Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования «Уральский федеральный университет
имени первого Президента России Б. Н. Ельцина»

Институт радиоэлектроники и информационных технологий – РТФ

Школа бакалавриата

ДОПУСТИТЬ К ЗАЩИТЕ ПЕРЕД ГЭК
Руководитель образовательной программы
09.03.04 Программная инженерия
_____ И. Н. Обабков
(подпись) (Ф.И.О.)
«26» мая 2025 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
Пояснительная записка

РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ МОДУЛЯ «УПРАВЛЕНИЕ ЗАЯВКАМИ» ВЕБ-СЕРВИСА «ТОЧКА СБОРА» ООО НПЦ «РИЦ»

Руководитель _____ Д. С. Смирнов
Нормоконтроль _____ В. Н. Васина
Обучающийся группы РИ-410944 _____ А. Н. Трефелова

Екатеринбург 2025

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования «Уральский федеральный университет
имени первого Президента России Б. Н. Ельцина»

Институт радиоэлектроники и информационных технологий – РТФ

Школа бакалавриата

Направление (специальность) 09.03.04 Программная инженерия

УТВЕРЖДАЮ
Руководитель образовательной программы
09.03.04 Программная инженерия
_____ И. Н. Обабков
(подпись) (Ф.И.О.)
«09» декабря 2024 г.

ЗАДАНИЕ

на выполнение выпускной квалификационной работы

обучающегося Трефеловой Алины Николаевны группы РИ-410944
(фамилия, имя, отчество)

1 Тема выпускной квалификационной работы
Разработка клиентской части модуля «Управление заявками» веб-сервиса «Точка сбора» ООО НПЦ «РИЦ»

Утверждена распоряжением по институту от «02» декабря 2024 г. № 33.02-05/334

2 Руководитель Смирнов Денис Сергеевич, ассистент кафедры ИИТ ИнФО
(Ф.И.О., ученая степень, ученое звание, должность)

3 Исходные данные к работе
Нормативная, учебная, методическая, периодическая литература по теме ВКР, материалы, полученные в ходе преддипломной практики, требования к системе от заказчика

4 Содержание пояснительной записки (перечень подлежащих разработке вопросов)
1 Анализ процесса набора студентов на практики и стажировки
1.1 Процесс цифровизации взаимодействия с клиентами в контексте набора на производственные практики и стажировки
1.2 Обзор существующих программных решений
1.3 Целевая аудитория, проблемы и требования
2 Проектирование и реализация клиентской части модуля «Управление заявками»
2.1 Определение требований к клиентской части
2.2 Функциональные требования
2.3 Структура интерфейса и навигация
2.4 Технологический стек и выбор инструментов для разработки
2.5 Архитектура и реализация клиентской части
2.6 Тестирование и внедрение решения

5 Перечень демонстрационных материалов
Презентация проекта, демонстрация работы клиентской части модуля, логика системы, частное техническое задание, акт к внедрению

6 Календарный план

Наименование этапов выполнения работы	Срок выполнения этапов работы	Отметка о выполнении
Анализ предметной области и аналогов	до 14.02.2025 г.	выполнено
Определение требований к клиентской части	до 21.03.2025 г.	выполнено
Реализация пользовательского интерфейса	до 11.04.2025 г.	выполнено
Интеграция с серверной логикой	до 25.04.2025 г.	выполнено
Тестирование и внедрение	до 06.05.2025 г.	выполнено
Оформление пояснительной записки	до 12.05.2025 г.	выполнено

Руководитель _____ Смирнов Денис Сергеевич
(подпись) (Ф.И.О.)

Задание принял к исполнению 06.12.2024 г. _____
(дата) (подпись обучающегося)

7 Выпускная квалификационная работа закончена «12» мая 2025 г.

Пояснительная записка и все материалы просмотрены
Считаю возможным допустить Трефелову Алину Николаевну к защите её выпускной квалификационной работы в Государственной экзаменационной комиссии.

Руководитель _____
(подпись)

8 Допустить Трефелову Алину Николаевну к защите выпускной квалификационной работы в Государственной экзаменационной комиссии (распоряжение по институту № 33.02-05/165 от «26» мая 2025 г.)

Руководитель образовательной программы
09.03.04 Программная инженерия _____ И. Н. Обабков
(подпись) (Ф.И.О.)

РЕФЕРАТ

Выпускная квалификационная работа бакалавра 57 с., 21 рис., 5 табл., 26 источн., 6 прил.

АВТОМАТИЗАЦИЯ ОБРАБОТКИ ЗАЯВОК, УПРАВЛЕНИЕ ЗАЯВКАМИ, WEB-РАЗРАБОТКА, КЛИЕНТСКАЯ ЧАСТЬ, ПРОЕКТНЫЕ ШКОЛЫ, FRONTEND-РАЗРАБОТКА.

Объект исследования – процесс обработки заявок студентов на стажировки и практики в рамках проектных школ НПЦ «РИЦ».

Предмет исследования – реализация клиентской части модуля «Управление заявками», включающая пользовательские интерфейсы и взаимодействие с серверной логикой системы.

Цель – реализовать клиентскую часть модуля «Управление заявками», обеспечивающего автоматизацию всех этапов обработки заявок.

Методы исследования: анализ, дедукция и создание.

Результатом работы стало создание клиентского модуля «Управление заявками», обеспечивающего взаимодействие между студентами, организаторами и руководителями.

Выпускная квалификационная работа выполнена в текстовом редакторе и представлена в электронном виде.

СОДЕРЖАНИЕ

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ 6
ВВЕДЕНИЕ 7
1 Анализ процесса набора студентов на практики и стажировки 9
1.1 Процесс цифровизации взаимодействия с клиентами в контексте набора на производственные практики и стажировки 9
1.2 Обзор существующих программных решений 12
1.3 Целевая аудитория, проблемы и требования 17
2 Проектирование и реализация клиентской части модуля «Управление заявками» 19
2.1 Определение требований к клиентской части модуля 19
2.2 Функциональные требования 23
2.3 Структура интерфейса и навигация 24
2.4 Технологический стек и выбор инструментов для разработки 27
2.5 Архитектура и реализация клиентской части 32
2.6 Тестирование и внедрение решения 50
ЗАКЛЮЧЕНИЕ 52
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 55
ПРИЛОЖЕНИЕ А (справочное) Схема декомпозиции бизнес-процессов 58
ПРИЛОЖЕНИЕ Б (справочное) Схема связи данных и интерфейсов 59
ПРИЛОЖЕНИЕ В (обязательное) Частное техническое задание 62
ПРИЛОЖЕНИЕ Г (справочное) Элементы навигации по сайту 71
ПРИЛОЖЕНИЕ Д (обязательное) Реализованные интерфейсы 72
ПРИЛОЖЕНИЕ Е (справочное) Таблица с тестированием модуля 85

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

В настоящей выпускной квалификационной работе применяются следующие термины с соответствующими определениями:

API [1]	набор правил (интерфейс) и протоколов, которым соглашаются следовать две отдельные системы или программы — одна на стороне публикации, а другая на стороне доступа, позволяющих приложениям взаимодействовать друг с другом
Фронтенд (frontend) [2]	часть веб-приложения, которая обеспечивает отображение информации и взаимодействие с пользователем
Бэкенд (backend) [2]	эта часть отвечает за обработку данных и выполнение логики на сервере
CSS	язык стилей, который используется для описания внешнего вида и форматирования HTML-документов, позволяя изменять различные визуальные параметры
DOM	структура представления HTML-документа в виде объектов, которая позволяет скриптам изменять содержимое, структуру и стили веб-страницы
Пропсы (props)	механизм передачи данных от родительского компонента к дочернему в React-компонентах
Drag-and-drop	функциональность, позволяющая пользователю перемещать элементы интерфейса с помощью перетаскивания
Эндпойнт	URL-адрес или маршрут в API, который отвечает за выполнение определённого действия
Инвалидация	процесс пометки данных как устаревших или недействительных, чтобы при следующем запросе они были перезапрошены с сервера
Авторизация	процесс проверки прав пользователя на доступ к определённым данным или действиям в системе
Роунтинг	механизм в веб-приложениях для определения пути и отображения соответствующего компонента или страницы при переходе по определённому URL
Рефетч	повторное выполнение запроса для обновления данных, актуализации информации или синхронизации с сервером.

ВВЕДЕНИЕ

В условиях стремительной цифровизации современного общества автоматизация процессов взаимодействия с пользователями становится неотъемлемой частью эффективного и устойчивого функционирования организаций. Всё больше сфер переходят к использованию цифровых инструментов, что позволяет не только оптимизировать внутренние процессы, но и значительно улучшить качество предоставляемых услуг. Внедрение современных цифровых решений способствует снижению затрат времени и ресурсов, повышению точности обработки информации, систематизации данных, а также обеспечивает более оперативное реагирование на запросы со стороны пользователей и клиентов [3].

Особую актуальность данная проблема приобретает в сфере образовательных программ и проектных школ, ориентированных на развитие практических навыков студентов. В таких проектах обработка заявок на участие играет ключевую роль в организации процессов обучения и стажировок. С ростом числа участников, направлений и проектов возрастает нагрузка на организаторов, что делает традиционные методы работы с данными всё менее эффективными. Применение инструментов, таких как Google Forms и электронные таблицы, безусловно, на начальном этапе обеспечивает определённый уровень систематизации, однако в условиях масштабируемого проекта они начинают демонстрировать свои ограничения. К числу таких недостатков можно отнести необходимость ручной обработки информации, высокую вероятность ошибок при вводе и переносе данных, сложности в отслеживании текущего статуса заявок, а также отсутствие гибкой интеграции с другими сервисами, что существенно снижает уровень прозрачности и управляемости процессов.

В этих условиях использование автоматизированных систем становится логичным шагом в направлении повышения эффективности. Такие системы позволяют не только разгрузить организаторов от рутинных задач, но и

предоставить участникам более удобный, прозрачный и предсказуемый путь взаимодействия с платформой, обеспечивая при этом надёжное хранение и обработку всей необходимой информации.

Таким образом, проблема исследования заключается в необходимости полноценной автоматизации процесса обработки заявок в рамках проектных школ НПЦ «РИЦ».

Объектом исследования является процесс обработки заявок студентов на стажировки и практики в рамках проектных школ НПЦ «РИЦ».

Предметом исследования является реализация клиентской части модуля «Управление заявками», включающая пользовательские интерфейсы и взаимодействие с серверной логикой системы.

Целью данной работы является разработка клиентской части модуля **«**Управление заявками**»**, который обеспечит автоматизацию всех этапов обработки заявок — от подачи до мониторинга.

Для достижения поставленной цели были поставлены следующие задачи:

– рассмотреть предметную область и определить методы и инструменты разработки клиентской части;

– разработать пользовательские интерфейсы, учитывая функциональные и нефункциональные требования;

– реализовать взаимодействие клиентской части с серверной логикой системы для автоматизации процессов обработки заявок.

1 Анализ процесса набора студентов на практики и стажировки

1.1 Процесс цифровизации взаимодействия с клиентами в контексте набора на производственные практики и стажировки

Цифровизация взаимодействия с клиентами - это процесс интеграции информационных технологий для улучшения коммуникации и автоматизации бизнес-процессов. В сфере организации стажировок и практик данный процесс особенно актуален для обработки заявок студентов, поскольку позволяет повысить оперативность, прозрачность и контроль над стадиями подачи, рассмотрения и принятия решений.

Традиционные методы обработки заявок, как правило, происходят вручную и включают множество этапов: от сбора данных через различные формы до последующей ручной обработки заявок и их распределения между ответственными лицами. Это приводит к нескольким ключевым проблемам:

– ошибки, связанные с вручную введенными данными, могут привести к потере важной информации или неверному распределению заявок;

– каждый этап требует значительных временных и человеческих ресурсов для проверки, обработки и ввода данных вручную;

– процесс, основанный на человеческом вмешательстве, гораздо медленнее, чем автоматизированный, что увеличивает время отклика для студентов и уменьшает общую продуктивность;

– при увеличении количества заявок все эти процессы становятся труднопереводимыми на большие объемы, что приводит к задержкам и снижению эффективности.

Автоматизация процессов позволяет эффективно решить эти проблемы. Применение современных технологий позволяет значительно сократить количество ручной работы, ускоряя обработку заявок на каждом этапе. В отличие от традиционных методов, где заявки обрабатываются вручную через электронные таблицы или в бумажной форме, автоматизация значительно упрощает процесс.

Существует несколько подходов к автоматизации процессов обработки заявок, которые активно используются в различных отраслях. Одним из самых популярных является внедрение CRM-систем, которые позволяют централизованно управлять заявками, их статусами и взаимодействовать с пользователями. CRM-системы могут быть настроены под конкретные нужды организации, автоматизируя различные аспекты обработки заявок. Они помогают систематизировать заявки, контролировать их обработку и поддерживать актуальность данных.

Другим важным инструментом является использование web-форм для подачи заявок. Такие формы структурируют все необходимые данные и интегрируются с backend-системами, что позволяет автоматически перенаправлять заявки для дальнейшей обработки. Это не только ускоряет процесс, но и упрощает взаимодействие с пользователем.

Чат-боты и роботы также играют свою роль в автоматизации: они могут проводить первичную фильтрацию заявок, предоставлять студентам нужную информацию или автоматически перенаправлять заявки в зависимости от их категории.

Таким образом можно выделить ключевые преимущества автоматизации:

– сокращение времени обработки - автоматизация позволяет значительно ускорить процесс обработки заявок, освободив время сотрудников от рутинных задач, так как все данные собираются в одном месте, без необходимости их ручного ввода и сортировки;

– уменьшение ошибок - автоматические системы, такие как CRM, исключают человеческий фактор, минимизируя риски ошибок в данных;

– гибкость и масштабируемость - в отличие от ручных процессов, автоматизация может легко масштабироваться - при увеличении числа заявок не потребуется увеличивать количество сотрудников, а все процессы будут выполняться одинаково эффективно независимо от объема;

– быстрая обратная связь - автоматизированные системы могут

немедленно оповещать студентов о статусе их заявки или необходимости в дальнейших действиях.

В таблице 1.1 представлен анализ выявленных проблем и способов их решения посредством автоматизации. Это позволяет систематизировать рассмотренные ранее аспекты и подчеркнуть значимость цифровизации в контексте обработки заявок.

Таблица 1.1 – Сводный анализ проблем и преимуществ автоматизации процессов подачи заявок

Проблема	Влияние на процесс	Возможности автоматизации	Результат внедрения
Ручная обработка большого объема заявок	Высокая нагрузка на персонал, рост количества ошибок	Использование веб-форм, интеграция с CRM	Снижение времени обработки и числа ошибок
Отсутствие прозрачности обработки	Пользователь не знает, что происходит с его заявкой	Система статусов, уведомления, личный кабинет	Повышение доверия, улучшение пользовательского опыта
Потеря или дублирование заявок	Низкое качество обслуживания, пропущенные возможности	Централизованное хранение и уникальная идентификация	Повышение точности, снижение числа дублирующих заявок
Длительная обратная связь	Снижение удовлетворённости студентов и партнёров	Автоматические уведомления, чат-боты	Повышение скорости реакции, рост лояльности

На основе вышесказанного можно сделать вывод, что автоматизация - это необходимое условие для повышения эффективности обработки заявок в условиях большого потока данных. Она решает проблемы, связанные с человеческим фактором, ускоряет работу и улучшает качество взаимодействия с пользователями. Внедрение таких технологий, как CRM-системы и автоматизированные веб-формы, позволяет значительно повысить производительность и снизить риск ошибок.

1.2 Обзор существующих программных решений

1.2.1 Анализ технологий для реализации интерфейсов

Существует множество программных решений, которые могут помочь в создании эффективной системы для обработки заявок студентов, включая фреймворки, библиотеки и подходы для отображения данных, взаимодействия с API и других важных аспектов. Рассмотрим основные технологии для реализации клиентской части.

React.js - наиболее популярный фреймворк [4], его основное преимущество - это компонентный подход, который позволяет создавать повторно используемые элементы UI и эффективно управлять состоянием приложения, благодаря чему React идеально подходит для реализации интерфейсов с большим количеством динамических элементов, таких как формы для подачи заявок и отображения их статусов.

Vue.js - еще один фреймворк для создания пользовательских интерфейсов, который часто используется для разработки легковесных и быстрых приложений, его основные преимущества - это простота в обучении и гибкость.

Angular - комплексный фреймворк, который предоставляет все необходимое для создания масштабируемых приложений с продвинутым управлением состоянием, маршрутизацией и интеграцией с API, однако обладает строгой структурой, из-за чего подходит для более крупных и сложных проектов.

Одним из ключевых аспектов при разработке клиентской части является интеграция с серверной логикой. Важно, чтобы данные, поступающие от пользователей (например, информация о заявках или изменении их статусов), корректно передавались и обрабатывались на сервере. Для этого могут быть использованы различные решения.

REST API - это классический способ обмена данными между клиентом

и сервером, он позволяет отправлять и получать данные в формате JSON, что является удобным для веб-приложений, особенно для динамичного обновления информации о заявках.

GraphQL - альтернатива REST, позволяющая делать запросы на сервер с более гибкой структурой, запрашивая только необходимые данные, что делает его удобным для работы с большими объемами данных.

WebSockets - для реализации возможности мгновенных уведомлений о статусе заявки и других важных обновлений, можно использовать WebSockets, что позволит клиенту получать обновления в реальном времени без необходимости обновлять страницу.

Для удобства отображения данных и взаимодействия с пользователем используется несколько популярных библиотек.

Material-UI - библиотека компонентов для React, которая предоставляет готовые элементы UI, стилизованные в соответствии с принципами Material Design.

Ant Design - еще одна популярная библиотека компонентов для React, которая ориентирована на создание корпоративных приложений с широким набором готовых компонентов, таких как таблицы, формы и кнопки [5].

Chart.js и Recharts - библиотеки для отображения графиков и диаграмм, которые могут быть полезны для визуализации прогресса студентов, статистики по заявкам и других данных.

Важной частью интерфейса является работа с заявками и отображение их данных. Существует несколько решений, которые могут помочь в удобной визуализации информации.

Канбан-доски - отличный инструмент для визуализации процесса обработки заявок, в них заявки отображаются в виде карточек, которые можно перетаскивать по колонкам, отражающим статус заявки, для их реализации на фронтенде можно использовать React и Redux.

Диаграммы и графики - для отображения статистики по заявкам можно использовать графики и диаграммы, которые помогут отслеживать процесс

работы с заявками и показывать информацию о статусах, количестве заявок по категориям и так далее, в их разработке помогут такие библиотеки как как Chart.js или Recharts.

Модальные окна и всплывающие элементы - для отображения дополнительной информации о заявке или работе с ней можно использовать модальные окна, которые открываются по клику.

Как было упомянуто выше, автоматизация обработки заявок и взаимодействия с кандидатами может быть достигнута через несколько ключевых решений.

CRM-системы - как правило, используются для централизованного управления заявками и их статусов, для фронтенда можно интегрировать CRM через API, используя такие инструменты, как Axios или Fetch, что позволяет автоматически обновлять данные о статусах заявок и отслеживать их процесс.

Web-формы - для сбора данных от студентов эти формы могут быть интегрированы с бэкенд-системами через REST API или GraphQL, что позволяет автоматизировать процесс обработки и сохранения данных.

Чат-боты и роботы - автоматизация начальных этапов обработки заявок, таких как ответы на часто задаваемые вопросы, фильтрация заявок по категориям или автоматическая отправка уведомлений о статусах.

Таким образом, в ходе анализа были рассмотрены популярные библиотеки и фреймворки, применяемые для создания пользовательских интерфейсов в веб-приложениях. Каждая из представленных технологий обладает своими преимуществами и ориентирована на решение определённого круга задач — от стилизации и визуализации данных до построения комплексных интерфейсных решений. Их гибкость, соответствие современным требованиям и активная поддержка сообщества делают эти инструменты актуальными для проектирования клиентской части информационных систем. Проведённый обзор позволил сформировать целостное представление о существующих подходах и стал основой для последующего выбора технологий.

1.2.2 Анализ готовых решений

При анализе существующих решений, ориентированных на автоматизацию процессов подачи и обработки заявок, особое внимание уделялось реализации клиентской части системы, ее структуре, взаимодействию с пользователем и подходам к организации интерфейсов. Рассмотрим несколько наиболее распространенных систем и их клиентские части.

Одним из первых решений, которое стоит рассмотреть, является Google Forms, система, позволяющая создавать простые формы для сбора данных. Важной особенностью интерфейса является возможность гибкой настройки видимости полей в зависимости от ранее введенных данных, что позволяет организовать несложные многошаговые формы (простые правила если-то). Однако функциональности для интеграции с более сложными бизнес-процессами, например, обработки заявок с этапами и ролями, в Google Forms не хватает. Это ограничивает возможности системы для более сложных процессов автоматизации.

В Bitrix24, напротив, реализован продвинутый механизм управления бизнес-процессами через воронки (pipeline), где каждому этапу процесса можно привязать определенные формы, а также назначить ответственные роли. Это позволяет более гибко управлять заявками, а канбан-доски для визуализации этапов упрощают следование за процессом. Однако такой подход требует большего внимания к настройке и структуре данных, а также более высокой степени кастомизации под специфические нужды пользователя.

Trello использует подход канбан-доски, где можно организовать задачи в виде карточек, но для более сложных сценариев подачи заявок нужно подключать дополнительные плагины или Power-Ups. Это позволяет расширить функциональность, но увеличивает сложность настройки. Несмотря на это, визуализация процессов в Trello даёт хорошее представление

о статусе заявок.

Не стоит забывать и о специализированных low-code решениях, таких как Retool и Form.io, которые позволяют с минимальными усилиями настраивать как формы, так и серверную логику. В этих системах клиентская часть объединена с серверной частью через drag-and-drop интерфейс, что значительно ускоряет процесс разработки и адаптации под конкретные нужды. Однако такой подход также требует определённой квалификации для настройки сложных взаимодействий и автоматизации.

В процессе анализа были рассмотрены пять решений, наиболее часто применяемых для организации обработки заявок в веб-приложениях. В таблице 1.2 приведены их ключевые преимущества и ограничения, которые позволяют оценить применимость каждого инструмента в контексте специфики проектируемой системы.

Таблица 1.2 – Преимущества и ограничения рассмотренных решений

Решение	Преимущества	Ограничения
Google Forms	Простота использования, быстрая настройка, интеграция с Google Таблицами	Нет поддержки ролей, слабая автоматизация, не подходит для сложных сценариев
Trello	Визуализация в виде канбан-досок, интуитивный интерфейс	Ограниченная автоматизация, требует Power-Ups для расширения, нет ролей
Bitrix24	Широкие возможности CRM, автоматизация, поддержка ролей	Сложность освоения, избыточность для простых задач
Retool	Гибкая настройка интерфейсов, глубокая автоматизация, поддержка API	Высокий порог входа, требует продвинутых навыков, возможно избыточен
Form.io	Конструктор форм, поддержка ролей и логики на стороне сервера	Необходимость настройки backend-части, ограниченный UI

В рамках нашего проекта необходимо интегрировать лучшие практики из рассмотренных решений для реализации многоступенчатых форм, гибкой системы ролей и этапов. Выбор технологий играет ключевую роль в создании системы, которая будет легко адаптироваться к изменяющимся требованиям.

Использование фреймворков, таких как React, Redux и Ant Design, позволяет создать динамичные и масштабируемые интерфейсы, которые легко интегрируются с серверной логикой через REST API, что обеспечит быстрое обновление данных.

1.3 Целевая аудитория, проблемы и требования

Основной целевой аудиторией данного проекта является заказчик - организатор стажировок и практик, который сталкиваются с необходимостью обработки большого объема заявок от студентов. Главной проблемой, с которой он сталкивается, является низкая степень автоматизации в процессе обработки заявок, что приводит к значительным временным затратам и снижению эффективности работы.

