
lg flex items-center justify-center">
                                <Search className="w-6 h-6 text-primary" />
                                </div>
                                <h3 className="font-medium text-
foreground">Удобный поиск</h3>
                                <p className="text-sm text-muted-foreground">
                                Находите мероприятия по категориям, датам и
форматам проведения
                                </p>
                                </div>

                                <div className="space-y-3">
                                <div className="w-12 h-12 bg-primary/10 rounded-
lg flex items-center justify-center">
                                <Calendar className="w-6 h-6 text-primary" />
                                </div>
                                <h3 className="font-medium text-
foreground">Простая регистрация</h3>
                                <p className="text-sm text-muted-foreground">
                                Записывайтесь на мероприятия в один клик и
управляйте записями
                                </p>
                                </div>

                                <div className="space-y-3">

```

```

        <div className="w-12 h-12 bg-primary/10 rounded-
lg flex items-center justify-center">
            <Users className="w-6 h-6 text-primary" />
        </div>
        <h3 className="font-medium text-foreground">Для
организаторов</h3>
        <p className="text-sm text-muted-foreground">
Публикуйте мероприятия и управляйте участниками
        </p>
    </div>

    <div className="space-y-3">
        <div className="w-12 h-12 bg-primary/10 rounded-
lg flex items-center justify-center">
            <Shield className="w-6 h-6 text-primary" />
        </div>
        <h3 className="font-medium text-
foreground">Надёжность</h3>
        <p className="text-sm text-muted-foreground">
Модерация контента и проверка организаторов
        </p>
    </div>
</div>
</section>

<section id="events-preview" className="py-16 bg-card
border-t border-border">
    <div className="max-w-7xl mx-auto px-6">
        <div className="flex items-center justify-between
mb-8">
            <div>
                <h2 className="text-2xl font-semibold text-
foreground">Популярные мероприятия</h2>
                <p className="text-muted-foreground mt-
1">Актуальные образовательные программы</p>
            </div>
            <Link to="/events">
                <Button variant="outline" className="gap-2">
                    Все мероприятия
                    <ArrowRight className="w-4 h-4" />
                </Button>
            </Link>
        </div>

        {loading ? (
            <div className="text-muted-foreground">Загрузка
мероприятий...</div>
        ) : (
            <div className="grid md:grid-cols-3 gap-6">

```

```

        {featuredEvents.map((event) => (
            <EventCard
                key={event.id}
                event={event}
                tagsMap={tagsMap}
                agesMap={agesMap}
            />
        ))}
    </div>
  )}
</div>
</section>

    <section className="py-16 bg-background border-t
border-border">
    <div className="max-w-4xl mx-auto px-6 text-center
space-y-6">
        <h2 className="text-2xl font-semibold text-
foreground">
            Готовы начать обучение?
        </h2>
        <p className="text-muted-foreground">
            Присоединяйтесь к тысячам пользователей, которые
уже развивают свои навыки через нашу платформу
        </p>
        <Link to="/events">
            <Button size="lg" className="gap-2">
                Посмотреть все мероприятия
                <ArrowRight className="w-4 h-4" />
            </Button>
        </Link>
    </div>
</section>
</div>
);
}

```

Файл «EventsPage.jsx»

```

import { useEffect, useMemo, useState } from 'react';
import { Input } from '../ui/input';
import { Button } from '../ui/button';
import { Search, Filter } from 'lucide-react';
import { Checkbox } from '../ui/checkbox';
import { Label } from '../ui/label';
import { apiRequest } from '../../api/client';
import { EventCard } from '../ui/EventCard';

export default function EventsPage() {
  const [events, setEvents] = useState([]);
  const [loading, setLoading] = useState(true);
  const [searchQuery, setSearchQuery] = useState('');

```