Основной задачей для организатора является полная автоматизация процесса обработки заявок. Система должна обеспечивать возможность гибкой настройки, позволяя автоматически изменять статусы заявок на основе заданных правил и сценариев.

Также для работы с данными важно обеспечить полноту их отображения и управления, позволяя организатору при необходимости эффективно отслеживать статус заявок, фильтровать и сортировать их в реальном времени.

Для реализации этих задач будет использован подход модульного монолита. Это архитектурное решение представляет собой структуру приложения, в которой вся система разрабатывается и разворачивается как единое целое, но логически разделена на отдельные модули, каждый из которых отвечает за определённый функционал. Такой подход позволяет достичь следующих преимуществ:

– простота развертывания - модульный монолит легко развертывается и обновляется, поскольку все компоненты разворачиваются одновременно, без необходимости координировать обновления отдельных сервисов, как в микросервисной архитектуре, что упрощает управление и поддержку;

– гибкость - несмотря на единую кодовую базу, архитектура позволяет структурировать проект по функциональным модулям, благодаря чему в случае необходимости можно изменить или добавить новый модуль, минимально затрагивая другие компоненты системы, что делает разработку более гибкой;

– производительность - в отличие от микросервисов, которые обмениваются данными через сеть, все компоненты модульного монолита взаимодействуют напрямую в рамках одного процесса, что исключает задержки на сетевые запросы и повышает общую скорость обработки данных.

Таким образом, анализ целевой аудитории показал, что основная проблема организатора — это низкий уровень автоматизации процессов обработки заявок, что ведёт к затратам времени и ресурсов на выполнение рутинных задач. Полная автоматизация этих процессов с помощью гибкой настройки системы позволит значительно повысить производительность и точность работы.

Важным аспектом является выбор архитектурного решения. Для данного проекта был выбран подход модульного монолита, который позволяет создать систему с чётким разделением функционала, при этом обеспечивая высокую производительность, гибкость и простоту развертывания.

2 Проектирование и реализация клиентской части модуля «Управление заявками»

2.1 Определение требований к клиентской части модуля

Проект «Уральский центр стажировок», реализуемый ежегодно НПЦ «РИЦ», направлен на организацию и проведение стажировок и практик для студентов в рамках Проектной школы [6, С. 29]. Эти образовательные программы ориентированы на развитие практических навыков студентов в таких областях, как разработка в 1С, веб-разработка и других направлениях. С каждым годом количество участников проекта увеличивается, что приводит к возрастанию нагрузки на организаторов и снижению эффективности текущих процессов.

В настоящее время процесс подачи заявок и их обработки происходит вручную: студенты заполняют формы через Google Forms, а затем организаторы вручную обрабатывают данные с помощью таблиц. Такой подход требует значительных временных затрат и постоянного контроля, что усложняет работу и увеличивает вероятность ошибок.

Для повышения эффективности работы и снижения нагрузки на организаторов была выдвинута необходимость разработки и внедрения специализированной системы, которая автоматизирует процессы подачи и обработки заявок, а также обеспечит удобство и прозрачность всех этапов работы. В рамках этого проекта требуется разработать клиентскую часть системы, которая будет отвечать за удобный интерфейс взаимодействия с пользователями, а также за управление заявками и их статусами.

Для эффективной разработки клиентской части системы аналитиком было проведено распределение функционала по ролям, что позволило чётко разграничить обязанности и действия пользователей. Это распределение стало основой для дальнейшего анализа и проектирования системы, так как на его основе определялись ключевые функции, доступные для каждой роли. Итоговое распределение приведено в таблице 2.1.

Таблица 2.1 – Распределение функционала по ролям

Функционал	Администратор системы	Организатор	Руководитель направлений	Проектант
Управление ролями	Управляет ролями пользователей	Ограниченный доступ к управлению ролями	–	–
Управление мероприятиями	–	Создает, редактирует, удаляет мероприятия и направления	Управляет направлением в рамках мероприятия	–
Управление заявками	–	Создает форму для мероприятия, управляет параметрами приема заявок	Не управляет заявками, следит за их обработкой в рамках направления	Подает заявку, отслеживает статус
Настройки процесса	–	Настраивает этапы и параметры мероприятия для подачи заявок	Следит за процессом в рамках своей роли, но не настраивает этапы	Поддерживает свой процесс подачи заявки, отслеживает события по этапам
Уведомления и сообщения	–	Настраивает уведомления для участников мероприятия	–	Получает уведомления о статусе своей заявки
Редактирование информации	–	Редактирует информацию о мероприятии и направлениях, создает и изменяет описания	Редактирует только информацию в рамках своего направления	Редактирует только информацию о себе

После этого была проведена декомпозиция бизнес-процессов, которая позволила разбить текущие процессы на более мелкие задачи и точно определить, какие именно процессы подлежат автоматизации. Каждая задача была проанализирована, и на основе этого были выделены критические моменты, требующие оптимизации. Пример схемы можно увидеть на рисунке 2.1, а полная диаграмма бизнес-процессов представлена рисунке А.1 приложения А.

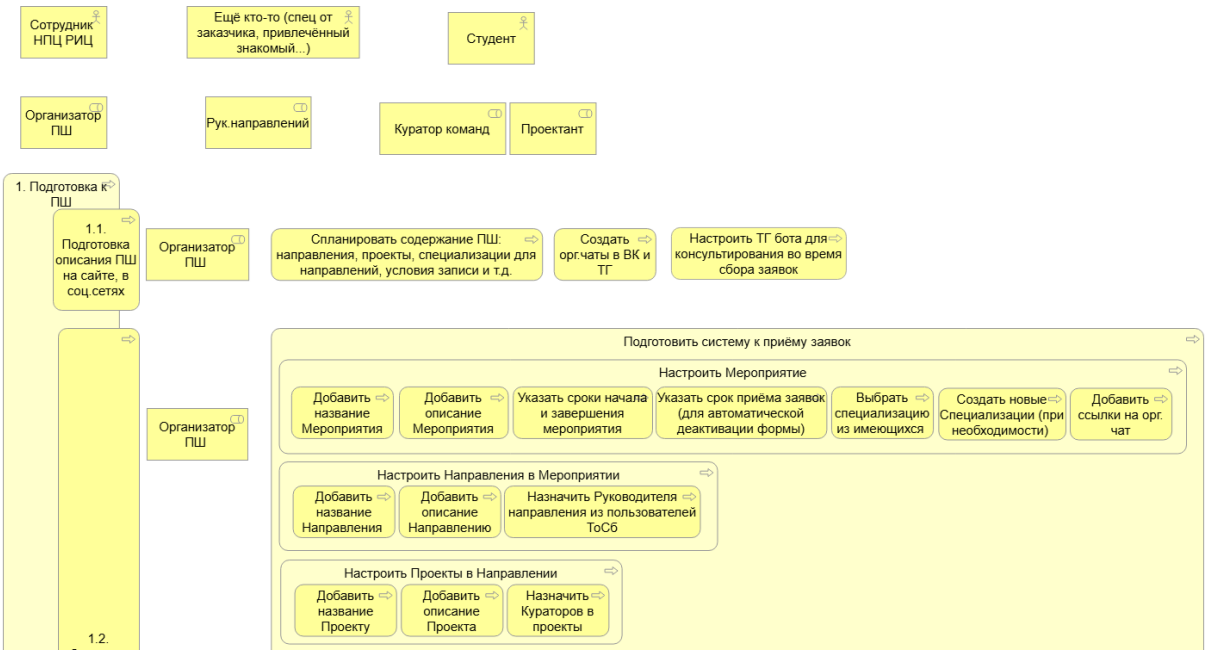

Рисунок 2.1 – Часть схемы декомпозиции бизнес-процесса создания мероприятия в системе – формирование структуры мероприятия

Затем была разработана модель данных, охватывающая все компоненты приложения и базы данных (рисунки Б.1–Б.3 в приложении Б). Моделирование данных включало определение структуры информации, необходимой для работы системы, а также взаимодействие между различными частями данных (например, заявки, статусы, пользователи). Это позволило backend-разработчику создать основу для построения базы данных и дальнейшей работы с данными в реальном времени.

После этого был разработан User Flow - последовательность действий пользователей при взаимодействии с системой. User Flow позволяет точно понять, как пользователь будет перемещаться по интерфейсу и какие шаги он будет предпринимать для выполнения своих задач. Схема отображает логику взаимодействия каждого типа пользователя с системой и помогает понять, как будет организован процесс подачи заявок, управления ими и выполнения других задач [7]. Например, на рисунке 2.2 User Flow для студента описывает процесс подачи заявки: он заходит в личный кабинет, выбирает мероприятие, заполняет анкету, подтверждает отправку заявки и отслеживает ее статус.

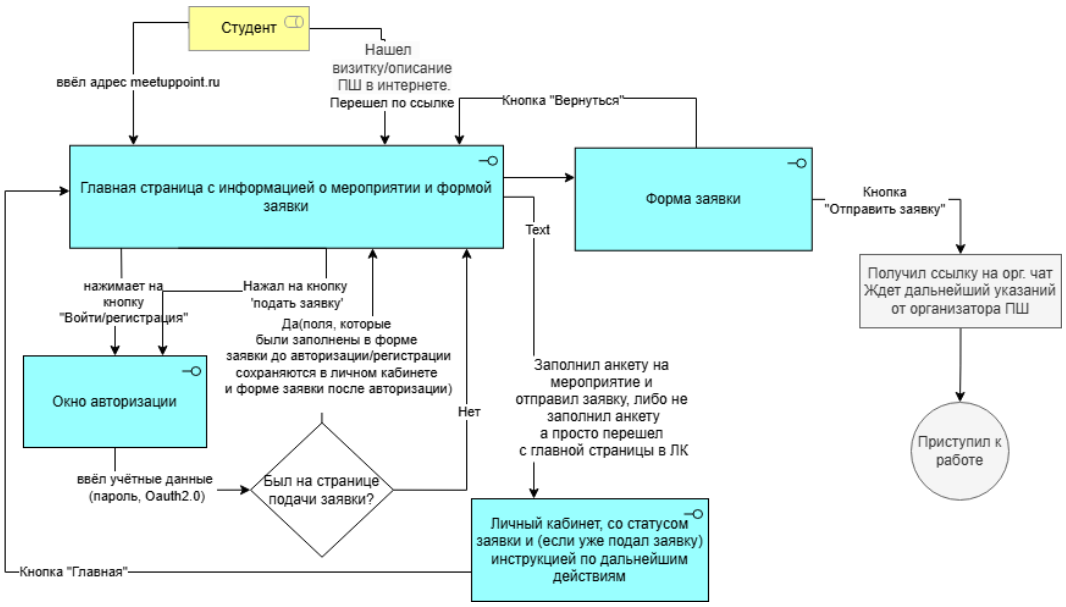

Рисунок 2.2 – Часть схемы User Flow на примере студента

На основе проведённого аналитического исследования и описанных выше этапов, дизайнером были созданы макеты пользовательского интерфейса, которые визуализируют логику взаимодействия пользователей с системой. Эти макеты отражают ключевые элементы интерфейса, такие как формы подачи заявок, отображение статусов, канбан-доски, а также другие элементы, требуемые для реализации функционала системы.

Также для эффективной реализации клиентской части системы «Управление заявками» было составлено частное техническое задание (ЧТЗ) [8]-[9] на основе следующих данных:

– анализ текущего состояния процесса обработки заявок, выявление основных проблем и узких мест;

– результаты обследования бизнес-процессов;

– обзор существующих программных решений и их возможностей для автоматизации аналогичных процессов.

Основной целью данного документа является формализация требований к веб-приложению, что позволяет определить перечень задач, подлежащих выполнению в рамках разработки, а также установить единые критерии оценки результатов.

Подробное содержание частного технического задания представлено в Приложении В, что обеспечивает его доступность для дальнейшего использования в процессе разработки и контроля выполнения работ.

Таким образом, все результаты анализа стали основой для определения требований к клиентской части системы. Эти этапы помогли точно сформулировать задачи для реализации интерфейса, учитывая потребности пользователей и специфику работы с заявками. Основными требованиями стали возможность управлять мероприятиями, заявками, профилями пользователей, а также гибкость в настройке форм подачи заявок, статусов и роботов. В дальнейшем будет реализована детальная настройка и интеграция этих функциональных возможностей в рамках выбранной архитектуры системы.

2.2 Функциональные требования

На основе частного технического задания (см. Приложение В) можно выделить список ключевых функций системы для клиентской части модуля «Управление заявками»:

– регистрация и авторизация через почту и Google аккаунт;

– просмотр и редактирование личного профиля;

– доступ к просмотру профилей других пользователей;

– создание, редактирование и удаление мероприятий;

– отображение списка мероприятий и просмотр детальной информации;

– создание, редактирование и удаление направлений;

– отображение списка направлений;

– создание, редактирование и удаление проектов;

– отображение списка проектов и просмотр детальной информации;

– настройка формы подачи заявки: выбор полей и их порядок:

– создание и редактирование статусов заявок;

– просмотр заявок в виде таблицы и канбан-доски;

– изменение статусов заявок и завершение приема заявок;

– автоматическое заполнение данных о студенте с возможностью их корректировки;

– отправка заявки через настроенную форму.

2.3 Структура интерфейса и навигация

Процесс проектирования пользовательского интерфейса для модуля «Управление заявками» был основан на функциональных требованиях, сформулированных ранее и User Flow. В результате этого этапа были определены основные страницы и элементы навигации, которые должны быть доступны пользователю. Полный список страниц представлен в таблице 2.2.

Таблица 2.2 – Список страниц

№	Название	Путь	Описание
1	2	3	4
1	Главная страница	/	Приветствие, отображается информация о системе и компании.
2	Вход	/login	Авторизация пользователя в системе с использованием почты или Google
3	Регистрация	/register	Создание нового аккаунта пользователя через почту или Google
4	Подтверждения email	/email-verified	Подтверждает успешную верификацию email-адреса пользователя после регистрации
5	Сброс пароля	/password-reset-request	Возможность может запросить сброс пароля
6	Подтверждения сброса пароля	/password-reset/confirm	Подтверждение нового пароля после запроса на его сброс
7	Профиль пользователя	/profile	Просмотр и редактирование своего профиля
8	Профиль другого пользователя	/profile/:id	Страница, на которой отображается профиль другого пользователя.

Продолжение таблицы 2.2

1	2	3	4
9	Список мероприятий	/events	Отображение списка мероприятия
10	Список направлений	/directions	Отображение списка направлений
11	Список проектов	/projects	Отображение списка проектов
12	Страница мероприятия	/event/:id	Подробная информация о конкретном мероприятии
13	Страница проекта	/project/:id	Подробная информация о конкретном проекте
14	Настройка мероприятия	/event-setup	Настройка мероприятия перед созданием
15	Настройка направлений	/directions-setup	Создание и редактирование направлений внутри мероприятия
16	Настройка проектов	/projects-setup	Создание и редактирование проектов внутри мероприятия
17	Подтверждение настроек мероприятия	/event-setup-save	Подтверждение всех выбранных настроек мероприятия перед его сохранением
18	Редактирование мероприятия	/event/:id/edit	Редактирование раннее созданного мероприятия
19	Управление заявками	/requests	Управления заявками на мероприятия

Для обеспечения удобной навигации в клиентской части разрабатываемого модуля предусмотрены элементы управления пользовательским перемещением между различными разделами и функциональными модулями. К числу этих элементов относятся:

– боковая панель навигации (рисунок Г.1 в приложении Г) - основной инструмент перехода между основными разделами системы после авторизации (мероприятия, направления, проекты, заявки и т. д.);

– верхняя панель (header) (рисунки Г.2-Г.3 в приложении Г) -

расположена в верхней части интерфейса и включает элементы управления, относящиеся к пользовательскому профилю, а также кнопки выхода и авторизации;

– контекстные кнопки перехода - размещаются в интерфейсе страниц (например, "Создать мероприятие", "Редактировать", "Подробнее") и обеспечивают навигацию между связанными действиями;

– ссылки - используются для переходов между связанными страницами (например, переход от профиля к заявке);

– навигация по вкладкам - используется для переключения вкладок внутри одной группы страниц.

Также была учтена структура пользовательского интерфейса, обеспечивающая предсказуемость поведения системы (Рис. 2.3). Каждый интерфейс построен на едином композиционном шаблоне, включающем следующие основные элементы:

1) функциональная шапка - содержит глобальные элементы навигации: логотип, ссылки на основные разделы, доступ к профилю пользователя и действия типа выхода из системы. В отдельных случаях в шапке могут быть размещены дополнительные управляющие элементы, зависящие от роли пользователя.

2) контейнер страницы - ограниченная по ширине центральная зона, в которой размещается содержимое. Он включает в себя:

а. внутреннюю шапку раздела — заголовок текущей страницы (например, «Список мероприятий»), а также контекстные элементы управления: кнопки действий (например, «Создать»), фильтры, поиск, выпадающие списки.

б. основную содержательную область — таблицы, формы, карточки, канбан-доски или профили, в зависимости от текущего маршрута.

3) системные сообщения и модальные окна - отображаются поверх основного интерфейса для взаимодействий, требующих подтверждения или дополнительного ввода данных.



Рисунок 2.3 – Пример структуры пользовательских интерфейсов

Единообразие структуры всех страниц и повторяемость интерфейсных паттернов позволяют пользователю быстрее адаптироваться в системе.

Разработка структуры пользовательского интерфейса и системы навигации была основана на анализе функциональных требований и предполагаемых пользовательских сценариев. Основное внимание было уделено доступу к ключевым разделам и логичности маршрутизации.

2.4 Технологический стек и выбор инструментов для разработки

2.4.1 Обоснование выбора стека технологий

Разработка клиентской части модуля была осуществлена с использованием набора технологий, представленного в таблице 2.3. Выбор каждой технологии был основан на анализе различных вариантов, рассмотренных в разделе 1.2.1 «Анализ технологий для реализации интерфейсов». В ходе этого анализа были оценены подходящие технологии с точки зрения их функциональных возможностей, совместимости с другими компонентами системы, а также особенностей конкретного проекта. В результате были выбраны те инструменты и фреймворки, которые наилучшим образом соответствуют поставленным задачам и позволяют эффективно реализовать требуемую функциональность.

Таблица 2.3 – Технологический стек

Название	Описание
TypeScript	Строго типизированный язык программирования, основанный на JavaScript
React	Библиотека для построения пользовательских интерфейсов с компонентной архитектурой
Vite	Сборщик модулей с быстрым стартом и поддержкой мгновенных обновлений
Redux Toolkit	Библиотека для управления состоянием приложения, построенная на Redux
RTK Query	Библиотека для работы с серверными запросами (API), кэшированием и синхронизацией состояния
React Router	Библиотека для организации навигации в React-приложениях
SCSS	Препроцессор для CSS, расширяющий возможности стандартного CSS (переменные, вложенность и т. д.)
React Context API	API для управления состоянием в приложении React, позволяющее передавать данные между компонентами без необходимости использования props
Ant Design	UI-библиотека компонентов для React, предоставляющая готовые элементы интерфейса, такие как кнопки, таблицы, формы и т. д.
dayjs и date-fns	Библиотеки для работы с датами и временем, предлагающие API для парсинга, форматирования и манипулирования датами
react-beautiful-dnd	Библиотека для реализации функционала перетаскивания элементов в интерфейсе (drag and drop) в приложениях React
Npm	Менеджер пакетов для JavaScript, используемый для установки и управления зависимостями в проекте

Таким образом, выбранный стек технологий обеспечивает эффективную реализацию клиентской части модуля с учетом требований по функциональности, масштабируемости и удобству разработки. Кроме того, использование современных инструментов позволяет повысить читаемость и поддерживаемость кода.

2.4.2 Применение библиотек и инструментов для реализации функционала

В рамках реализации клиентской части системы были использованы библиотеки и инструменты, обеспечивающие гибкость, читаемость кода и масштабируемость проекта.

В качестве основы для построения интерфейсов был выбран React, который позволяет использовать компонентный подход для разработки. Благодаря этому стало возможным организовать отдельные компоненты для каждой части интерфейса, например, для создания и редактирования мероприятий, отображения заявок и профилей пользователей. React эффективно управляет состоянием и обновлением DOM, что улучшает производительность и снижает количество ререндеров [4].

TypeScript значительно улучшил работу с пропсами, состоянием компонентов и API-ответами. С помощью строгой типизации стало возможным избежать многих типов ошибок на этапе компиляции, что упростило поддержку и отладку кода [10]. Это особенно важно при работе с большими и сложными компонентами, такими как формы для создания и редактирования мероприятий или обработки заявок. В качестве примера можно привести интерфейс для работы с проектами (Рис. 2.4), который четко определяет типы данных.

```typescript
/**
 * Интерфейс для описания проекта.
 */
export interface Project {
  project_id?: number; // Идентификатор проекта (опционально)
  directionSet: number; // Идентификатор набора направлений
  name: string; // Название проекта
  description: string | null; // Описание проекта (может быть пустым)
  curatorsSet: number[]; // Массив идентификаторов кураторов проекта
}
```

Рисунок 2.4 – Интерфейс для представления данных проекта

Для организации маршрутизации в приложении был использован React Router. Это решение не только позволяет организовать основные пути, но и обеспечивает защиту маршрутов на основе ролей пользователей [11]. Например, организаторам доступны страницы для управления заявками, создания и редактирования мероприятий, настройки направлений и проектов (Рис. 2.5).

```
{/* Доступ только организатору */}
  <Route element={<RequireRole allowedRoles={['organizer']} />}>
    <Route path="/requests" element={<RequestsManagement />} />
    <Route path="/create-new-specialization" element={<CreateSpecializationForm />} />
    <Route path="/event-setup" element={<EventForm />} />
    <Route path="/event/:id/edit" element={<EventForm />} />
    <Route path="/directions-setup" element={<SetupDirectionForm/>} />
    <Route path="/projects-setup" element={<ProjectForm />} />
    <Route path="/stages-setup" element={<StagesPage />} />
    <Route path="/event-setup-save" element={<EventSetupSummary />} />
  </Route>
```

Рисунок 2.5 – Пример реализации маршрутов, доступных только организаторам

Для управления состоянием приложения был выбран Redux Toolkit, который с помощью RTK Query упрощает работу с API. В рамках приложения Redux используется для хранения данных о текущем пользователе, информации о заявках, а также для процесса создания и редактирования мероприятий. RTK Query позволил автоматически генерировать хуки для работы с API, что ускорило процесс интеграции с сервером и управление кешированием данных [12]. Например, при запросе данных о заявке или мероприятии, RTK Query автоматически кэширует результаты, уменьшая количество повторных запросов к серверу [13].

Для реализации системы пользовательских ролей был использован React Context API. С его помощью можно легко передавать информацию о текущем пользователе и его ролях между различными компонентами приложения, не прибегая к глобальному состоянию Redux [14]. Это позволило упростить архитектуру и сделать код более чистым.

Для создания и стилизации компонентов интерфейса был выбран Ant Design. Библиотека предоставила готовые элементы для работы с таблицами, формами, модальными окнами, кнопками и другими элементами UI [5]. Например, для создания форм подачи заявок и редактирования мероприятий были использованы компоненты формы Ant Design, что позволило значительно ускорить разработку.

Для стилизации компонентов использовался SCSS (Sass), что позволило организовать модульные и легко переиспользуемые стили. Это было особенно важно при разработке тем и переменных для различных элементов интерфейса. SCSS помог избежать дублирования стилей и улучшил читаемость кода [15]. Например, переменные для цветов (Рис. 2.6) и шрифтов были вынесены в отдельные файлы, что упростило настройку темы.



Рисунок 2.6 – Пример использования SCSS-переменных для реализации цветовой палитры проекта

Для работы с датами были использованы библиотеки dayjs и date-fns. Они позволили выполнять преобразование и форматирование дат, что было важно для отображения сроков заявок, мероприятий и других данных [16]. Например, для отображения даты подачи заявки и даты мероприятия использовались функции форматирования из dayjs, что обеспечило единообразие и правильное отображение дат в разных частях приложения.

Для реализации функционала drag-and-drop (перетаскивания) был

использован react-beautiful-dnd. Эта библиотека была применена для создания канбан-досок, где пользователи могут перетаскивать заявки между различными статусами, а также стали частью настройки мероприятия [17].

В качестве сборщика проекта использовался Vite. В конфигурационном файле vite.config.ts настраивались алиасы, оптимизация сборки, а также поведение для разных окружений (например, для разработки и продакшн) [18]. Это обеспечило высокую скорость сборки и быструю загрузку приложения.

Для установки и управления зависимостями был использован Npm. С помощью Npm были установлены все необходимые библиотеки и инструменты, что обеспечило легкость в обновлении зависимостей и добавлении новых библиотек [19].

Для взаимодействия с сервером использовался формат данных JSON [20]. Это позволило легко обмениваться данными между клиентом и сервером, например, при отправке заявок или получении информации о мероприятиях.

2.5 Архитектура и реализация клиентской части

2.5.1 Структура проекта и организация кода

Для создания поддерживаемой кодовой базы в проекте была использована модульная структура [21]. Это позволяет разделить функциональные части системы на независимые, но взаимосвязанные модули, что облегчает разработку и поддержку проекта на протяжении всего его жизненного цикла.

Проект организован в несколько основных директорий, каждая из которых отвечает за свою часть функционала:

src - основная директория, содержащая исходный код проекта.

– App/ включает основные файлы конфигурации и инициализации приложения, такие как API-слайс, конфигурации, хранилище данных и основные компоненты пользовательского интерфейса (UI);

– assets/: содержит все статичные ресурсы, такие как изображения и

иконки, используемые в приложении;

– Components/: включает все компоненты интерфейса, разделённые на категории:

а. Common/ - общие элементы UI, такие как кнопки и уведомления;

б. Forms/ - различные поля ввода для форм;

в. PageComponents/ - компоненты, используемые для отображения данных на страницах, такие как панели и модальные окна;

г. Pagination/ - компоненты для пагинации данных;

д. Sections/ - компоненты, которые используют другие компоненты для отображения больших частей страницы;

е. Selectors/ - компоненты для выбора значений;

– Features/: представляет собой разделение по функциональности для работы с состояниями:

а. ApiSlices/ - для управления API-запросами через Redux Toolkit;

б. Auth/ - для обработки логики аутентификации;

в. context/ - контексты для работы с глобальным состоянием;

г. store/ - конфигурация хранилища состояния приложения;

д. utils/ - вспомогательные функции

– Pages/: компоненты, предназначенные для целых страниц, каждая папка в этом разделе отвечает за свою страницу в приложении;

– Styles/: содержит глобальные стили и специфические стили для компонентов и страниц.

Каждый модуль в проекте организован по принципу разделения ответственности. Для каждого функционального блока создаются следующие основные элементы:

– компоненты - каждый UI-компонент описывает определённую часть пользовательского интерфейса, что помогает минимизировать взаимозависимости и облегчить переиспользование компонентов;

– стили - для каждого компонента, страницы или группы страниц существует отдельный файл стилей, что улучшает их поддержку и облегчает

внесение изменений;

– утилиты и вспомогательные функции - общие функции и утилиты, которые используются в нескольких местах;

– типы и модели данных - все типы данных и интерфейсы используются для определения структуры объектов данных;

– Redux-слайсы и хуки для работы с API - каждый слайс Redux управляет состоянием для определённого функционала, а хуки используются для взаимодействия с сервером, что позволяет централизованно управлять состоянием приложения и обеспечивать лёгкое взаимодействие с сервером.

Визуальная схема на Рисунке 2.7 демонстрирует взаимодействие всех перечисленных элементов в рамках одного функционального блока страницы. В центре находится страница, которая включает контейнер и компоненты. Вспомогательные функции, типы данных, API-слайсы и стили взаимодействуют с компонентами, обеспечивая их работу и доступ к данным.

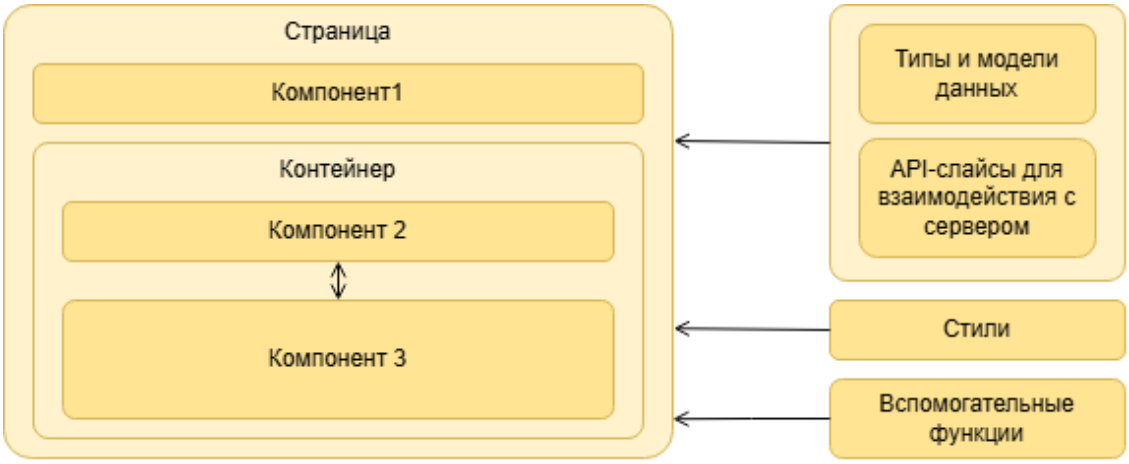

Рисунок 2.7 – Общая структура компонентов функционального блока

Для улучшения взаимодействия в команде и облегчения понимания кода новым разработчикам в проекте внедрены принципы документирования с использованием JSDoc [22]. Основные цели документирования:

– облегчить понимание кода новым разработчикам;

– ускорить процесс внесения изменений и исправлений;

– улучшить взаимодействие между членами команды.

Принципы документирования:

– использование аннотаций @param, @returns и @example;

– описание предназначения компонентов и их основных свойств;

– регулярное обновление документации по мере изменений в коде.

Каждый компонент и функция имеют соответствующие комментарии с описанием входных и выходных данных, а также примером использования, что помогает не только поддерживать код, но и быстро ориентироваться в проекте при необходимости изменений. Пример документирования кода:

```
/**
 * Компонент для отображения иконки с всплывающей подсказкой при наведении.
 *
 * @component
 * @example
 * // Пример использования
 * <InfoCircle text="Это текст подсказки!" />
 *
 * @param {Object} props - Свойства компонента.
 * @param {string} props.text - Текст, который будет отображаться в подсказке при наведении.
 * @returns {JSX.Element} Компонент иконки с подсказкой.
 */
```

Таким образом, проект использует модульную структуру, что облегчает организацию и поддержку кода. Такая структура способствует лучшей изоляции компонентов, улучшает читаемость кода и упрощает масштабирование системы. Внедрение JSDoc и стандартизация документирования ускоряют процесс вникания в проект и взаимодействие внутри команды. В итоге, такая организация позволяет не только эффективно работать с текущими задачами, но и гибко расширять проект в будущем.

2.5.2 Организация взаимодействия с серверной частью и модель хранения данных

Взаимодействие с сервером в приложении организовано через RTK Query, что позволяет централизованно управлять запросами, их состоянием и обработкой данных. Для этого используются API-слайсы, в которых описываются все эндпоинты и асинхронные операции. Это решение исключает дублирование кода и упрощает управление взаимодействиями с сервером.

Модель хранения данных в приложении также организована через Redux-слайсы, где для каждой сущности (например, пользователя, мероприятия, заявки) создаются отдельные слайсы для управления состоянием. Эти слайсы хранят данные, связанные с соответствующими сущностями, а также управляющие их состоянием: успешно загружены ли данные, произошла ли ошибка, или данные находятся в процессе загрузки.

Каждое состояние в слайсах типизируется с помощью интерфейсов, которые описывают структуру данных. Например, для направления в системе используется интерфейс, представленный на рисунке 2.8.

```typescript
/**
 * Интерфейс для представления данных направления.
 * Используется для описания структуры объектов направлений в системе.
 */
export interface Direction {
  id?: number; // Уникальный идентификатор направления.
  event: number; // Идентификатор события, к которому относится направление.
  name: string; // Название направления.
  description: string | null; // Описание направления.
  leader_id: number; // Идентификатор руководителя направления.
}
```

Рисунок 2.8 – Интерфейс для представления данных направления

Этот интерфейс помогает обеспечить строгую типизацию данных, с которыми работают слайсы. Это особенно важно для предотвращения ошибок при работе с данными и для улучшения поддерживаемости кода.

Эндпоинты — это HTTP-запросы, которые приложение отправляет на сервер для получения или изменения данных. В RTK Query настройка каждого эндпоинта происходит в одном месте — в API-слайсе, где для каждого запроса создается соответствующий метод. Например:

– GET для получения данных (например, информации о заявках);

– POST для создания новых данных (например, создание заявки или добавление нового мероприятия);

– PUT для обновления данных (например, изменение заявки или информации о мероприятии);

– PATCH для точечного обновления данных (например, изменение только статуса у заявки вместо редактирования информации о всей заявке);

– DELETE для удаления данных.

Эндпоинты в RTK Query описываются с использованием baseQuery — функции, которая отвечает за отправку запросов, и делятся на два типа: queries (для получения данных с использованием GET) и mutations (для изменения данных с использованием POST, PUT, PATCH, DELETE). Это позволяет чётко разделить операции получения и изменения данных, упрощая структуру API-слайсов и централизуя настройку взаимодействия с сервером (Рис. 2.9).

```
/**
 * API-слайс для работы с данными направлений.
 * Включает в себя эндпоинты для получения, создания, обновления и удаления направлений.
 */
const directionApi = apiSlice.injectEndpoints({
  endpoints: (builder) => ({

    // Эндпоинт для создания нового направления.
    createDirection: builder.mutation<Direction, Omit<Direction, 'id'>>({
      query: (newDirection) => ({
        url: '/api/direction/',   // URL для создания нового направления
        method: 'POST',           // Метод POST
        body: newDirection,       // Тело запроса: новое направление
        headers: {
          'Content-Type': 'application/json', // Указываем, что тело запроса в формате JSON
        },
        withCredentials: true,  // Отправка сессии/кредитов пользователя
      }),
      invalidatesTags: ['Direction'], // После создания нужно обновить список направлений
    }),
```

Рисунок 2.9 – Эндпойнт для создания нового направления

RTK Query автоматически генерирует React-хуки для взаимодействия с сервером. Это значительно упрощает работу с асинхронными запросами, так как хук уже содержит логику для отправки запроса и получения данных, а также для обработки ошибок и состояний загрузки.

Например, для getDirections генерируется хук useGetDirectionsQuery, а для createDirection — хук useCreateDirectionMutation. Эти хуки могут быть использованы непосредственно в компонентах (Рис. 2.10).

```
export default function DirectionsManagement(): JSX.Element {

  const { data: directions = [], isLoading } = useGetDirectionsQuery(); // Получение направлений с сервера.
  const [deleteDirection] = useDeleteDirectionMutation(); // Мутация для удаления направления.
```

Рисунок 2.10 – Пример использования хуков в компоненте

RTK Query предоставляет встроенные механизмы кеширования, что позволяет избежать повторных запросов к серверу, если данные уже были загружены. Кешированные данные хранятся на протяжении времени, пока не наступит инвалидация кеша (например, при изменении данных на сервере или через принудительный рефетч). Это значительно улучшает производительность и уменьшает нагрузку на сервер. Каждый запрос поддерживает автоматическую инвалидацию кеша через конфигурацию providesTags и invalidatesTags.

В проекте используется централизованный подход для обработки состояний загрузки, ошибок и успешных состояний как через middleware, так и через RTK Query. Для улучшения пользовательского опыта и информирования пользователя о текущем процессе работы приложения, состояния отображаются с помощью UI-индикаций:

– состояние загрузки - отображается через индикаторы, такие как спиннеры или экраны загрузки, чтобы пользователи знали, что процесс выполнения операции еще не завершен;

– ошибки - когда возникает ошибка в запросе, middleware перехватывает ошибку, а UI-компоненты показывают пользователю сообщение о сбое;

– пустые состояния - если в ответе от сервера отсутствуют данные, показывается сообщение, например "Нет данных", что улучшает восприятие интерфейса.

RTK Query также предоставляет встроенные флаги состояния — isLoading, isSuccess, isError, которые позволяют эффективно управлять состояниями запросов (см. рисунок 2.10). Например, можно показывать спиннер при загрузке данных или отображать сообщение об ошибке, если запрос не удался. Этот механизм значительно упрощает обработку состояний в UI и помогает сделать взаимодействие с пользователем более отзывчивым и информативным.

Централизация взаимодействия с сервером в одном API-слайсе позволяет легко управлять всеми запросами в проекте. Это упрощает поддержку и тестирование кода, а также уменьшает вероятность ошибок. Вся логика взаимодействия с сервером теперь сосредоточена в одном месте, и все компоненты могут использовать автоматически сгенерированные хуки для отправки запросов, обработки ошибок и управления состоянием загрузки.

2.5.3 Разработка интерфейсов

Разработка функциональных модулей основывалась на техническом задании, представленном в приложении В.

Для разработки интерфейсов был выбран набор компонентов Ant Design, что значительно ускорило процесс разработки. Ант Дизайн предоставляет множество готовых элементов, таких как формы, таблицы, модальные окна, кнопки, меню навигации и другие компоненты, которые сразу можно интегрировать в проект. Это позволило избежать необходимости создавать эти элементы с нуля.

Однако, несмотря на использование готовых решений из Ant Design, для достижения уникального дизайна и соответствия специфичным требованиям проекта потребовалась кастомизация внешнего вида компонентов. Для этого

использовались SCSS-стили, которые предоставляют широкие возможности для гибкой настройки и адаптации интерфейса.

С помощью SCSS стало возможным использовать переменные для задания глобальных параметров (цвета, размеры шрифтов, отступы), что позволяет централизованно управлять стилями и легко менять внешний вид всего приложения (см. рисунок 2.6).

Миксины – это наборы стилей для их дальнейшего переиспользования. Например, для текстовых заголовков был создан миксин h1, представленный на рисунке 2.11.

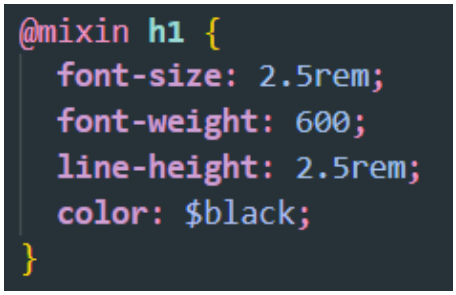

Рисунок 2.11 – Создание миксинов на примере заголовков

Вложенные правила для логичной и иерархичной структуры стилей позволили избежать дублирования CSS-селекторов и улучшить читаемость кода (Рис. 2.12).



Рисунок 2.12 – Пример иерархичной структуры стилей компонентов

Глобальные стили в проекте служат для определения общих правил оформления, таких как цветовая палитра, шрифты, типографика и другие базовые параметры. Эти стили применяются ко всем компонентам приложения, что позволяет обеспечить единообразие в визуальном оформлении.

Локальные стили применяются для настройки специфичных компонентов, таких как карточки, кнопки или формы, которые могут иметь уникальные визуальные особенности. Это позволяет избежать переписывания глобальных стилей и добавляет гибкости в проект.

Одним из ключевых принципов разработки интерфейса является создание переиспользуемых компонентов и UI-паттернов, что позволяет поддерживать код чистым, облегчает его поддержку и масштабирование, а также ускоряет процесс разработки. Этот подход основывается на идее использования одного и того же компонента в различных частях приложения, что позволяет избежать дублирования кода и сделать проект более удобным для будущих изменений.

Преимуществами переиспользуемых компонентов являются:

– упрощение поддержки - если нужно изменить поведение или внешний вид компонента, достаточно обновить его в одном месте, и изменения автоматически применятся ко всем использующим его частям интерфейса;

– масштабируемость - благодаря использованию универсальных компонентов можно быстро расширять интерфейс без необходимости писать новые элементы с нуля;

– чистота и читаемость кода - повторно используемые компоненты способствуют сокращению объемов кода и упрощают его структуру, что делает проект более понятным и удобным для других разработчиков.

Примером переиспользуемых компонентов могут являться поля с одинаковыми стилями и поведением, которые могут быть использованы в различных местах интерфейса. Для этого создаются универсальные компоненты, которые принимают различные параметры для изменения их

поведения, стиля и действий при клике (Рис. 2.13).



Рисунок 2.13 – Состояния поля для поиска

Формы могут быть представлены как компоненты, которые принимают данные и логику в виде пропсов и используют общие стили и обработчики. Например, форма для редактирования направления может быть реализована как компонент и применяться в разных частях приложения (Рис. 2.14).

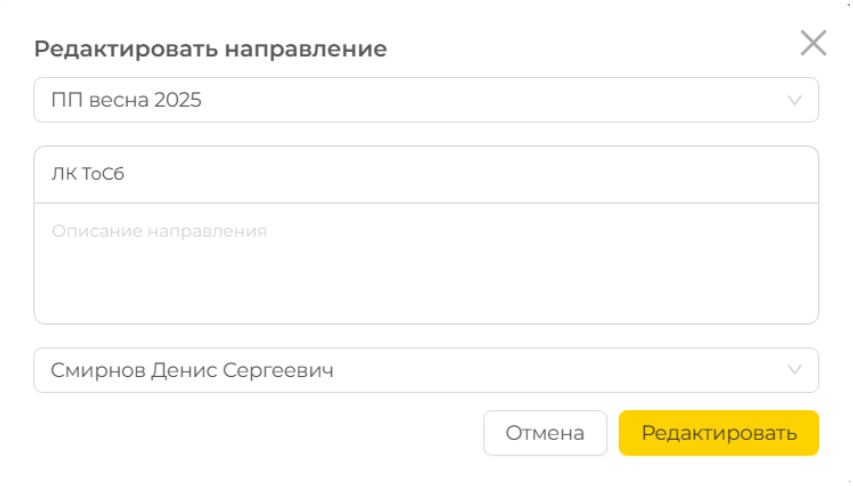

Рисунок 2.14 – Компонент формы для редактирования направлений, открывающейся с помощью модального окна

Использование UI-паттернов позволяет создавать универсальные решения для часто встречающихся задач в интерфейсе, таких как:

– модальные окна - для отображения дополнительной информации или получения ввода от пользователя;

– таблицы и списки - для отображения данных в структурированном виде, с возможностью фильтрации, сортировки и пагинации;

– карточки - для отображения информации в компактной форме.

Поддержка состояний загрузки, ошибок и пустых состояний — важная

часть пользовательского опыта, которая помогает пользователю понять, что происходит в приложении. Индикация таких состояний помогает уменьшить неопределенность. Важно, чтобы пользователь всегда знал, на каком этапе находится приложение, и мог своевременно реагировать на возможные проблемы.

Когда приложение загружает данные или выполняет какие-либо операции, важно предоставить пользователю визуальное подтверждение этого процесса. Это помогает избежать путаницы и сохраняет уверенность пользователя в том, что приложение работает.

В использовании компонента загрузки из Ant Design, например, компонента Spin, можно показать индикатор загрузки во время асинхронных операций. Пример данной логики представлен на рисунке 2.15. В этом примере, пока данные загружаются, отображается спиннер, а после загрузки — сам контент. Это позволяет пользователю видеть, что приложение активно работает и что данные скоро будут загружены.

```
const { data: events, isLoading } = useGetEventsQuery(); // получаем список мероприятий с сервера

if (isLoading) {
  return <Spin />; //если идет загрузка мероприятий показываем компонент Spin
}
```

Рисунок 2.15 – Пример отображения процесса загрузки

Когда происходит ошибка в процессе загрузки данных или выполнения действий, важно уведомить пользователя о проблеме. Это может быть ошибка сервера, неудачная попытка отправки данных или любая другая непредвиденная ситуация.

Можно использовать компонент Notification для отображения сообщений об ошибках. Например, ошибка при создании мероприятия (Рис. 2.16). В этом примере, если возникает ошибка из-за неполноты заполненных данных, она будет отображена в виде красного сообщения с иконкой ошибки. Это помогает пользователю быстро понять, что произошло,

и принять нужные меры (например, повторить попытку).

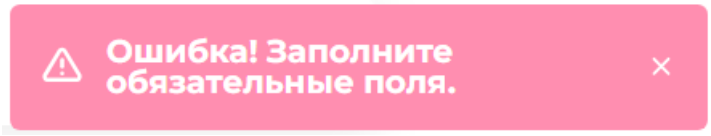

Рисунок 2.16 – Пример отображения уведомления с ошибкой

Когда в приложении нет данных для отображения (например, список пуст, пользователь ещё не добавил информации или результатов нет), важно предоставить пользователю информацию, объясняющую, что именно происходит. Это улучшает восприятие и предотвращает чувство неопределенности. Можно использовать простой текст или кастомную графику для отображения пустых состояний.

На рисунке 2.17 компонент Empty из Ant Design отображает стандартное сообщение о том, что нет данных для отображения. Такой подход позволяет создать информативное пустое состояние, которое улучшает взаимодействие с пользователем.

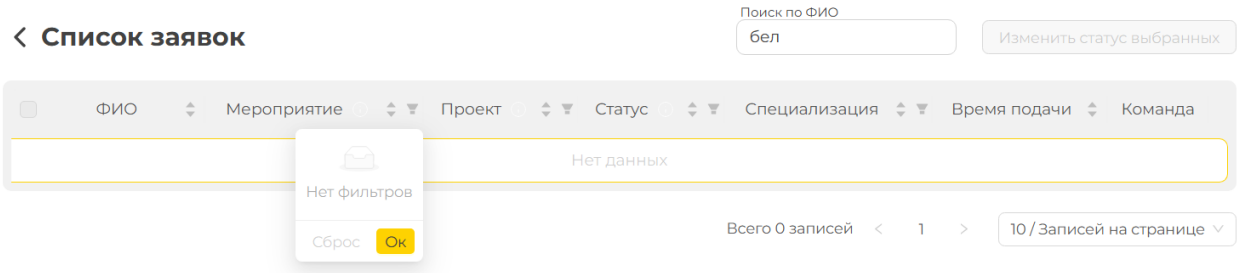

Рисунок 2.17 – Сообщение об отсутствии результатов поиска

Для повышения производительности и обеспечения плавной работы интерфейса в приложении применяются различные техники оптимизации, такие как мемоизация компонентов и пагинация. Эти методы помогают ускорить рендеринг, улучшить отзывчивость интерфейса и уменьшить нагрузку на систему при работе с большими объемами данных.

Мемоизация — это процесс запоминания результатов выполнения

функции или рендеринга компонента с тем, чтобы не выполнять их повторно, если входные данные не изменились. Это помогает уменьшить количество ненужных перерисовок компонентов и повысить производительность интерфейса. Для мемоизации компонентов можно использовать React.memo, который предотвращает перерендеринг компонента, если его пропсы не изменились.

Пагинация помогает разделить большие объемы данных на страницы и загружать их по частям, что снижает нагрузку на систему и ускоряет рендеринг. Вместо того чтобы загружать и отображать все данные сразу, используется только часть данных, что значительно улучшает отзывчивость интерфейса. В Ant Design для этого можно использовать компонент Pagination.

На рисунке 2.18 используется пагинация, чтобы разделить список из 26 элементов на страницы по 10 элементов, а также позволяет переключаться между страницами и выбирать количество отображаемых элементов. Это значительно улучшает производительность, так как загружается только ограниченное количество данных за один раз.