```

const [selectedFormats, setSelectedFormats] = useState([]);
const [selectedCategories, setSelectedCategories] =
useState([]);
const [selectedAges, setSelectedAges] = useState([]);
const [tagsMap, setTagsMap] = useState({});
const [agesMap, setAgesMap] = useState({});

useEffect(() => {
  const loadEvents = async () => {
    try {
      const [eventsData, tagsData, agesData] = await
Promise.all([
        apiRequest('/api/events'),
        apiRequest('/api/events/tags'),
        apiRequest('/api/events/ages'),
      ]);

      setTagsMap(Object.fromEntries(tagsData.map((t) => [t.id,
t.tag])));
      setAgesMap(Object.fromEntries(agesData.map((a) => [a.id,
a.age_category])));

      const visibleEvents = Array.isArray(eventsData)
        ? eventsData.filter((event) => event.active === true &&
event.deleted === false)
        : [];

      setEvents(visibleEvents);
    } catch (error) {
      console.error(error);
      setEvents([]);
    } finally {
      setLoading(false);
    }
  };

  loadEvents();
}, []);

const formats = ['online', 'offline', 'hybrid'];

const categories = useMemo(
  () => [...new Set(events.map((e) =>
e.tag_id).filter(Boolean))],
  [events]
);

const ages = useMemo(
  () => [...new Set(events.map((e) =>
e.age_id).filter(Boolean))],

```

					ПГУ1.090301.39.001 ПЗ	Лист
						98
Изм.	Лист	№ докум.	Подпись	Дата		

```

    [events]
  );

  const filteredEvents = useMemo(() => {
    return events.filter((event) => {
      const q = searchQuery.toLowerCase();

      const matchesSearch =
        searchQuery === '' ||
        (event.name || '').toLowerCase().includes(q) ||
        (event.description || '').toLowerCase().includes(q) ||
        (event.author || '').toString().toLowerCase().includes(q)
      ||
        (tagsMap[event.tag_id] || '').toLowerCase().includes(q)
      ||
        (agesMap[event.age_id] || '').toLowerCase().includes(q);

      const matchesFormat =
        selectedFormats.length === 0 ||
selectedFormats.includes(event.format);

      const matchesCategory =
        selectedCategories.length === 0 ||
selectedCategories.includes(event.tag_id);

      const matchesAge =
        selectedAges.length === 0 ||
selectedAges.includes(event.age_id);

      return matchesSearch && matchesFormat && matchesCategory &&
matchesAge;
    });
  }, [events, searchQuery, selectedFormats, selectedCategories,
selectedAges, tagsMap, agesMap]);

  const toggleFilter = (value, selected, setter) => {
    if (selected.includes(value)) setter(selected.filter((v) =>
v !== value));
    else setter([...selected, value]);
  };

  const clearFilters = () => {
    setSearchQuery('');
    setSelectedFormats([]);
    setSelectedCategories([]);
    setSelectedAges([]);
  };

  return (
    <div className="min-h-screen bg-background">

```

```

<section className="bg-card border-b border-border py-8">
  <div className="max-w-7xl mx-auto px-6">
    <h1 className="text-2xl font-semibold text-foreground
mb-6">Каталог мероприятий</h1>
    <div className="flex gap-3">
      <div className="flex-1 relative">
        <Search className="absolute left-3 top-1/2 -
translate-y-1/2 w-5 h-5 text-muted-foreground" />
        <Input
          placeholder="Поиск по названию, описанию или
организатору..."
          value={searchQuery}
          onChange={(e) => setSearchQuery(e.target.value)}
          className="pl-10 border border-border bg-card"
        />
      </div>
      {
        (searchQuery || selectedFormats.length > 0 ||
selectedCategories.length > 0 || selectedAges.length > 0) && (
          <Button variant="outline" onClick={clearFilters}>
            Сбросить фильтры
          </Button>
        )
      }
    </div>
  </div>
</section>

<div className="max-w-7xl mx-auto px-6 py-8">
  <div className="flex gap-8">
    <aside className="w-[280px] flex-shrink-0 hidden
lg:block">
      <div className="bg-card border border-border rounded-
lg p-6 space-y-6">
        <div className="flex items-center gap-2">
          <Filter className="w-5 h-5 text-muted-foreground"
/>
          <h2 className="font-medium text-
foreground">Фильтры</h2>
        </div>

        <div className="space-y-3">
          <h3 className="text-sm font-medium text-
foreground">Формат проведения</h3>
          <div className="space-y-2">
            {formats.map((format) => (
              <div key={format} className="flex items-
center gap-2">
                <Checkbox
                  id={`format-${format}`}
                  checked={selectedFormats.includes(format)}

```