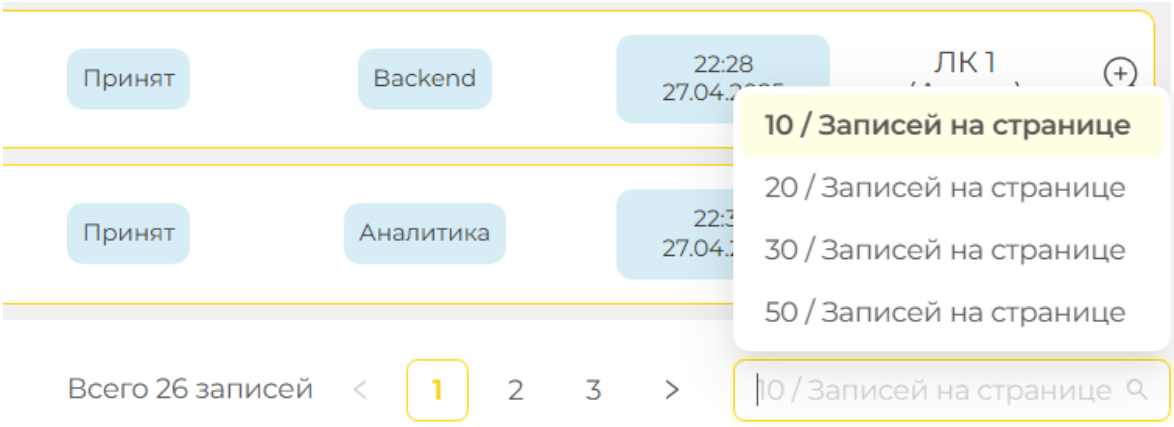

Рисунок 2.18 – Сообщение об отсутствии результатов поиска

Канбан доска — это визуальный инструмент для управления процессами и задачами, который используется для отображения этапов работы и позволяет отслеживать прогресс. Канбан основан на принципе drag and drop и разделения задач на несколько колонок, каждая из которых представляет определённый

статус или этап выполнения работы. Каждая задача (в данном случае заявка) перемещается между колонками, что позволяет наглядно видеть её текущее состояние. Основными элементами канбан-доски являются:

– колонки - каждый этап работы, например: "Новая заявка", "В процессе", "Завершено";

– карточки - объекты, которые перемещаются между колонками, представляя конкретные элементы;

– перетаскивание карточек - возможность перетаскивать карточки элементов из одной колонки в другую в зависимости от их статуса.

В модуле «Управление заявками» канбан-доски реализованы с помощью библиотеки react-beautiful-dnd и используются для двух ключевых целей:

– управление заявками - для наглядного отображения этапов обработки заявок, чтобы пользователи могли легко отслеживать текущий статус каждой заявки;

– настройка роботов и триггеров - для создания гибкой и наглядной настройки, где пользователи могут визуально управлять настройками рабочих процессов, привязывая действия к изменениям статусов.

В процессе разработки значительное внимание было уделено разделению интерфейса на более мелкие компоненты, описанные выше, которые затем были собраны в более крупные, функциональные блоки. Например, элементы интерфейса, такие как кнопки, поля ввода и карточки, были оформлены как мини-компоненты, которые затем использовались для создания более сложных компонентов, таких как списки, формы и меню.

Списки представляют собой компоненты, которые отвечают за отображение коллекций данных, например, для отображения заявок или мероприятий. Эти списки могут включать в себя функциональность сортировки, фильтрации и пагинации, что позволяет эффективно работать с большими объемами данных, не перегружая интерфейс.

В проекте списки реализованы в виде таблиц, которые являются универсальными компонентами, способными отображать данные различных

сущностей. Компонент ListTable выступает в качестве общего компонента для всех таблиц и принимает через пропсы данные (data) и описание столбцов (columns), а также дополнительный пропс на необходимость выделять строки (onSelectRows) (Рис. 2.19). Структура столбцов определяется заранее и позволяет гибко настраивать их внешний вид и функциональность:

– заголовок (header);

– метод рендеринга (render);

– ключ для сортировки (sortKey) – возможность сортировки по столбцу;

– текст подсказки (text) – добавляет иконку, наведение на которую покажет подсказку;

– ширину столбца (width) – при необходимости определяет точную ширину столбца;

– фильтры (filters) – возможность фильтрации по заданным фильтрам;

– автоматическое создание фильтров (autoFilters) – возможность фильтрации по всем возможным фильтрам в столбце.

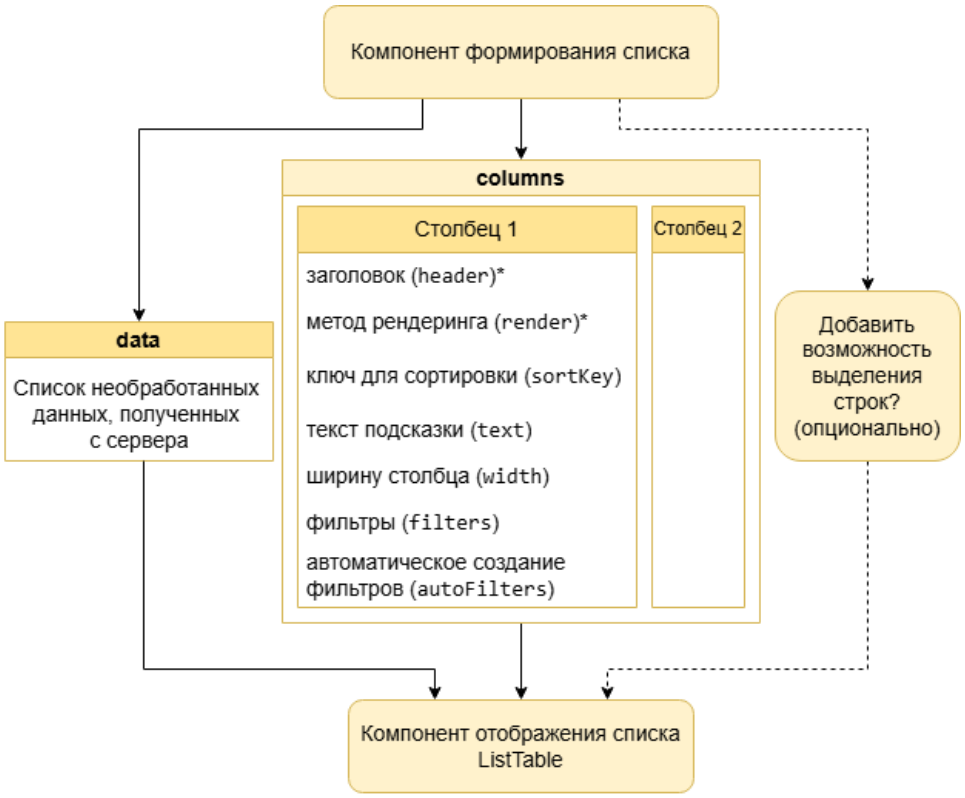

Рисунок 2.19 – Схема формирования таблиц с данными

Такой подход позволяет создавать динамические таблицы с гибкой структурой столбцов, адаптированной под различные требования и сценарии взаимодействия. Например, в списке заявок организатор может выделять заявки, фильтровать их по мероприятиям, проектам, статусам и специализациям, а также сортировать по всем столбцам (Рис. 2.20).

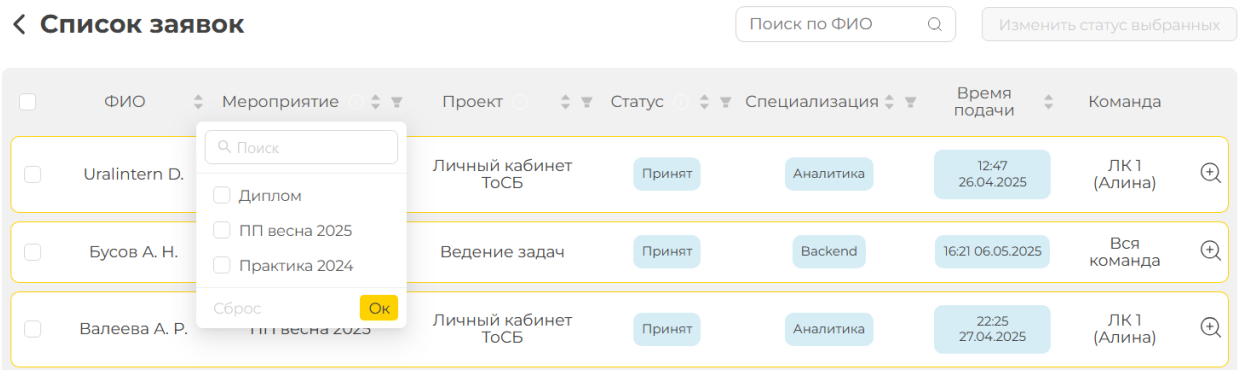

Рисунок 2.20 – Пример формирования списка заявок

Формы в данном модуле играют ключевую роль, особенно при создании и редактировании мероприятий. Для удобства пользователей была реализована многоступенчатая форма, которая разделяет процесс на несколько шагов. Это позволяет избежать перегрузки интерфейса и делает процесс создания мероприятий более понятным и последовательным.

Для реализации многоступенчатого подхода был использован Redux для временного хранения данных. Внутренний Redux-слайс сохраняет данные, введённые на каждом шаге, что позволяет пользователю свободно перемещаться между этапами без потери введённой информации. На каждом шаге формы данные обновляются и сохраняются в хранилище, а при переходе на итоговую страницу пользователь видит все выбранные настройки и может внести изменения.

После завершения заполнения формы данные поочерёдно отправляются на сервер, и при успешной отправке внутреннее хранилище очищается. В случае редактирования мероприятия данные предварительно загружаются с сервера и заполняют соответствующие формы, позволяя пользователю

корректировать существующие настройки.

На рисунке 2.21 представлена структура взаимодействия форм, внутреннего хранилища и сервера при создании и редактировании мероприятий.

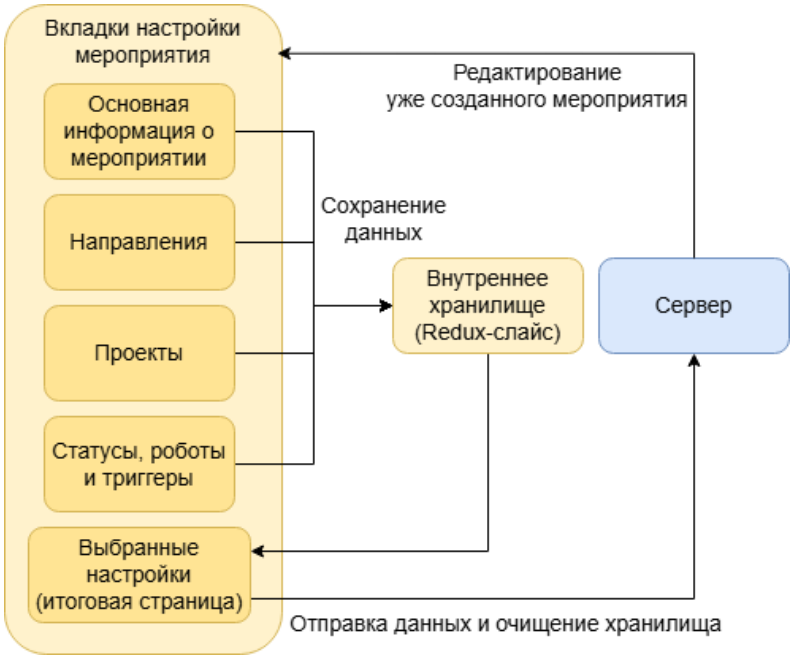

Рисунок 2.21 – Схема движения данных при настройке мероприятия

Важным аспектом интерфейса является использование контекста для управления ролями пользователей. Для централизованного управления ролями пользователей в приложении с помощью React Context был реализован контекст UserRoleContext, который определяет текущую роль пользователя и контролирует доступ к различным элементам интерфейса [23]. Этот контекст используется на уровне всего приложения и позволяет компонентам динамически определять права пользователя и корректировать отображение элементов. Основные элементы реализации:

– провайдер контекста (UserRoleProvider) - оборачивает всё приложение и обеспечивает доступ к данным о текущей роли пользователя;

– маршрутизация по ролям - использование компонента RequireRole, который проверяет наличие у пользователя определённой роли и разрешает доступ к маршрутам на основе этой роли (см. рисунок 2.5);

– хуки для проверки прав:

а. hasRole - проверяет наличие роли у пользователя,

б. hasPermission - проверяет наличие конкретного разрешения,

в. getRoleForObject - определяет доступ пользователя к конкретному объекту, например, к мероприятию.

Использование такого подхода позволяет избежать излишней сложности в интерфейсе, а также гарантирует безопасность данных и доступ к ним только соответствующим пользователям. Все это реализуется с помощью контекста, который управляет состоянием ролей и доступом на уровне всего приложения, что обеспечивает эффективное и безопасное взаимодействие с системой.

Таким образом, разделение интерфейса на мини-компоненты, их последовательная интеграция в более сложные блоки и использование контекста для управления ролями позволяет создать гибкую, адаптивную и эффективную систему, которая подстраивается под потребности каждого пользователя и улучшает общий пользовательский опыт. Все реализованные макеты и интерфейсы представлены в Приложении Д.

2.6 Тестирование и внедрение решения

На этапе тестирования в проекте было использовано ручное тестирование ключевых пользовательских сценариев, что позволило выявить потенциальные проблемы на ранних стадиях разработки. Все сценарии тестировались командой с помощью специально подготовленной таблицы, в которой были описаны основные действия пользователей, их последовательность и ожидаемые результаты. Этот подход обеспечил всестороннюю проверку функционала приложения и позволил убедиться в его корректной работе. Часть таблицы представлена в приложении Е.

Кроссбраузерное тестирование проводилось в четырех основных браузерах: Chrome, Firefox, Яндекс и Edge. В процессе проверяли отображение интерфейса, его адаптивность и функциональность в разных средах, что

позволило убедиться, что приложение работает стабильно в популярных браузерах, которые могут использовать конечные пользователи.

Особое внимание было уделено проверке корректной работы UI-интерактивов, таких как drag-and-drop, модальные окна и формы. Эти элементы являются неотъемлемой частью интерфейса, и их работа важна для удобства пользователей. В ходе тестирования проверили, что все перетаскиваемые элементы, взаимодействие с формами и использование модальных окон происходят без сбоев и ошибок.

Тестирование с различными ролями пользователей проводилось для проверки корректной работы функционала в зависимости от уровня доступа. Мы проверили, как приложение реагирует на изменения ролей (организатор, руководитель направления, проектант, куратор) и убедились, что интерфейс подстраивается под нужды каждого пользователя, предоставляя ему доступ только к тем функциям, которые ему разрешены.

В будущем, для более обширного тестирования, планируется тестирование другими пользователями, в том числе участниками проектной школы, которые захотят записаться на летнюю практику. Это позволит проверить, как приложение работает в условиях реального использования и выявить дополнительные аспекты, которые могли быть упущены в процессе внутреннего тестирования.

Проект был развернут на хостинге https://meetuppoint.ru/, что обеспечило доступность приложения для пользователей. После завершения всех этапов разработки и тестирования решение было внедрено в рабочее окружение, что позволило начать использование системы в реальных условиях. Это обеспечило возможность получения обратной связи от реальных пользователей.

ЗАКЛЮЧЕНИЕ

В ходе выпускной квалификационной работы была проведена разработка клиентской части модуля «Управление заявками» для сервиса организации стажировок в ООО НПЦ «РИЦ». Работа была направлена на автоматизацию процессов обработки заявок, что позволило повысить эффективность управления стажировками и снизить нагрузку на организаторов.

На первом этапе работы была проведена проработка предметной области, что включало исследование текущего процесса обработки заявок, его проблем и ограничений. В результате был разработан набор требований, которые должны были быть реализованы в рамках клиентской части модуля.

Для определения методов и инструментов разработки был выбран стек технологий, включающий React для разработки интерфейсов и Redux для управления состоянием, а также SCSS для стилизации и Ant Design для использования готовых компонентов UI. Этот выбор обусловлен необходимостью создать современный и масштабируемый интерфейс.

Одной из ключевых задач было создание пользовательских интерфейсов, которые обеспечат взаимодействие с системой для разных категорий пользователей. Для этого были разработаны компоненты интерфейса с использованием библиотеки Ant Design, что позволило значительно ускорить процесс разработки и внедрить проверенные UI-элементы. В то же время, для обеспечения уникальности и соответствия проектным требованиям, были созданы кастомные SCSS-стили, которые обеспечили необходимую адаптивность и визуальную согласованность интерфейсов.

В процессе разработки интерфейса было реализовано несколько ключевых компонентов: формы, списки, меню и канбан-доски. Канбан-доски, в частности, оказались важным элементом интерфейса для визуализации и управления заявками. Эти доски использовались для настройки статусов

заявок и автоматических рабочих процессов, а также для организации заявок по различным критериям. Канбан-подход позволил сделать процесс более прозрачным и управляемым.

Взаимодействие с сервером осуществляется через API-запросы, что гарантирует быстрое и надежное обновление данных. Важным аспектом работы стало внедрение механизма управления состоянием с помощью Redux, который позволил эффективно обрабатывать асинхронные операции и поддерживать актуальность данных на интерфейсе.

Для обеспечения безопасности и корректной работы приложения была реализована авторизация и аутентификация пользователей, а также разделение функционала в зависимости от роли пользователя (организатор, проектант, руководитель направления, куратор). Это решение позволило четко разграничить доступ к различным частям системы, что улучшило UX и повысило безопасность.

На завершающем этапе работы была проведена комплексная проверка всех реализованных функциональных элементов интерфейса. Тестирование охватывало как основные пользовательские сценарии (создание, редактирование, просмотр заявок и мероприятий), так и граничные случаи, связанные с авторизацией, правами доступа и некорректными действиями пользователя. Проверка проводилась вручную и с привлечением тестировщиков со стороны заказчика.

После устранения выявленных замечаний клиентская часть модуля была успешно внедрена в рабочее окружение проекта. Все ключевые компоненты - формы, списки, панели управления и канбан-доска - были интегрированы и протестированы в связке с серверной частью. В результате модуль готов к эксплуатации и передан заказчику для последующего использования и обратной связи.

Таким образом, в ходе работы были решены следующие задачи:

– рассмотрены предметную область и определить методы и инструменты разработки клиентской части;

– разработаны пользовательские интерфейсы, учитывая функциональные и нефункциональные требования;

– реализовано взаимодействие клиентской части с серверной логикой системы для автоматизации процессов обработки заявок.

Результатом работы стало создание клиентского модуля «Управление заявками», обеспечивающего взаимодействие между студентами, организаторами и руководителями. Проработанные решения заложили основу для дальнейшей интеграции автоматизированных процессов, что в перспективе позволит значительно повысить эффективность обработки заявок и упростить управление ими.

Таким образом, цели, поставленные в начале работы, были успешно достигнуты, а результаты могут быть использованы для дальнейшего развития системы и интеграции новых функций в рамках проекта «Уральский центр стажировок».

Так же по итогам работы была написана пояснительная записка в соответствии с методическими указаниями и соответствующая ГОСТам о научно-исследовательской работе и о библиографических записях [24]-[26].

12.05.2025 

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Кукси Б. Введение в API-интерфейсы : учебное пособие. – 2024. – URL: https://zapier.com/resources/guides/apis (дата обращения: 09.05.2025).

2. Что такое Frontend и Backend-разработка? : сайт. – URL: https://www.insales.ru/blogs/university/chto-takoe-frontend-i-backend-razrabotka (дата обращения: 09.05.2025).

3. Бондаренко М. П. Цифровая трансформация системы клиентской поддержки как основа для повышения лояльности и качества обслуживания в авторизованных сервисных центрах / М. П. Бондаренко // журнал «Экономика: вчера, сегодня, завтра». – 2024. – Т. 14. № 11А. – С. 532-541. – DOI: 10.34670/AR.2024.21.83.059. – URL: http://www.publishing-vak.ru/file/archive-economy-2024-11/d6-bondarenko.pdf (дата обращения: 09.05.2025).

4. React Documentation : сайт. – URL: https://react.dev/ (дата обращения: 25.03.2025).

5. Ant Design : документация : сайт. – URL: https://ant.design/index-cn (дата обращения: 23.04.2025).

6. Смирнов Д. С. Проектирование информационной системы для организации студенческих стажировок в формате проектной школы : магистерская дис. УрГЭУ. Екатеринбург, 2025. – URL: https://vk.com/doc12283692_685837606?hash=DZCMsQZAj1ipyrBC3wpzkaRN05ctNFTP8Hak (дата обращения: 24.03.2025).

7. User flow: что это такое и почему без него нельзя делать приложения и сайты : сайт. – URL: https://skillbox.ru/media/design/user-flow-basics/ (дата обращения: 26.03.2025).

8. ГОСТ 19.201 - 78. Техническое задание на программу. Требования к содержанию и оформлению. – Введ. 01.07.1979. – Москва: Издательство стан дартов, 1979. – 8 с.

9. ГОСТ 34.602 - 89. Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание

автоматизированной системы. – Введ. 01.01.1990. – Москва: Госстандарт, 1990. – 16 с.

10. TypeScript Documentation : сайт. – URL: https://www.typescriptlang .org/ (дата обращения: 25.04.2025).

11. React Router Documentation : сайт. – URL: https://reactrouter.com/ (дата обращения: 25.03.2025).

12. Redux Toolkit Documentation : сайт. – URL: https://redux-toolkit.js.org/ (дата обращения: 25.03.2025).

13. RTK Query Documentation : сайт. – URL: https://redux-toolkit.js.org/ rtk-query/overview (дата обращения: 25.03.2025).

14. React Context Documentation : сайт. – URL: https://ru.react.js.org/docs/ context.html (дата обращения: 23.04.2025).

15. SCSS (Sass) Documentation : сайт. – URL: https://sass-lang.com/ (дата обращения: 25.03.2025).

16. Date-fns vs Dayjs: the Battle of JavaScript Date Libraries : сайт. – URL: https://www.dhiwise.com/post/date-fns-vs-dayjs-the-battle-of-javascript-date-libraries (дата обращения: 23.04.2025).

17. React Beautiful DnD Documentation : сайт. – URL: https://github.com/ atlassian/react-beautiful-dnd (дата обращения: 23.04.2025).

18. Vite Documentation : сайт. – URL: https://vite.dev/ (дата обращения: 25.03.2025).

19. NPM Documentation : сайт. – URL: https://www.npmjs.com/ (дата обращения: 23.04.2025).

20. Что такое JSON? : сайт. – URL: https://habr.com/ru/articles/554274/ (дата обращения: 23.04.2025).

21. Структурирование React-приложений : сайт. – URL: https://habr.com/ ru/companies/ruvds/articles/460793/ (дата обращения: 23.04.2025).

22. JSDoc Documentation : сайт. – URL: https://jsdoc.app/about-getting-started (дата обращения: 27.04.2025).

23. Уотмор Дж. React – Руководство по авторизации на основе ролей с

примером : сайт. – URL: https://jasonwatmore.com/post/2019/02/01/react-role-based-authorization-tutorial-with-example (дата обращения: 23.04.2025).

24. Т. И. Алферьева Методические рекомендации для подготовки и оформления выпускных квалификационных работ : нормативный документ : / Т. И. Алферьева, В. Н. Васина, Д. Б. Шадрин - Екатеринбург : УрФУ, 2019. – 94 с. - URL: https://study.urfu.ru/Aid/ViewMeta/13865 (дата обращения: 09.05.2025).

25. ГОСТ 7.32-2017. Система стандартов по информации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления : межгосударственный стандарт : утвержден и введен в действие приказом Федерального агентства по техническому регулированию и метрологии от 24 октября 2017 г. № 1494-ст /дата введения 2018-07-01. - (Система стандартов по информации, библиотечному и издательскому делу) // Кодекс : электронный фонд правовых и нормативно технических документов. URL: https://docs.cntd.ru/document /1200157208 (дата обращения: 09.05.2025).

26. ГОСТ Р 7.0.100-2018. Библиографическая запись. Библиографическое описание. Общие требования и правила составления : национальный стандарт Российской Федерации : утвержден и введен в действие приказом Федерального агентства по техническому регулированию и метрологии от 3 декабря 2018 г. № 1052-ст : введен впервые / дата введения 2019-07-01. (Система стандартов по информации, библиотечному и издательскому делу) // Кодекс : электронный фонд правовых и нормативно-технических документов. URL: https://docs.cntd.ru/document/1200161674 (дата обращения: 09.05.2025).

ПРИЛОЖЕНИЕ А
(справочное)

Схема декомпозиции бизнес-процессов

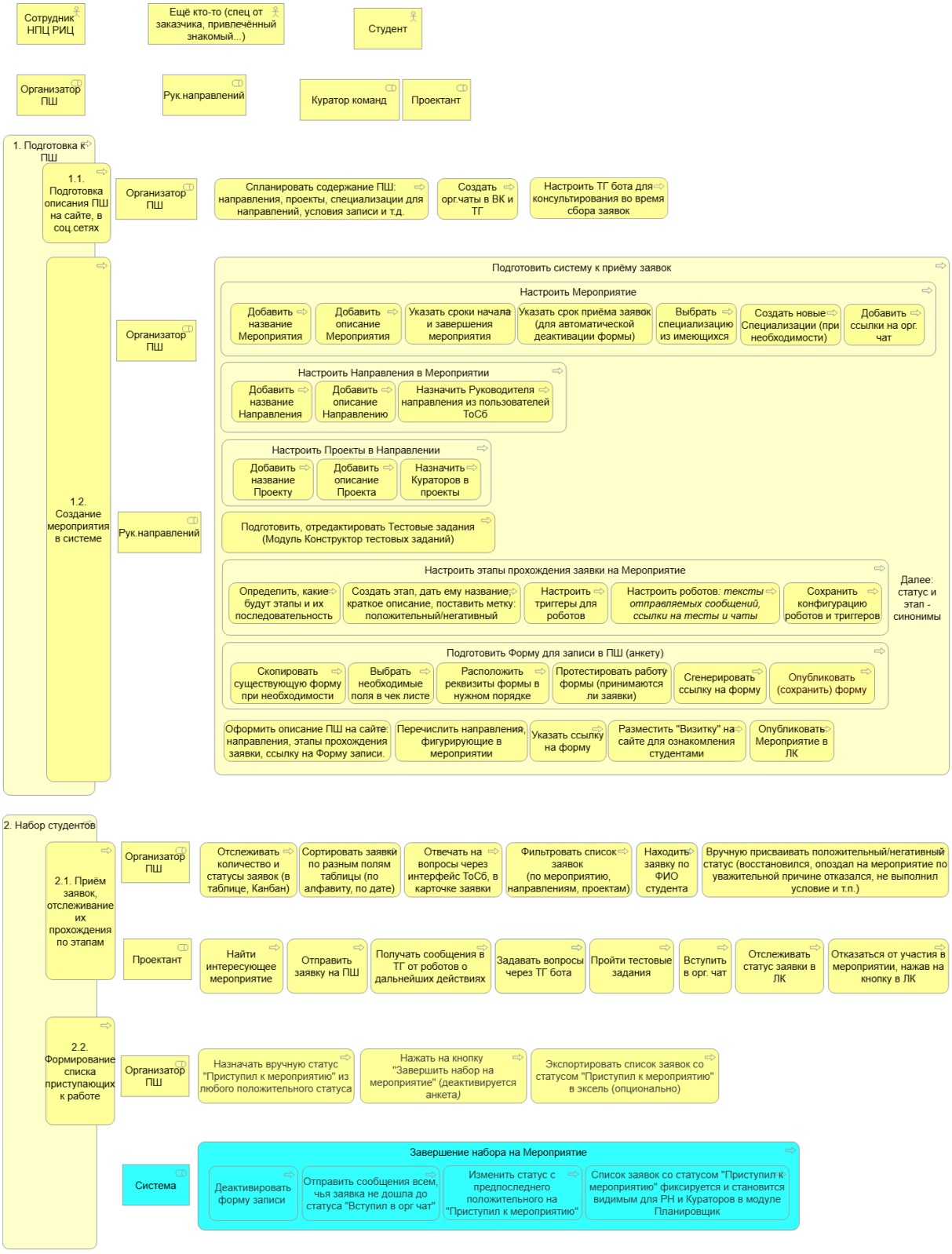

Рисунок А.1 – Схема декомпозиции бизнес-процессов приема заявок в рамках системы «Управление заявками»

ПРИЛОЖЕНИЕ Б
(справочное)

Схема связи данных и интерфейсов

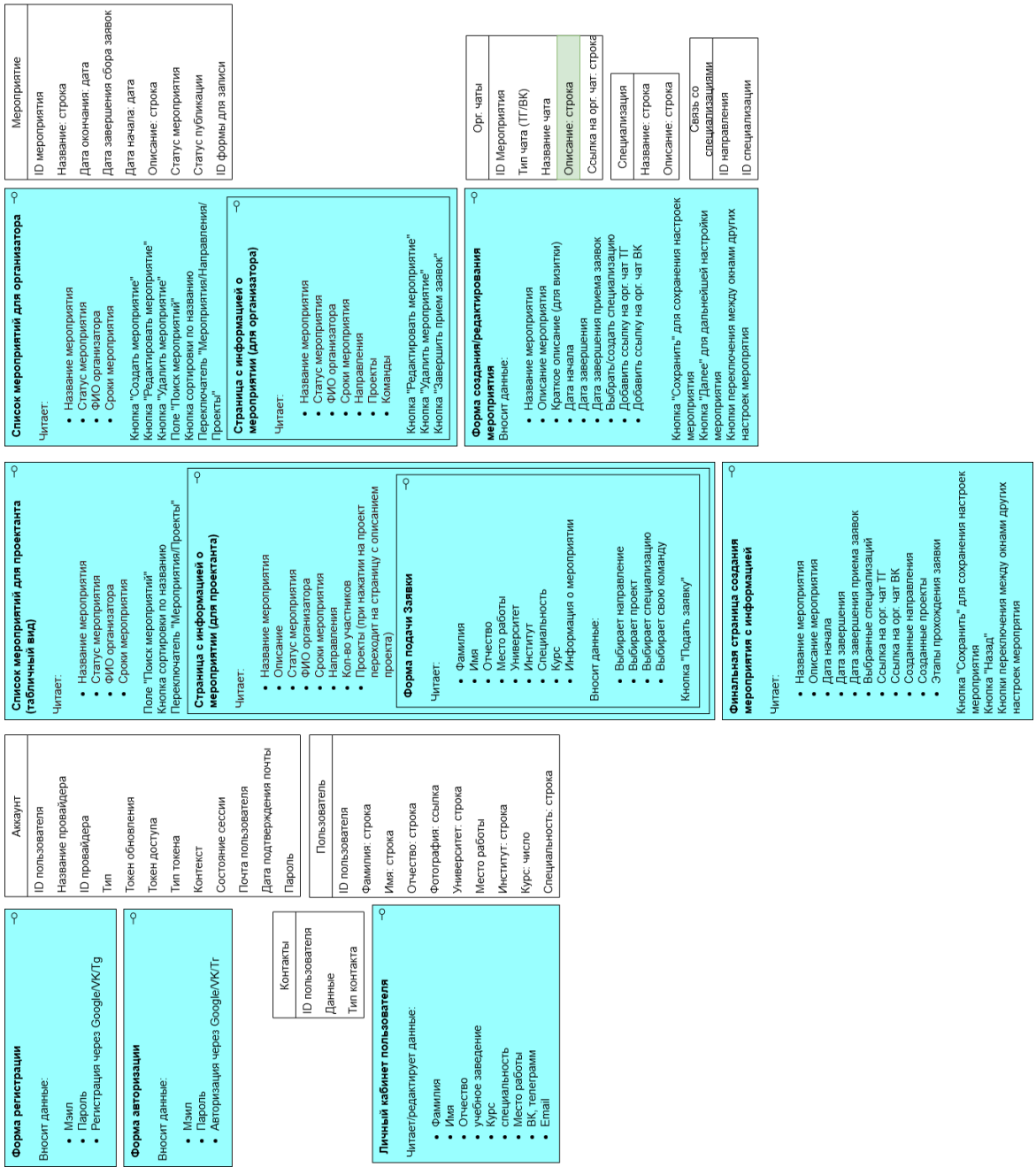

Рисунок Б.1 – Первая часть схемы связи данных и интерфейсов



Рисунок Б.2 – Вторая часть схемы связи данных и интерфейсов



Рисунок Б.3 – третья часть схемы связи данных и интерфейсов

ПРИЛОЖЕНИЕ В
(обязательное)

Частное техническое задание

Техническое задание

Согласовано

От имени «Заказчика»

_____ Смирнов Д. С.

Подпись Расшифровка подписи

Согласовано

От имени «Исполнителя»

_____ Трефелова А. Н.

Подпись Расшифровка подписи

Техническое задание на разработку клиентской части модуля «Управление заявками» веб-сервиса «Точка сбора» ООО НПЦ «РИЦ»

Екатеринбург 2025

Рисунок В.1 – Титульная страница частного технического задания

Продолжение ПРИЛОЖЕНИЯ В

Содержание

1 Общие сведения ... 3
1.1 Наименование системы ... 3
1.2 Наименование Исполнителя и Заказчика ... 3
1.3 Плановые сроки выполнения работ по созданию системы ... 3
2 Цели и назначение создания системы ... 4
2.1 Назначение приложения ... 4
2.2 Цель ... 4
2.3 Характеристика объектов автоматизации ... 4
3 Требования к веб-приложению ... 5
3.1 Функциональные требования ... 5
3.2 Нефункциональные требования: ... 7
4 Состав и содержание работ по созданию системы ... 8
5 Порядок контроля и приемки системы ... 9

2

Рисунок В.2 – Содержание частного технического задания

1 Общие сведения

1.1 Наименование системы

Клиентская часть модуля «Управление заявками» в рамках системы автоматизации обработки заявок на практики и стажировки для ООО НПЦ «РИЦ».

1.2 Наименование Исполнителя и Заказчика

Заказчик: ООО НПЦ «РИЦ»
Исполнитель: Трефелова Алина Николаевна

1.3 Плановые сроки выполнения работ по созданию системы

Плановая дата начала: 06.12.2024
Плановая дата окончания: 12.05.2025

Рисунок В.3 – Третья страница частного технического задания

2 Цели и назначение создания системы

2.1 Назначение приложения

Веб-приложение предназначено для автоматизации процессов обработки заявок на практики и стажировки студентов, обеспечивая взаимодействие между организаторами, студентами и руководителями в рамках проектных школ.

2.2 Цель

Реализовать клиентскую часть модуля «Управление заявками», обеспечивающего полный цикл взаимодействия пользователей с системой: настройку автоматизации подачи заявок, подачу заявок, их фильтрацию, отслеживание и изменение статусов.

2.3 Характеристика объектов автоматизации

Объектом автоматизации являются следующие процессы сервиса:

- процесс создания мероприятий в системе;
- процесс подачи заявок проектантами;
- процесс приёма заявок и прохождения их по этапам.

4

Рисунок В.4 – Четвертая страница частного технического задания

3 Требования к веб-приложению

3.1 Функциональные требования

1. Регистрация и авторизация пользователей:

– система должна поддерживать регистрацию и авторизацию через почту и Google аккаунт;

– пользователи должны иметь возможность войти в систему через указанные методы, используя стандартные механизмы аутентификации и авторизации;

2. Профиль пользователя:

– пользователи должны иметь возможность просматривать и редактировать свой профиль;

– необходимо обеспечить возможность просмотра профилей других пользователей;

3. Управление мероприятиями:

– организаторы должны иметь возможность создавать, редактировать и удалять мероприятия;

– мероприятия должны отображаться в виде таблицы, а также быть доступны для просмотра детально;

– система должна поддерживать возможность выбора отображаемых столбцов таблицы и их ширины, чтобы организатор мог адаптировать отображение под свои нужды;

4. Управление специализациями:

– система должна поддерживать создание, редактирование и удаление специализаций для мероприятий;

5. Управление направлениями:

– организатор должен иметь возможность создавать, редактировать и удалять направления мероприятий;

Рисунок В.5 – Пятая страница частного технического задания

– направления должны отображаться в виде таблицы с возможностью выбора отображаемых столбцов и их ширины;

6. Управление проектами:

– создание, редактирование и удаление проектов должны быть доступны пользователю с правами администратора;

– просмотр проектов должен быть возможен в виде таблицы с детальным отображением информации о каждом проекте;

7. Настройка формы подачи заявки:

– организатор может настраивать форму заявки, определяя порядок полей, их обязательность и типы данных;

8. Настройка статусов, роботов и триггеров:

– организатор должен иметь возможность создавать, редактировать и удалять статусы заявок;

– статусы должны отображаться в виде Канбан-доски для удобства отслеживания состояния заявок;

– система должна поддерживать добавление, редактирование и удаление роботов и триггеров для автоматической обработки заявок и их статусов;

9. Управление заявками:

– система должна предоставлять функционал для просмотра заявок на мероприятия в виде таблицы и Канбан-доски;

– организатор должен иметь возможность изменять статус заявки при необходимости;

– также необходимо предусмотреть возможность завершить прием заявок на мероприятие, что деактивирует форму заявки;

10. Подача заявки:

– система должна автоматически заполнять данные о студенте при подаче заявки (например, имя, контактные данные), чтобы сократить время подачи заявки;

Рисунок В.6 – Шестая страница частного технического задания

– студент должен иметь возможность вручную дополнить данные, если это необходимо;

– подача заявки должна происходить в заранее настроенной форме, что обеспечит стандартный и удобный процесс для всех пользователей;

11. Отслеживание статуса поданной заявки:

– студенты должны иметь возможность отслеживать статус своей поданной заявки в реальном времени через интерфейс системы.

3.2 Нефункциональные требования:

1. Интерфейс: адаптивный дизайн с учетом ролей пользователя;
2. Производительность: время загрузки страницы — не более 2 секунд;
3. Безопасность: аутентификация и авторизация с использованием JWT;
4. Совместимость: поддержка современных версий браузеров (Chrome, Firefox, Safari, Edge);
5. Масштабируемость: возможность добавления новых модулей без необходимости изменения существующего функционала;
6. Устойчивость к сбоям: восстановление данных при перезагрузке страницы.

Рисунок В.7 – Седьмая страница частного технического задания

4 Состав и содержание работ по созданию системы

1. Сбор и анализ требований к функционалу клиентской части модуля;
2. создание макетов и прототипов пользовательских экранов;
3. реализация интерфейсов с использованием React и Redux;
4. настройка взаимодействия с серверной частью через RTK Query;
5. проведение тестирования.

8

Рисунок В.8 – Восьмая страница частного технического задания

5 Порядок контроля и приемки системы

1. Проверка реализации каждого функционального модуля на соответствие требованиям ТЗ;

2. проведение тестирования на соответствие функциональным и нефункциональным требованиям;

3. оценка полноты реализации всех заявленных функций, проверка на устойчивость к ошибкам и корректность выполнения основных сценариев;

4. передача исходного кода и документации заказчику.

Рисунок В.9 – Девятая страница частного технического задания

ПРИЛОЖЕНИЕ Г
(справочное)

Элементы навигации по сайту

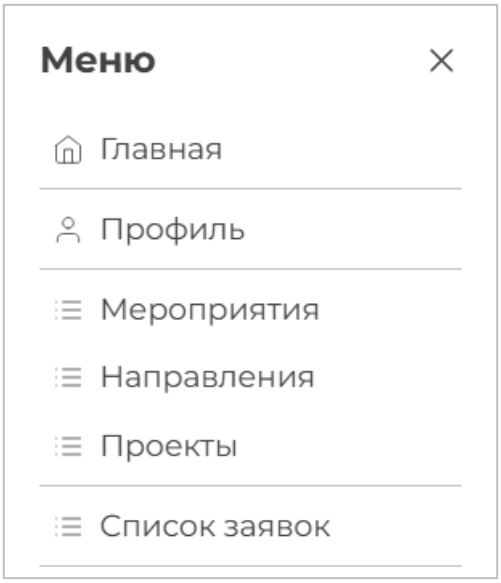

Рисунок Г.1 – Боковая панель навигации по сайту



Рисунок Г.2 – Верхняя панель сайта для авторизованных пользователей



Рисунок Г.3 – Верхняя панель сайта для неавторизованных пользователей

ПРИЛОЖЕНИЕ Д
(обязательное)