```

                                onChange={ () => toggleFilter(format,
selectedFormats, setSelectedFormats)}
                                />
                                <Label htmlFor={`format-${format}`}
className="text-sm text-muted-foreground capitalize cursor-pointer">
                                {format === 'online' ? 'Онлайн' : format
=== 'offline' ? 'Оффлайн' : 'Гибрид'}
                                </Label>
                                </div>
                                </div>
                                </div>
                                </div>

                                <div className="space-y-3 border-t border-border
pt-6">
                                <h3 className="text-sm font-medium text-
foreground">Категория</h3>
                                <div className="space-y-2">
                                {categories.map((tagId) => (
                                <div key={tagId} className="flex items-center
gap-2">
                                <Checkbox
                                id={`category-${tagId}`}
                                checked={selectedCategories.includes(tag
gId)}
                                onChange={ () => toggleFilter(tagId,
selectedCategories, setSelectedCategories)}
                                />
                                <Label htmlFor={`category-${tagId}`}
className="text-sm text-muted-foreground cursor-pointer">
                                {tagsMap[tagId] || tagId}
                                </Label>
                                </div>
                                </div>
                                </div>
                                </div>

                                <div className="space-y-3 border-t border-border
pt-6">
                                <h3 className="text-sm font-medium text-
foreground">Возрастная категория</h3>
                                <div className="space-y-2">
                                {ages.map((ageId) => (
                                <div key={ageId} className="flex items-center
gap-2">
                                <Checkbox
                                id={`age-${ageId}`}
                                checked={selectedAges.includes(ageId)}
                                onChange={ () => toggleFilter(ageId,
selectedAges, setSelectedAges)}

```

```

        />
        <Label htmlFor={`age-${ageId}`}
className="text-sm text-muted-foreground cursor-pointer">
        {agesMap[ageId] || ageId}
    </Label>
</div>
    )})
</div>
</div>
</div>
</aside>

<div className="flex-1 space-y-4">
    <p className="text-sm text-muted-foreground">
        Найдено мероприятий: <span className="font-medium
text-foreground">{filteredEvents.length}</span>
    </p>