Реализованные интерфейсы

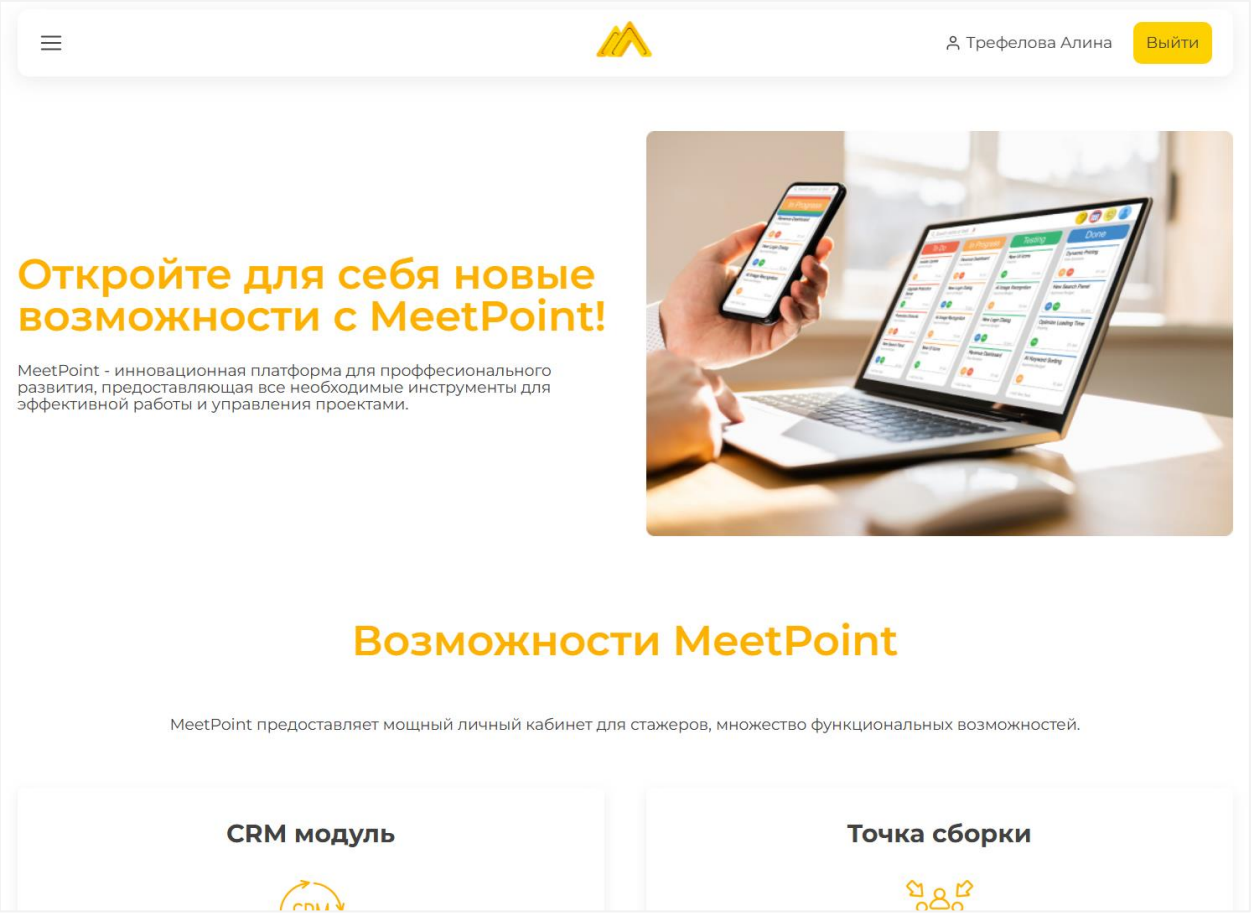

Рисунок Д.1 – Главная страница веб-приложения

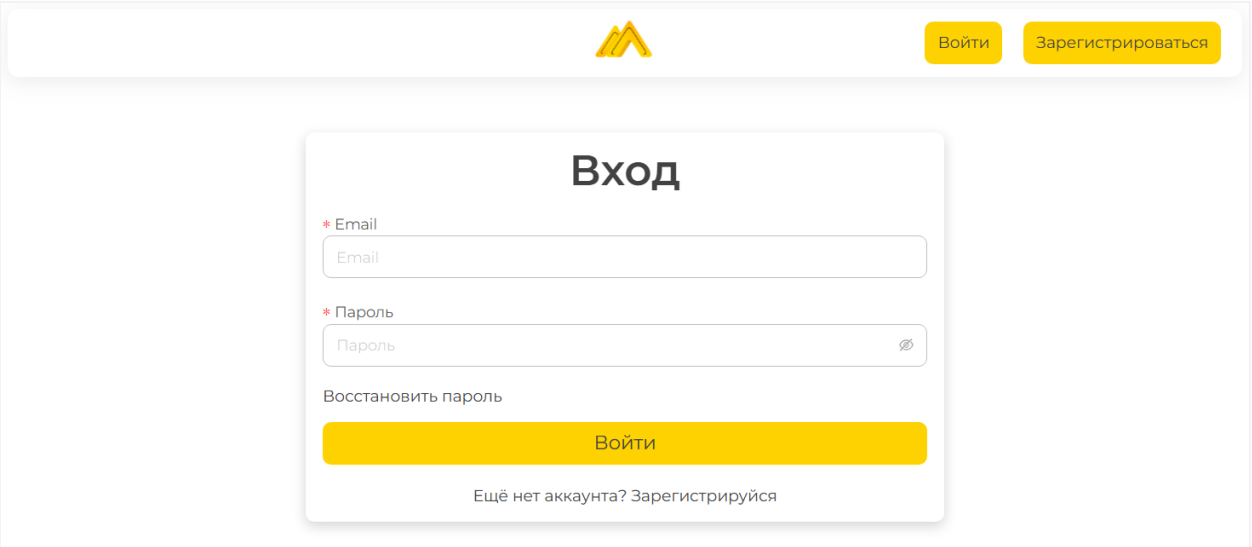

Рисунок Д.2 – Форма входа

Продолжение ПРИЛОЖЕНИЯ Д

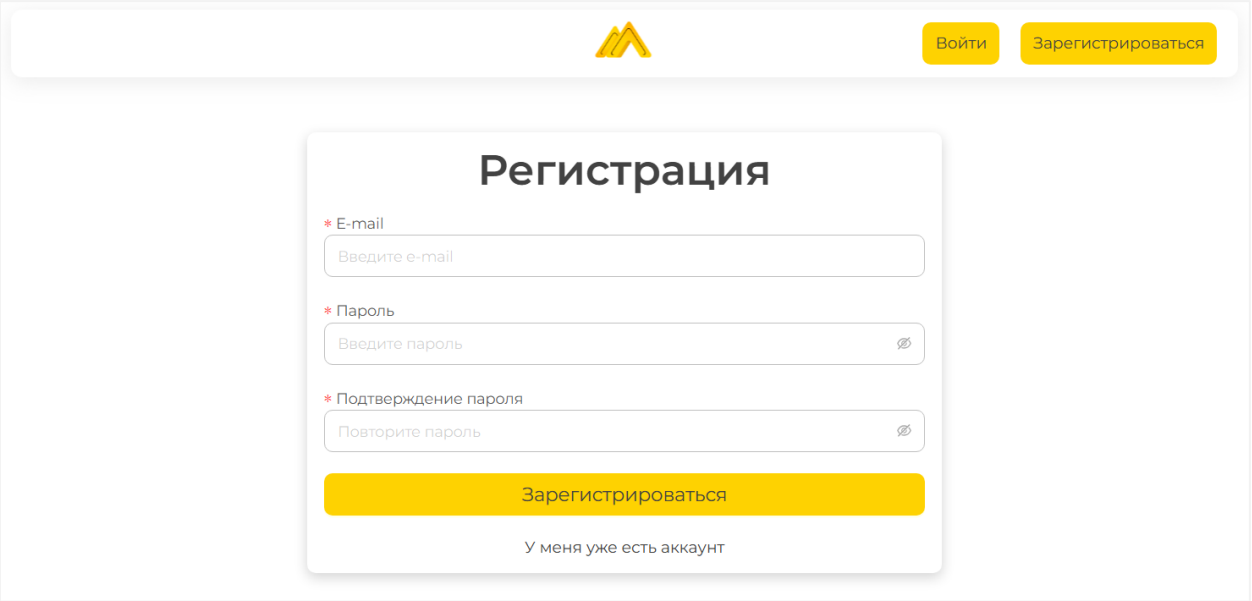

Рисунок Д.3 – Форма регистрации

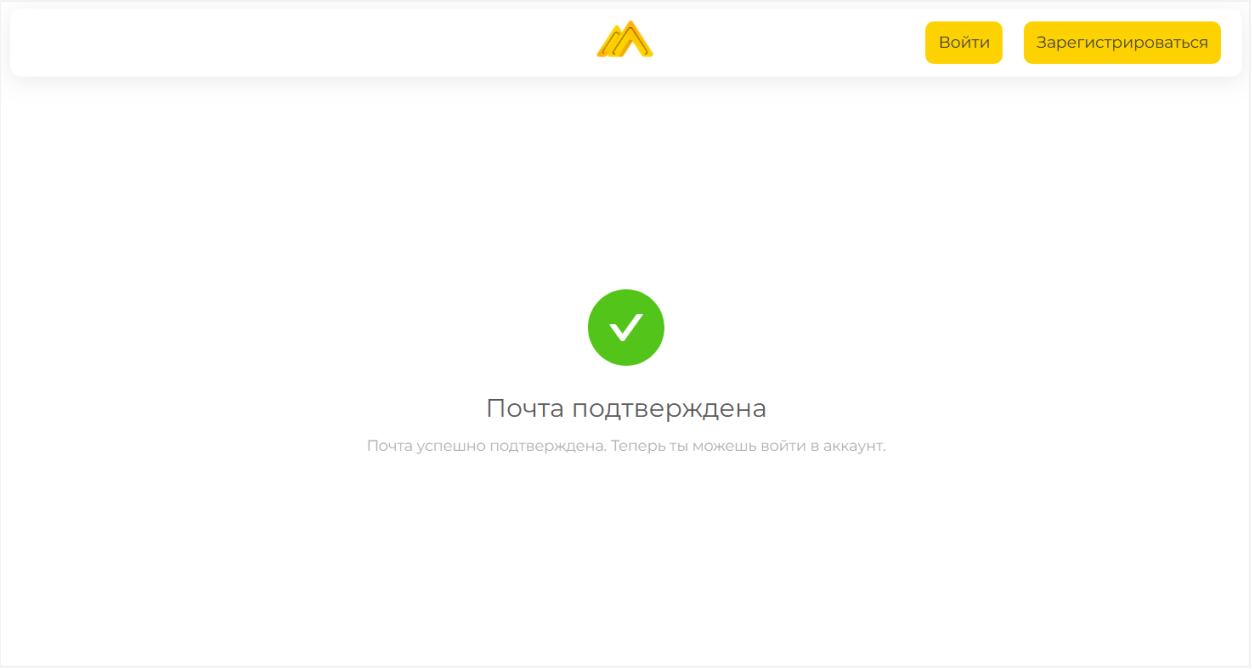

Рисунок Д.4 – Подтверждение почты

Продолжение ПРИЛОЖЕНИЯ Д

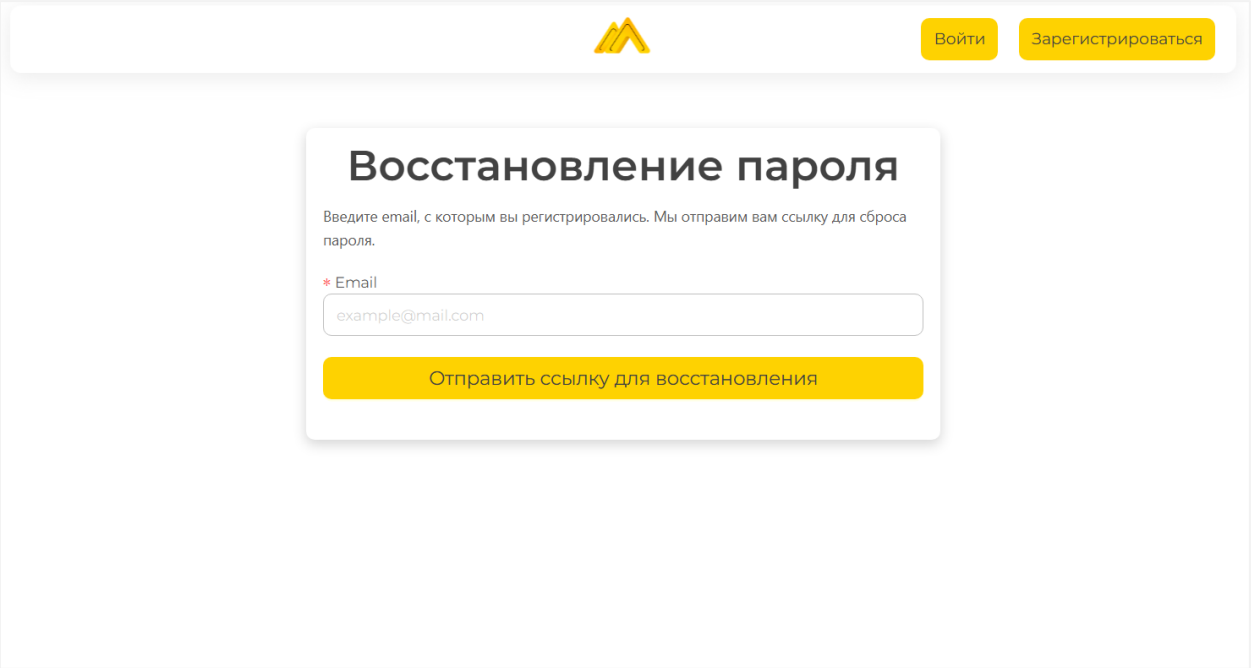

Рисунок Д.5 – Форма восстановления пароля

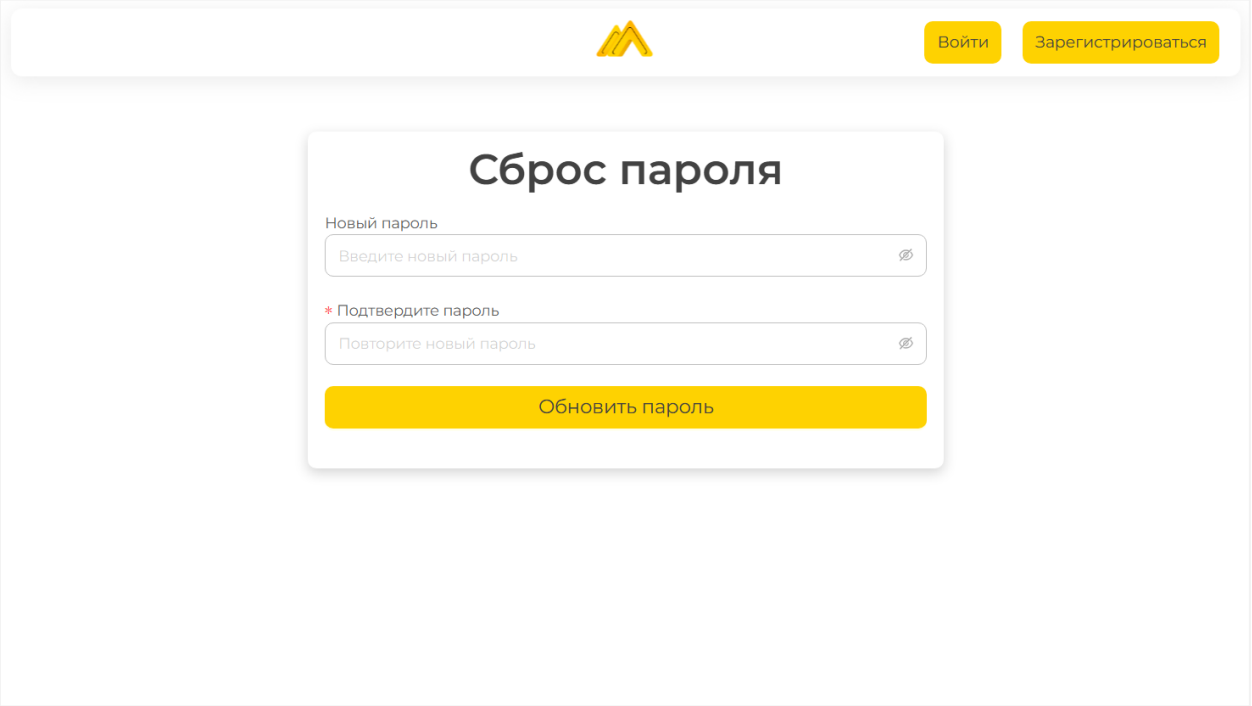

Рисунок Д.6 – Форма подтверждения сброса пароля

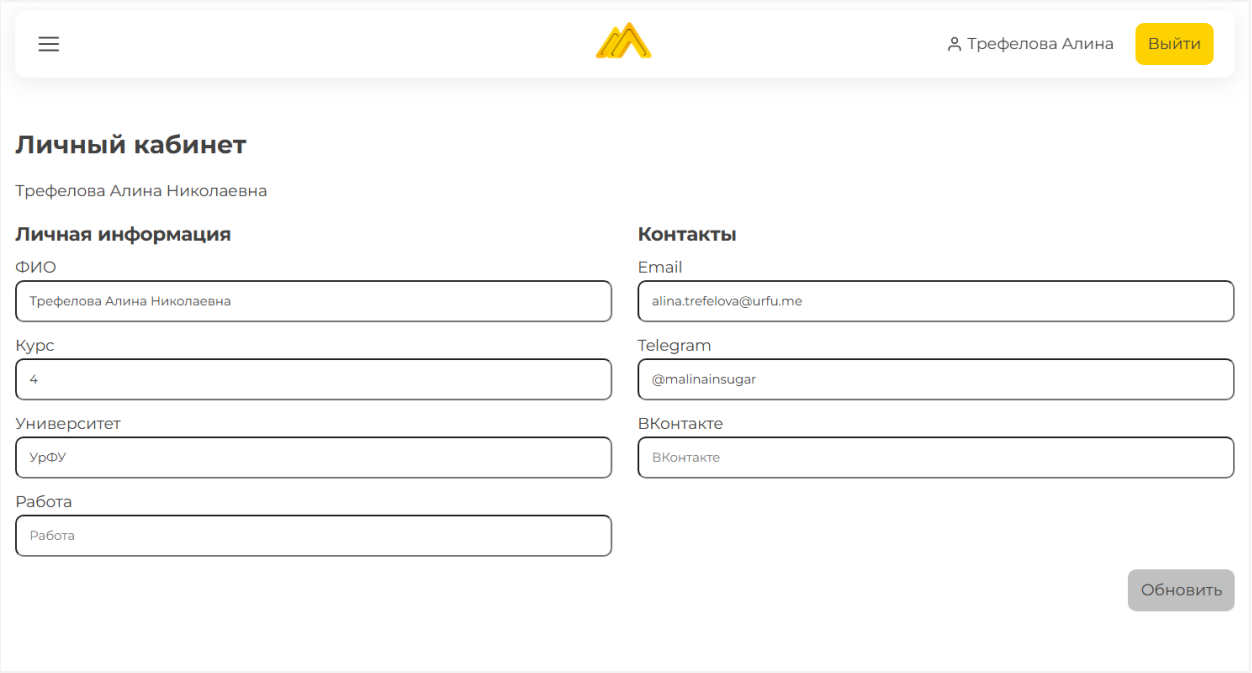

Рисунок Д.7 – Личный профиль пользователя

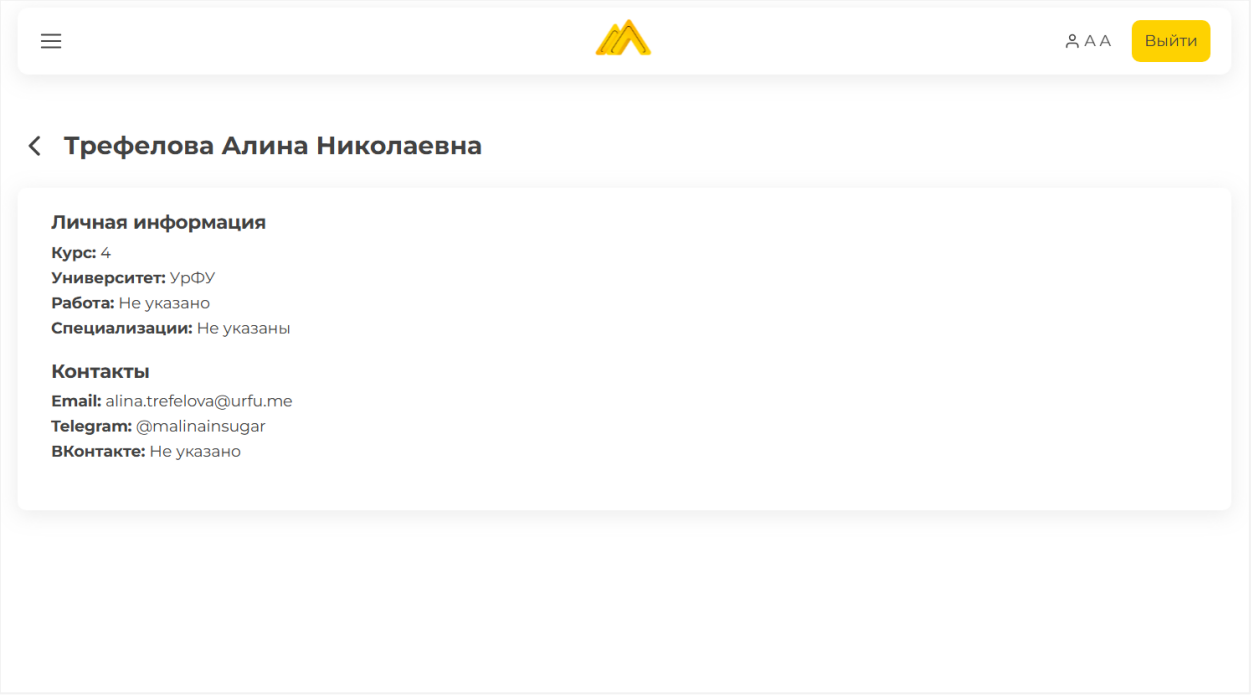

Рисунок Д.8 – Профиль другого пользователя

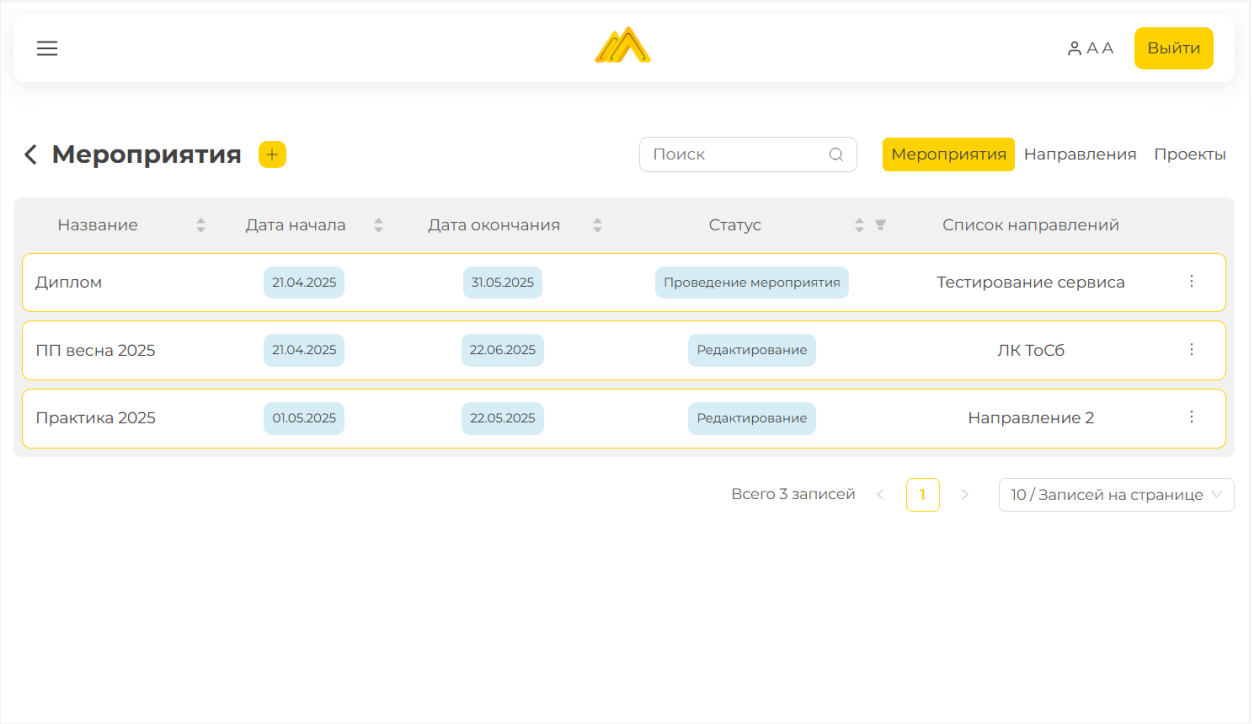

Рисунок Д.9 – Список мероприятий

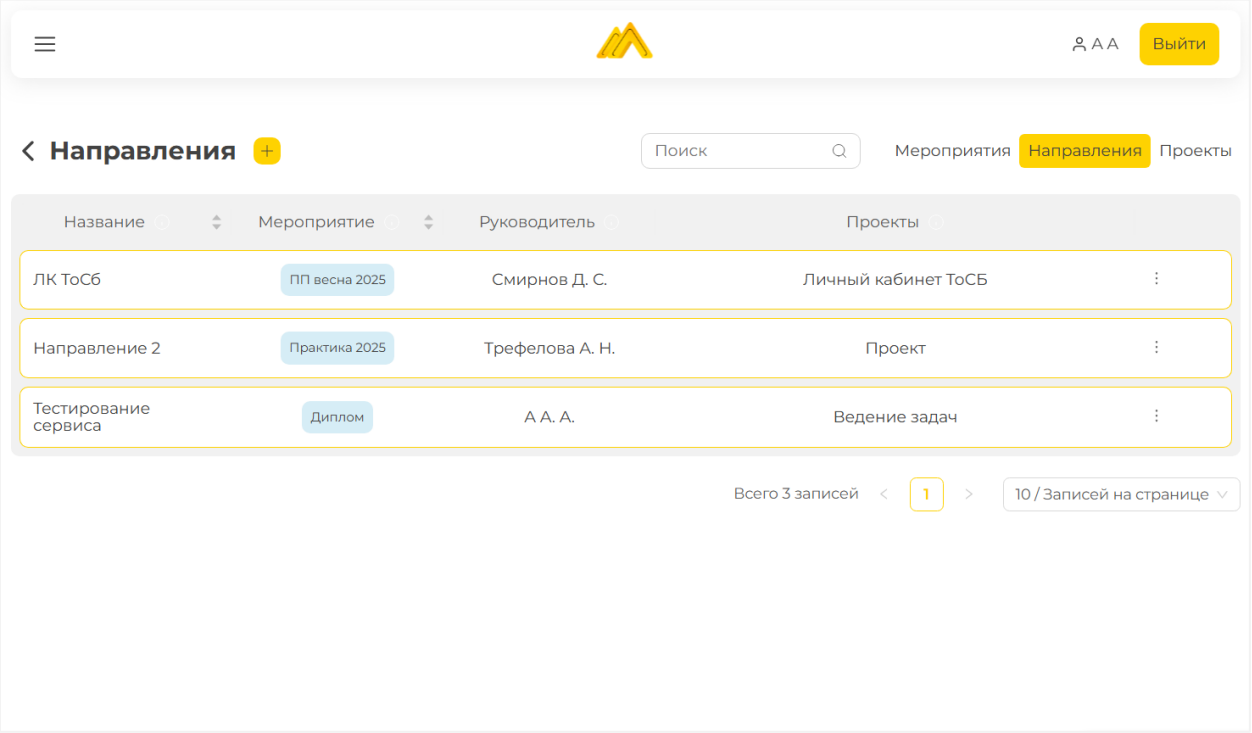

Рисунок Д.10 – Список направлений

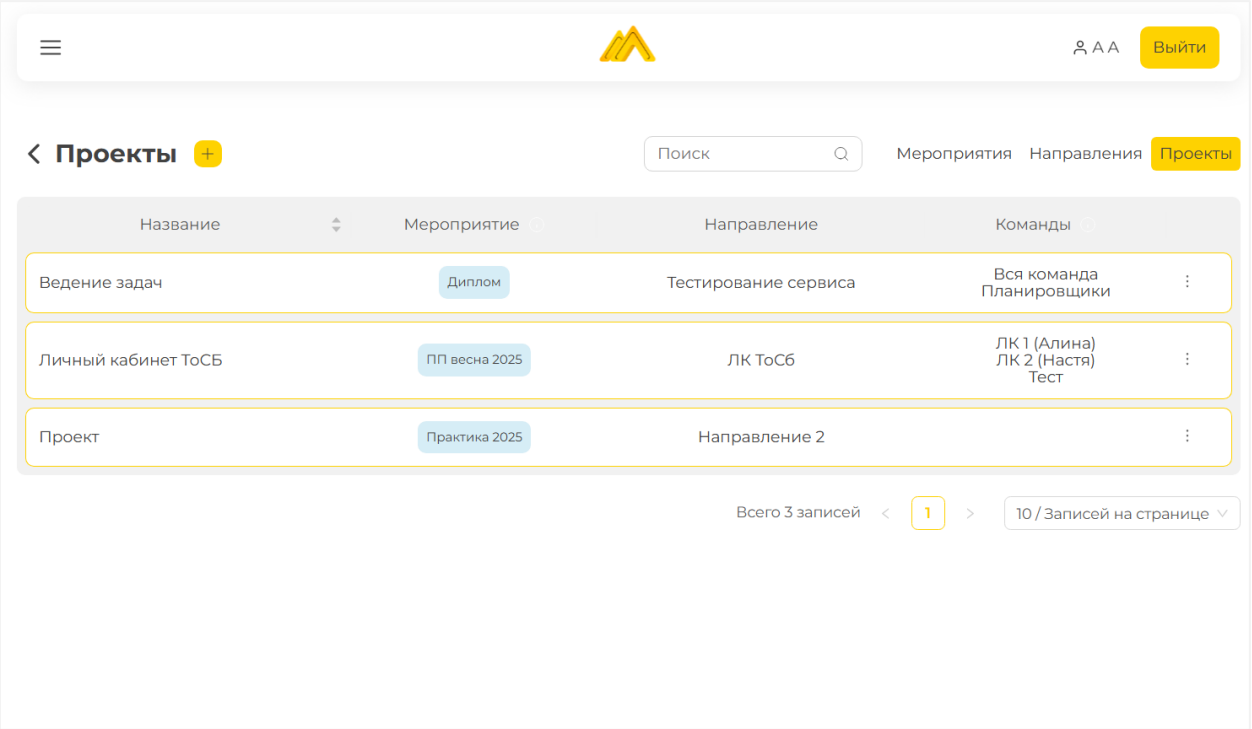

Рисунок Д.11 – Список проектов

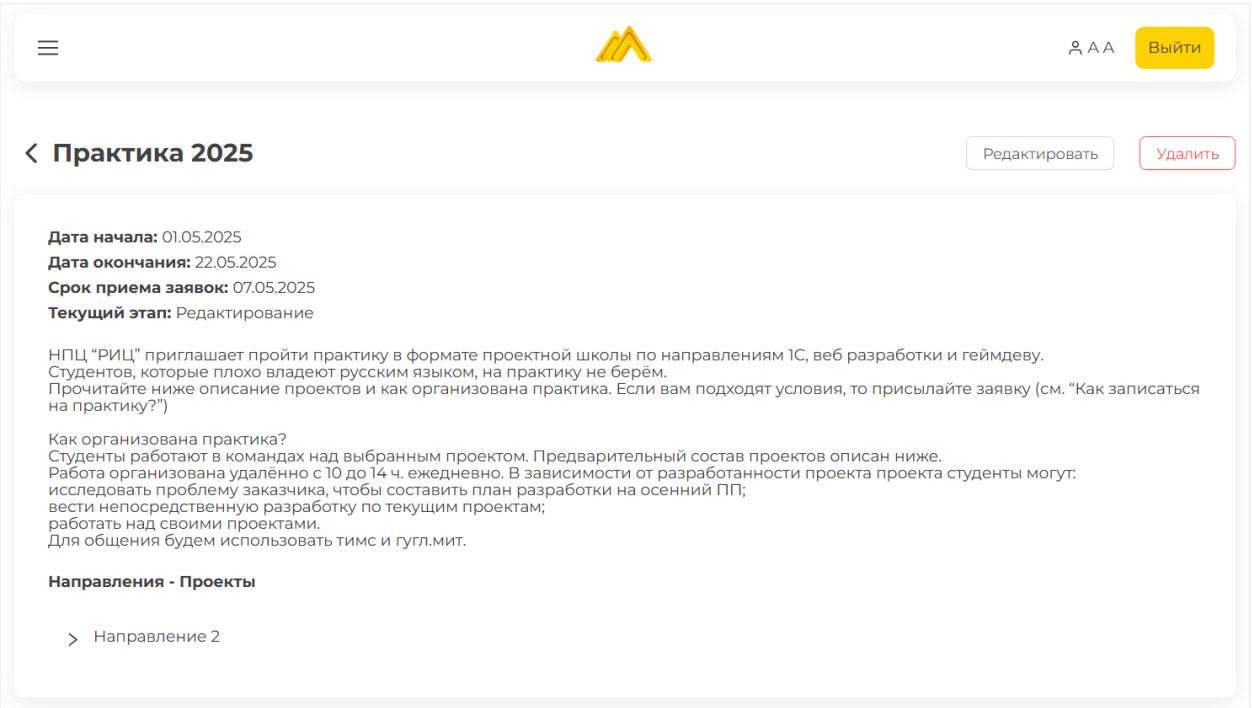

Рисунок Д.12 – Страница с информацией о мероприятии для организатора

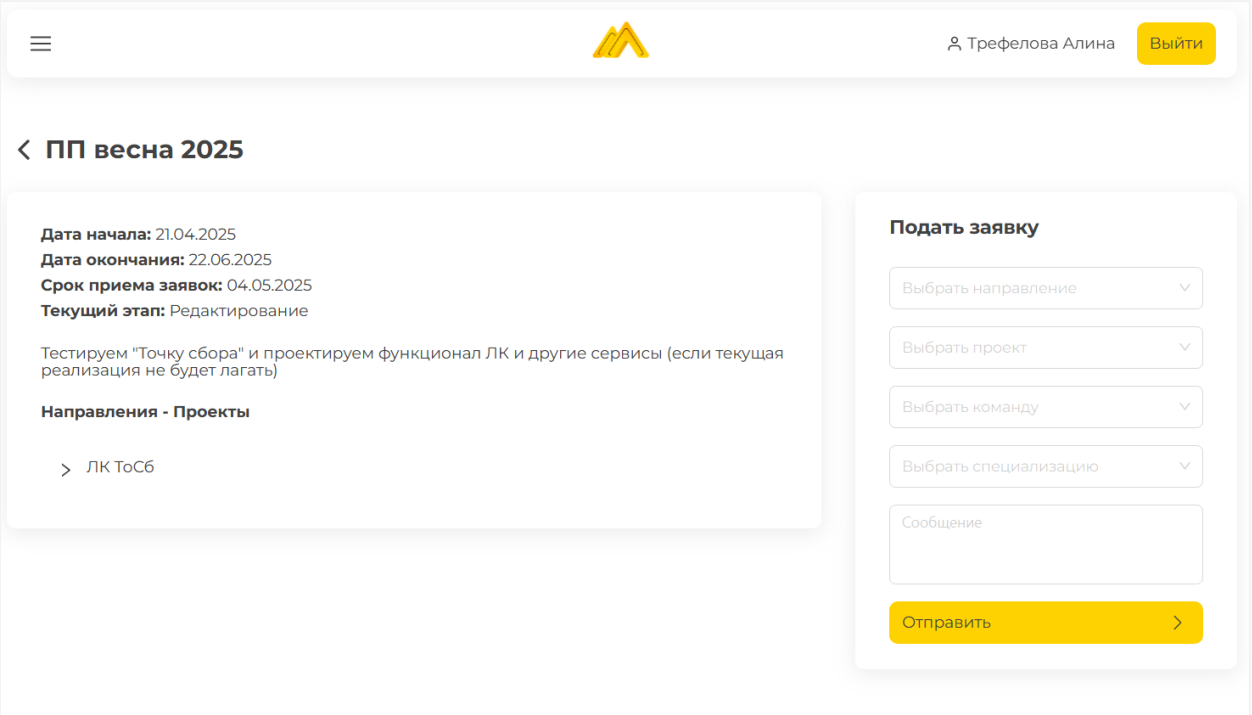

Рисунок Д.13 – Форма для подачи заявки для проектанта

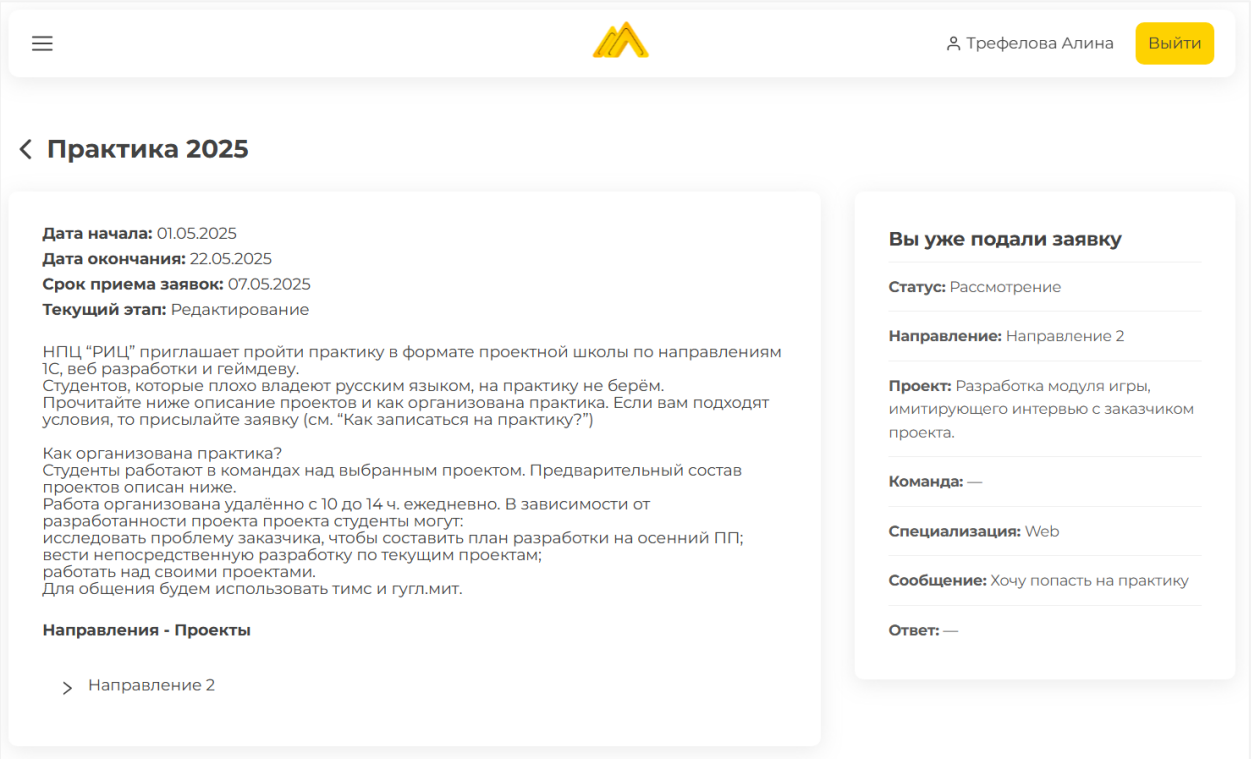

Рисунок Д.14 – Информация о поданой заявке для проектанта

Продолжение ПРИЛОЖЕНИЯ Д

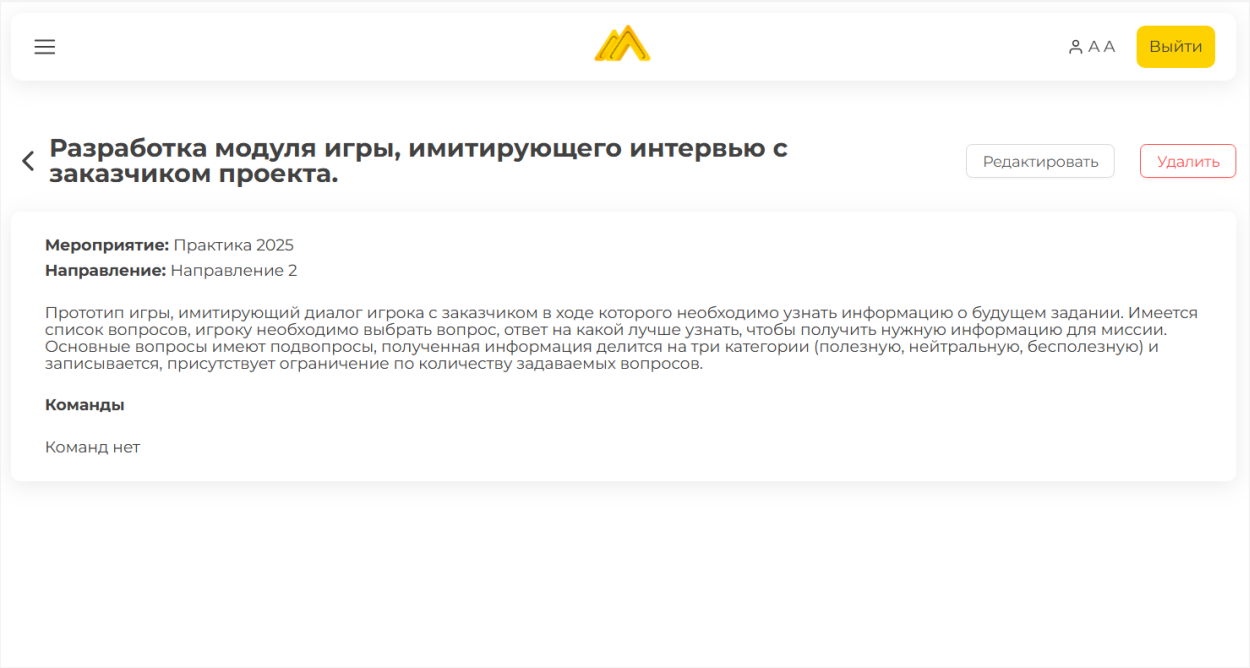

Рисунок Д.15 – Страница с информацией о проекте

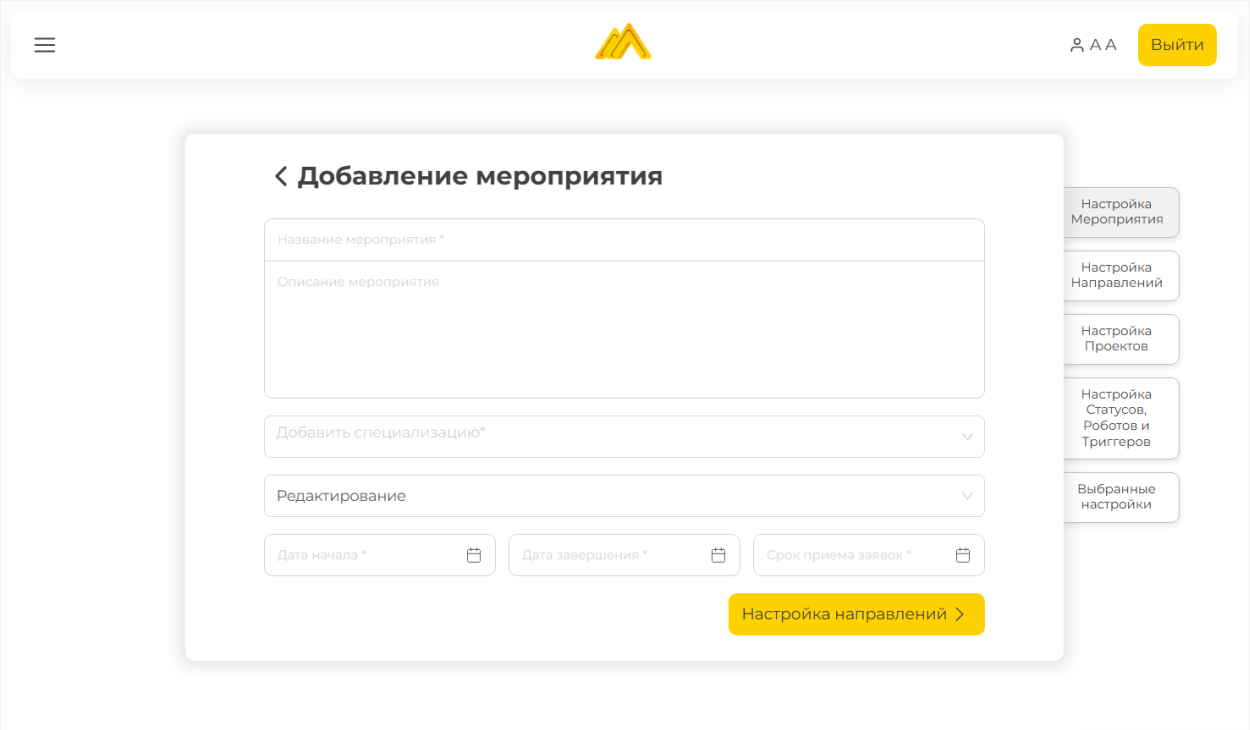

Рисунок Д.16 – Форма настройки мероприятия

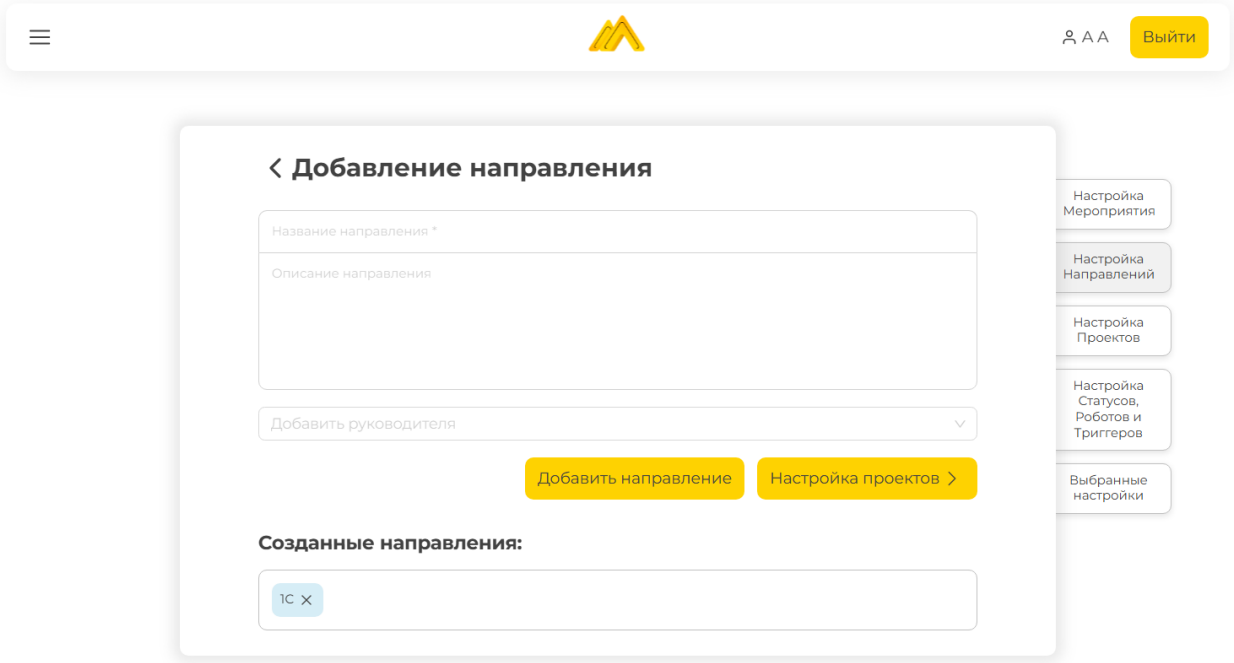

Рисунок Д.17 – Форма добавления направлений

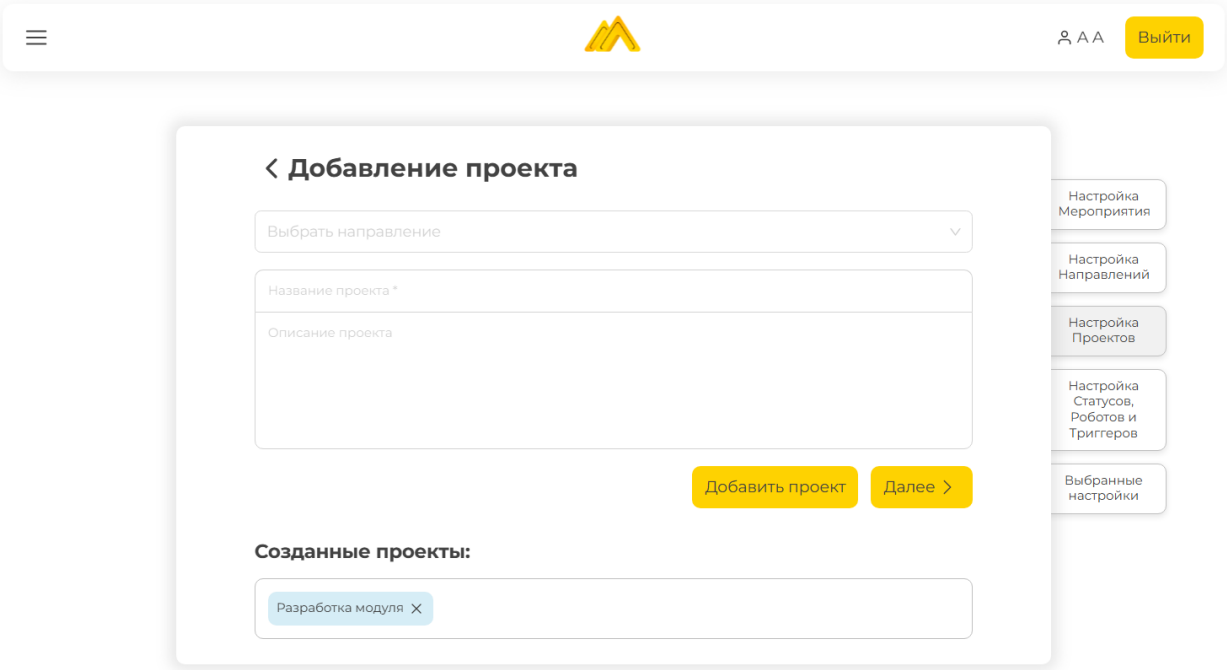

Рисунок Д.18 – Форма добавления проектов

Продолжение ПРИЛОЖЕНИЯ Д

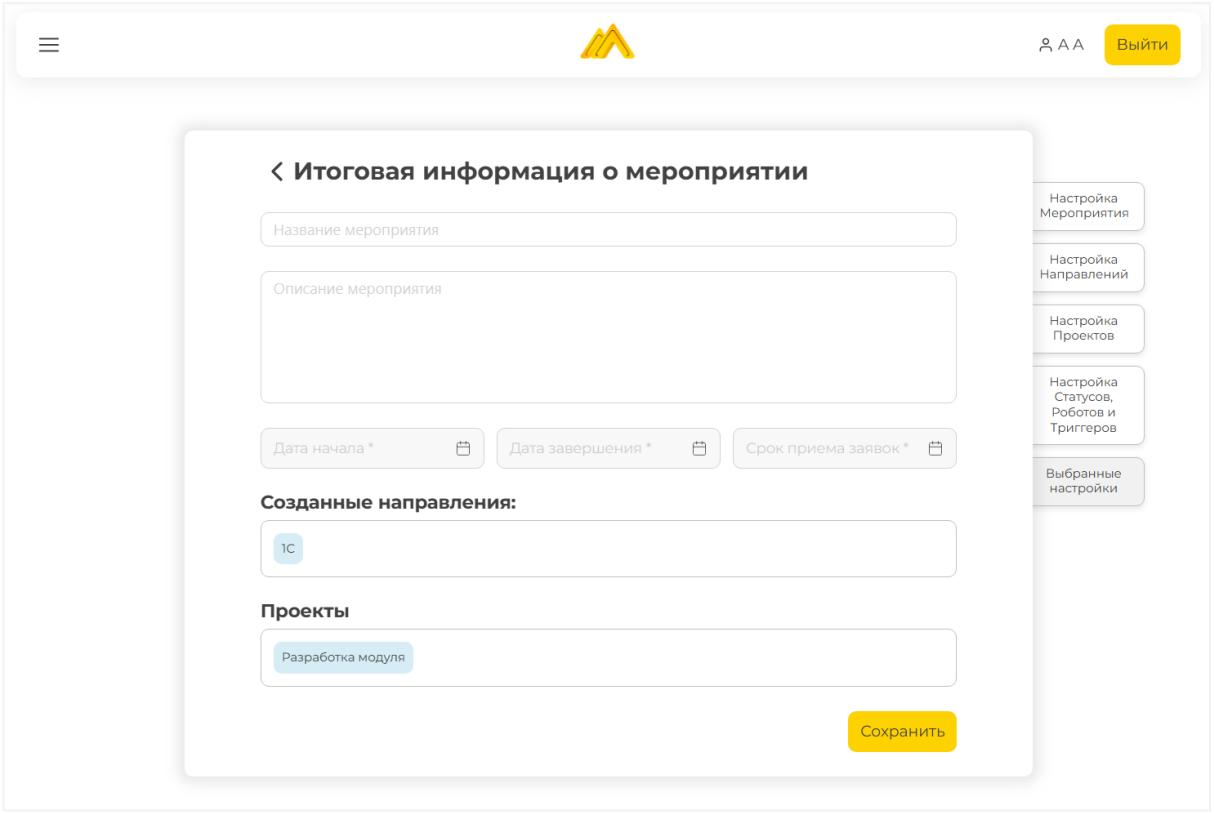