    {loading ? (
        <div className="bg-card border border-border
rounded-lg p-12 text-center">
            <p className="text-muted-foreground">Загрузка
мероприятий...</p>
        </div>
    ) : filteredEvents.length === 0 ? (
        <div className="bg-card border border-border
rounded-lg p-12 text-center">
            <p className="text-muted-foreground">Мероприятия
не найдены</p>
        </div>
    ) : (
        <div className="grid gap-4 lg:grid-cols-2 2xl:grid-
cols-3">
            {filteredEvents.map((event) => (
                <EventCard key={event.id} event={event}
tagsMap={tagsMap} agesMap={agesMap} />
            ))}
        </div>
    )}
</div>
</div>
</div>
</div>
);
}

```

Приложение В

(обязательное)

Программа публикации мероприятий дополнительного образования

Презентация

Федеральное государственное бюджетное
образовательное учреждение высшего образования

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

«Программа для публикаций мероприятий дополнительного образования»

Захаров Антон Сергеевич
Направление подготовки:
09.03.01 «Информатика и вычислительная
техника»
группа: 22ВВП1

Руководитель:
к.т.н., доцент Федюнин Роман Николаевич

Актуальность

Проблема

- Отсутствует единая платформа для поиска образовательных мероприятий
- Сегмент на данный момент по большей части реализован очно: школы, клубы, секции
- Ограниченные возможности публикации для организаторов

Недостатки

- Публикация доступна только организациям или партнёрам
- Ориентация на развлекательный контент
- Отсутствует взаимодействие между участниками и организаторами

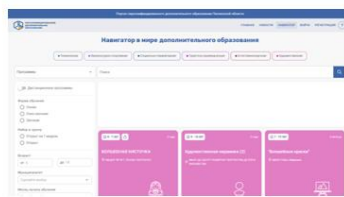
Применение

- Объединение участников и организаторов
- Повышение доступности дополнительного образования
- Упрощение процесса поиска и выбора мероприятий
- Снижение барьеров для выхода новых организаторов

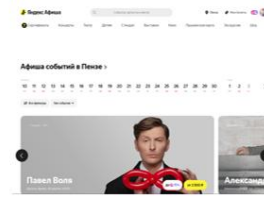
2

					ПГУ1.090301.39.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		103

Аналоги



Портал ПФДО



Яндекс Афиша



Культура.РФ

3

Цель и задачи

Цель Разработать веб-приложение для публикации, поиска и регистрации на мероприятия дополнительного образования с функциями модерации и управления профилем

Задачи

1. Анализ предметной области и изучение аналогов
2. Проектирование архитектуры клиент-серверного приложения
3. Разработка REST API на Node.js с Express
4. Реализация клиентской части на React
5. Проектирование и реализация БД на PostgreSQL
6. Реализация аутентификации JWT и тестирование

4

Обзор решения

Веб-приложение для публикации, поиска и записи на мероприятия дополнительного образования с административным управлением контентом

Функции

- Управление мероприятиями
- Поиск и фильтрация
- Регистрация на события
- Модерация контента

Роли

- Неавторизованный пользователь
- Авторизованный пользователь
- Администратор

Возможности

- Личный кабинет
- Управление профилем
- Управление мероприятиями

5

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.39.001 ПЗ

Лист

104

Обзор решения

Веб-приложение для публикации, поиска и записи на мероприятия дополнительного образования с административным управлением контентом

Функции	Роли	Возможности