Рисунок Д.19 – Итоговая информация о мероприятии

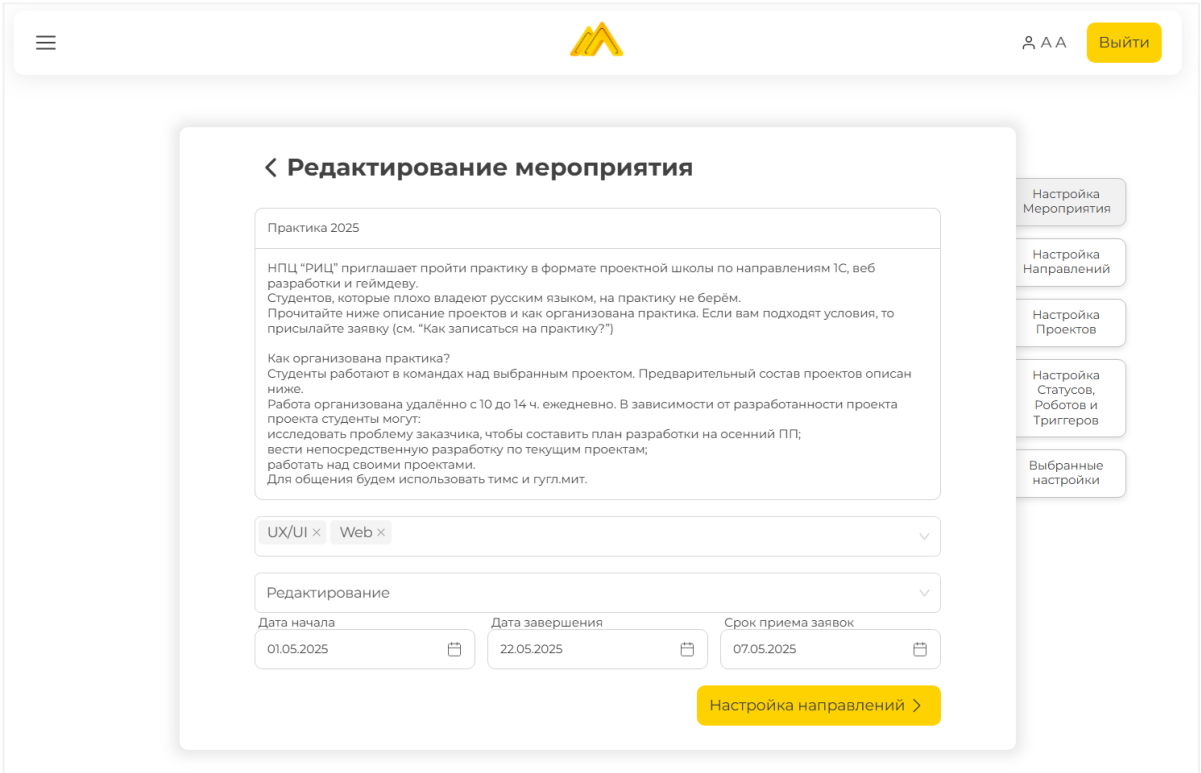

Рисунок Д.20 – Форма редактирования мероприятия

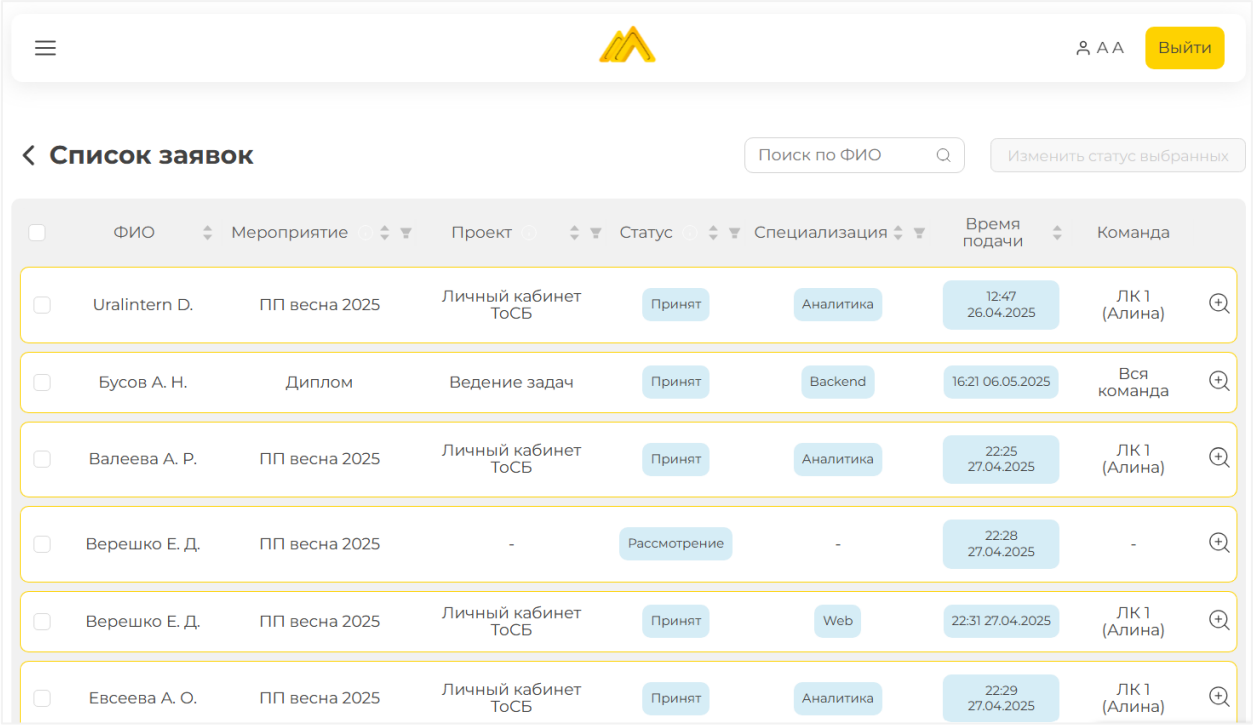

Рисунок Д.21 – Список поданых заявок

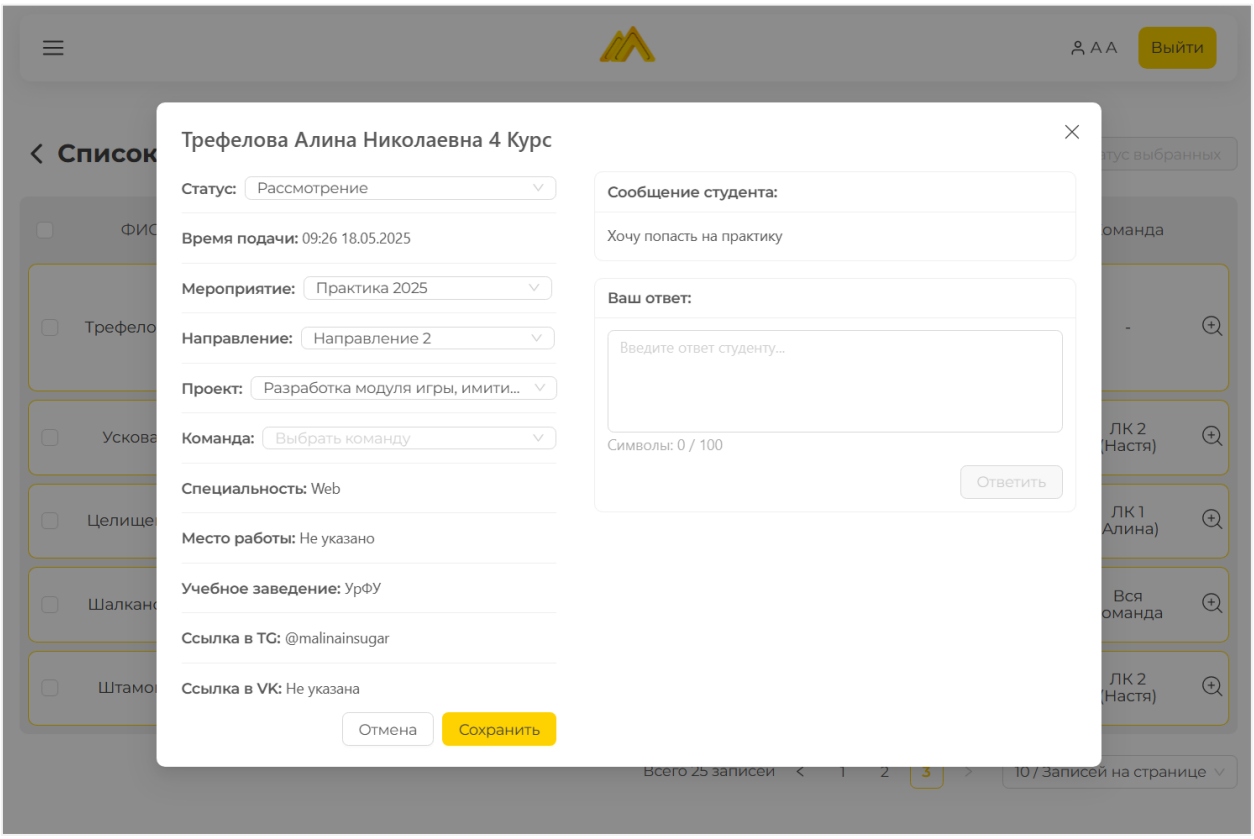

Рисунок Д.22 – Форма редактирования и просмотра заявки

Продолжение ПРИЛОЖЕНИЯ Д

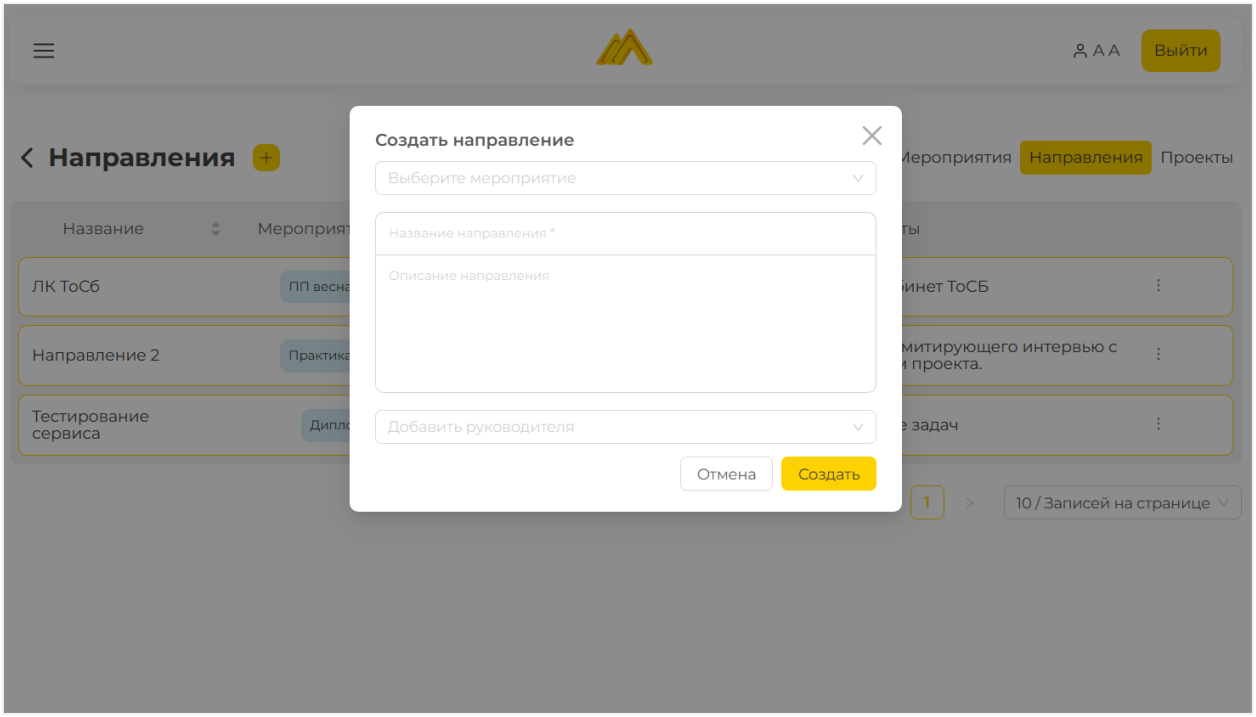

Рисунок Д.23 – Форма создания направления

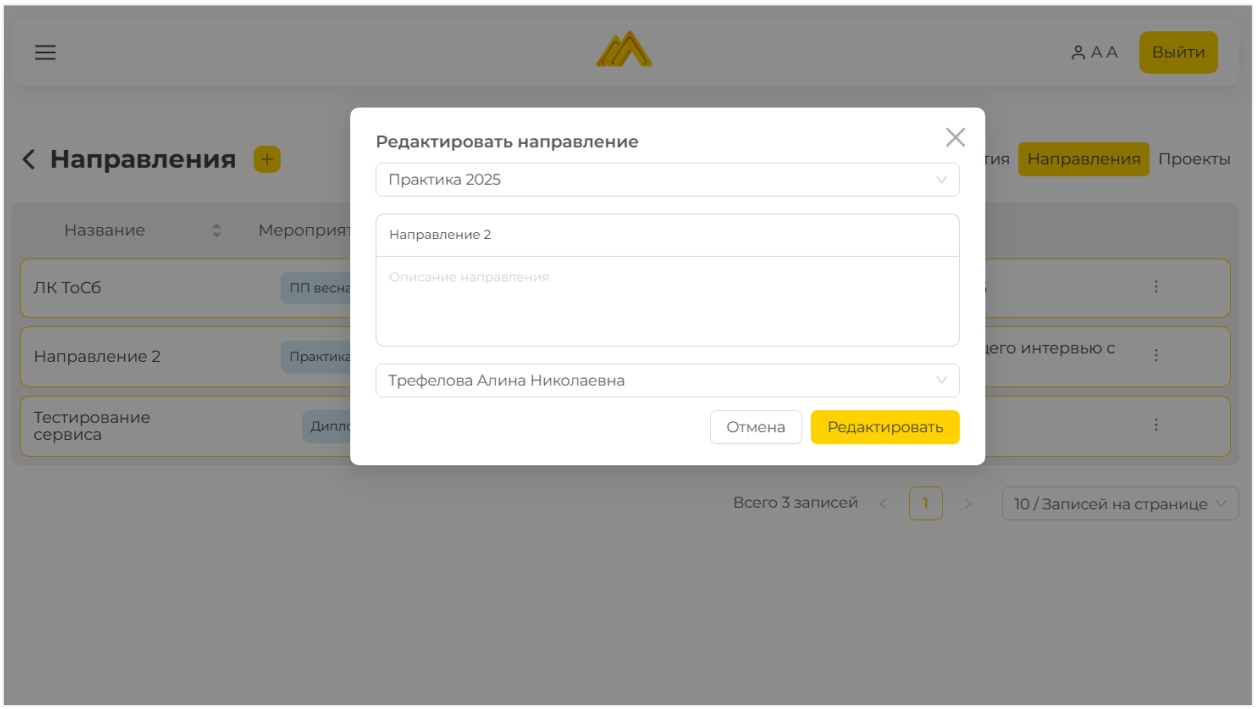

Рисунок Д.24 – Форма редактирования направления

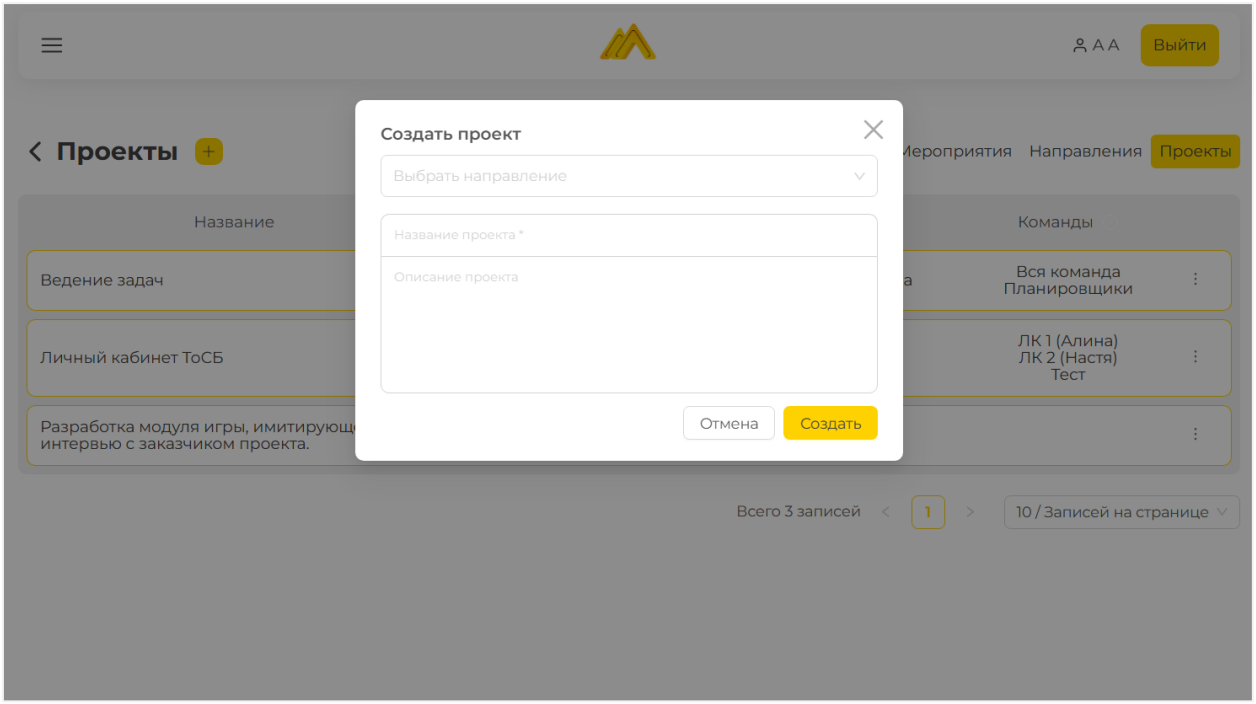

Рисунок Д.25 – Форма создания проекта

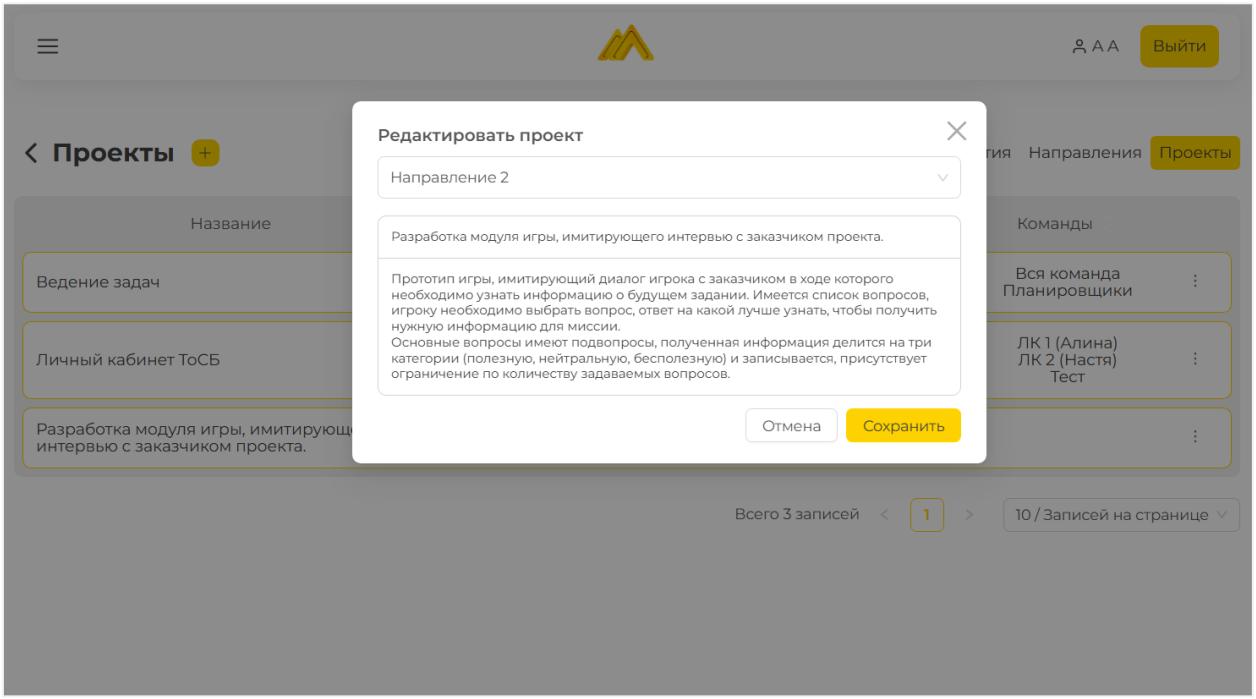

Рисунок Д.26 – Форма редактирования проекта

ПРИЛОЖЕНИЕ Е
(справочное)

Таблица с тестированием модуля

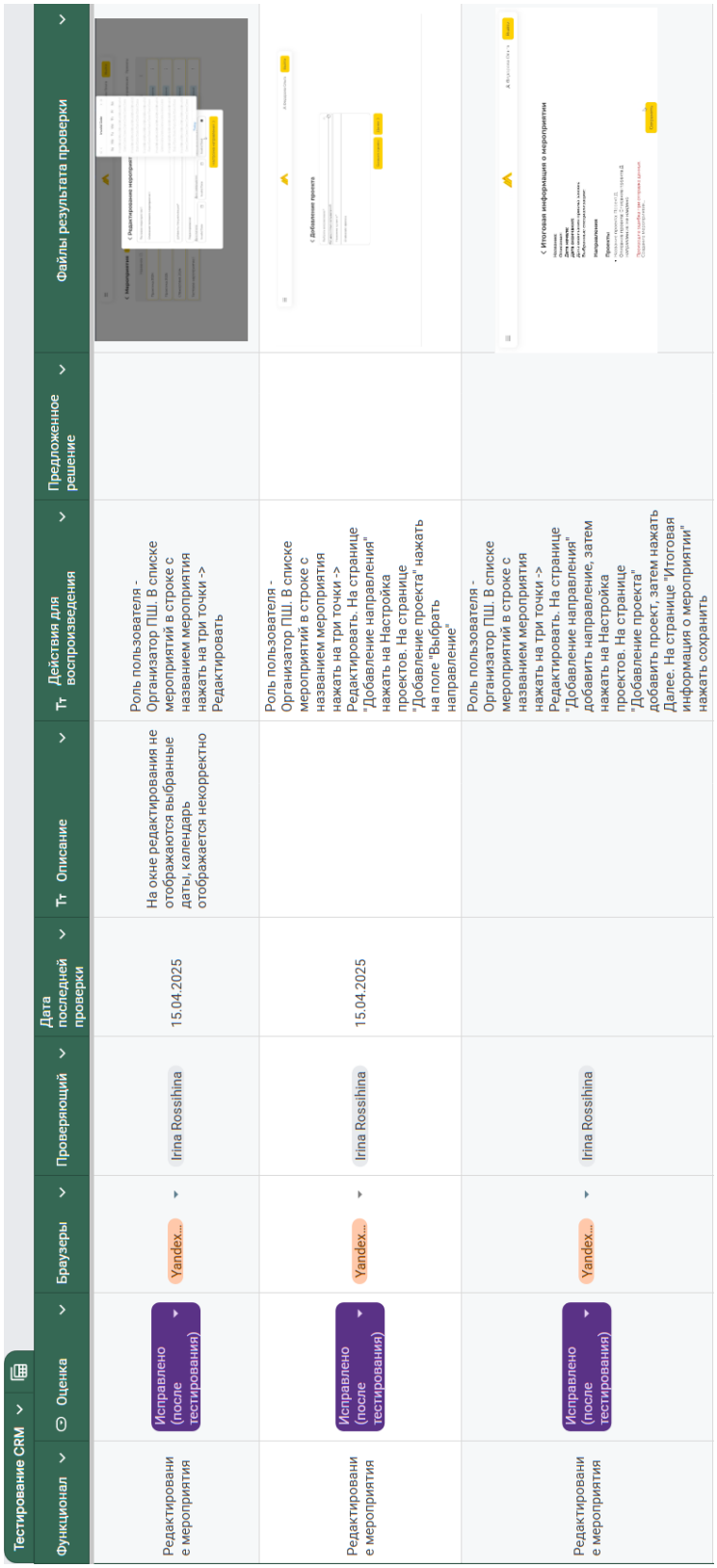

Рисунок Е.1 – Таблица для тестирования функций приложения внутри команды