<ul style="list-style-type: none">Управление мероприятиямиПоиск и фильтрацияРегистрация на событияМодерация контента	<ul style="list-style-type: none">Неавторизованный пользовательАвторизованный пользовательАдминистратор	<ul style="list-style-type: none">Личный кабинетУправление профилемУправление мероприятиями

5

Архитектура системы

Для реализации программы была выбрана трехуровневая архитектура системы

Основные уровни:

- Уровень представления
 - Пользовательский интерфейс
 - Взаимодействие с пользователем
 - Обмен данных с сервером
- Уровень бизнес логики
 - Обработка запросов от клиента
 - Контроль логики работы системы
 - Взаимодействие с БД через API
- Уровень данных
 - Хранение данных
 - Управление БД

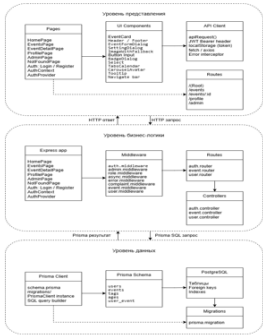


Диаграмма компонентов

6

Технологический стек

FRONTEND <ul style="list-style-type: none">React 18ViteTailwind CSSRadix UI	BACKEND <ul style="list-style-type: none">Node.jsJWTExpress.jsREST API
DATABASE <ul style="list-style-type: none">PostgreSQLPrisma ORMSQL запросы	SECURITY <ul style="list-style-type: none">BcryptCORSHTTP only cookies

7

Сценарий взаимодействия

Процесс: Клиент выполняет действие → Система валидирует данные → Сохранение в БД

Участники процесса

Клиент	Веб-браузер пользователя
Сервер	Серверное приложение Express.js
БД	PostgreSQL

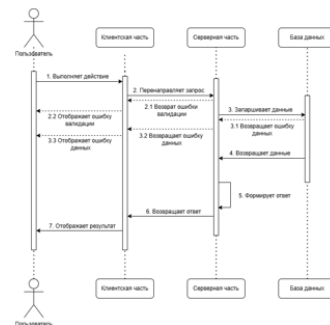


Диаграмма последовательности

8

Структура базы данных

Основные сущности:

Users (Пользователи)	Roles (Роли)
Events (Мероприятия)	Registrations (Регистрации)
Categories (Категории)	

Связи между таблицами:

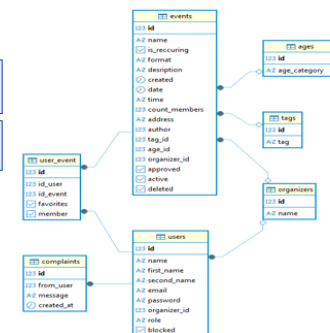
Users ↔ Roles (один ко многим)

Users → Events (один ко многим)

Users → Registrations ↔ Events (регистрации)

Events → Categories (одна категория, много событий)

Users (Admin) → Events (модерация)



Структура базы данных

9

Основной функционал

Каталог мероприятий

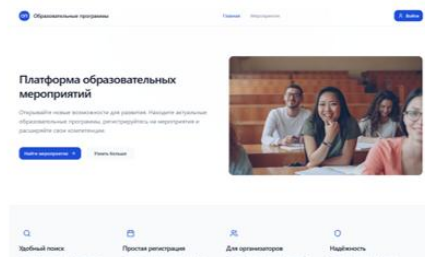
- Список всех событий
- Фильтрация по категориям
- Поиск по названию

Личный кабинет

- История записей
- Управление профилем
- Личные данные

Административная панель

- Модерация событий
- Управление пользователями
- Статистика



Главная страница

10

Изм.	Лист	№ докум.	Подпись	Дата

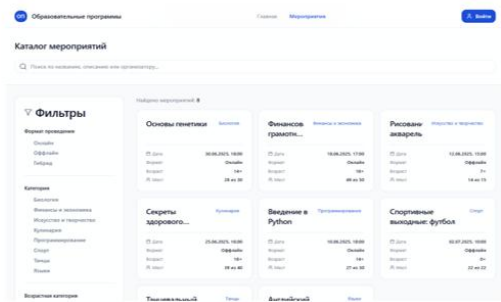
ПГУ1.090301.39.001 ПЗ

Лист

106

Основной функционал

Каталог мероприятий

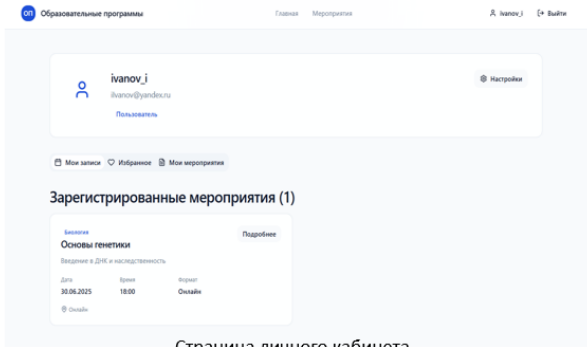


Страница каталога мероприятий

11

Основной функционал

Личный кабинет

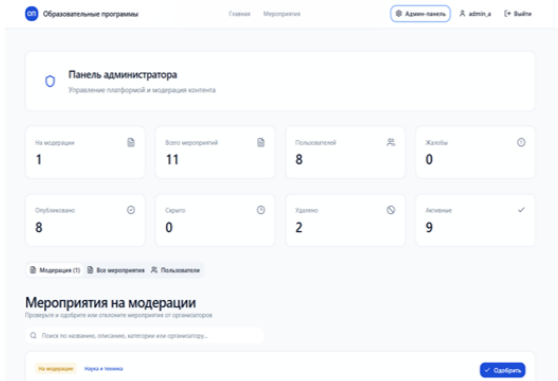


Страница личного кабинета

12

Основной функционал

Админпанель



Страница инструментов администратора

13

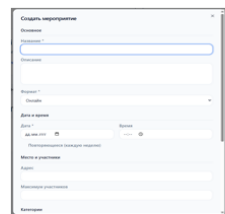
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.39.001 ПЗ

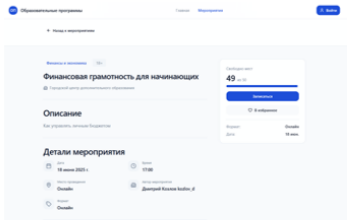
Результаты работы

В ходе выполнения ВКР были получены следующие результаты

- Список всех событий
- Фильтрация по категориям
- Поиск по названию



Форма создания мероприятия



Страница мероприятия

14

Заключение

Достигнутые результаты

- Разработано веб-приложение с открытой публикацией мероприятий
- Реализован поиск и фильтрация по ключевым параметрам
- Обеспечена запись пользователей на мероприятия
- Реализована поддержка ролей
- Внедрена система модерации контента
- Спроектирована и реализована база данных
- Реализована клиент-серверная архитектура и API

15

Спасибо за внимание!

Готов ответить на ваши вопросы

16

Изм.	Лист	№ докум.	Подпись	Дата