МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г. ШУХОВА»

(БГТУ им. В.Г. Шухова)

Институт *информационных технологий и управляющих систем*

Кафедра *программного обеспечения вычислительной техники и автоматизированных систем*

Направление подготовки *09.03.04 – Программная инженерия*

Направленность (профиль) образовательной программы *Разработка программно-информационных систем*

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

на тему:

«Разработка front-end Web – приложения – учебной среды с чатами и AI-анализом кода лабораторных работ»

Студент: Бондаренко Сергей Владимирович

Зав. кафедрой: канд. техн. наук, доц. Поляков В.М.

Руководитель: Мельников А.Б.

К защите допустить:

Зав. кафедрой _____ /Поляков В.М./

«____» _____ 2025 г.

Белгород 2025 г.

МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г. ШУХОВА»

(БГТУ им. В.Г. Шухова)

Институт *информационных технологий и управляющих систем*

Кафедра *программного обеспечения вычислительной техники и автоматизированных систем*

Направление подготовки *09.03.04 – Программная инженерия*

Направленность (профиль) образовательной программы *Разработка программно-информационных систем*

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

на тему:

«Разработка front-end Web – приложения – учебной среды с чатами и AI-анализом кода лабораторных работ»

Студент: Бондаренко Сергей Владимирович

Зав. кафедрой: канд. техн. наук, доц. Поляков В.М.

Руководитель: Мельников А.Б.

К защите допустить:

Зав. кафедрой _____ /Поляков В.М./

«____» _____ 2025 г.

Белгород 2025 г.

ОПРЕДЕЛЕНИЯ, СОКРАЩЕНИЯ И ОБОЗНАЧЕНИЯ

- ИИ(AI) — искусственный интеллект.
- SPA (Single Page Application) — одностраничное веб-приложение, при котором навигация осуществляется без перезагрузки страниц.
- SSR (Server-Side Rendering) — рендеринг веб-страниц на стороне сервера перед отправкой клиенту.
- SSG (Static Site Generation) — предварительная генерация HTML-страниц во время сборки.
- JWT (JSON Web Token) — формат токенов для безопасной передачи информации между сторонами в виде JSON-объекта.
- FSD (Feature-Sliced Design) — архитектурный подход к построению front-end приложений, основанный на разделении по смысловым срезам.
- API (Application Programming Interface) — программный интерфейс для взаимодействия между компонентами программного обеспечения.
- UI (User Interface) — пользовательский интерфейс.
- React — библиотека JavaScript для построения пользовательских интерфейсов.
- Next.js — фреймворк на базе React с поддержкой SSR и SSG.
- TypeScript — надмножество JavaScript, добавляющее статическую типизацию.
- Socket.IO — библиотека для реализации двусторонней связи между клиентом и сервером в реальном времени.
- Auth.js — библиотека для реализации аутентификации и авторизации в приложениях на Next.js.

– Redux — библиотека для управления состоянием приложений.
– WebSocket — протокол для установления постоянного соединения между клиентом и сервером для обмена данными в реальном времени.

Содержание

Введение

Развитие цифровых технологий в сфере образования значительно меняет способы взаимодействия между преподавателями и студентами, предоставляя новые возможности для обучения и обмена информацией. В условиях дистанционного и смешанного обучения особенно важной становится необходимость создания платформ, которые бы объединяли образовательные инструменты в едином пространстве. Веб-приложения, которые решают задачи взаимодействия, позволяют сократить барьеры между преподавателями и студентами, улучшить коммуникацию и повысить качество образования. Цифровая среда должна обеспечивать не только размещение учебных материалов и заданий, но и средства для общения, автоматической оценки и анализа решений с использованием современных технологий, включая искусственный интеллект.

Актуальность темы заключается в потребности создания интегрированной образовательной платформы, которая объединяет функции чатов, проведения занятий и автоматического анализа решений, используя возможности ИИ. На данный момент отсутствует единая система, которая бы эффективно сочетала в себе эти ключевые аспекты: возможность общения через чаты, создание заданий и автоматизированную проверку решений с помощью ИИ. Современные платформы, как правило, фрагментированы — отдельные системы для чатов, другие для размещения заданий, третьи для автоматической проверки кода, что значительно усложняет организацию учебного процесса и снижает его эффективность. Разработка интегрированного решения, которое объединило бы эти элементы, позволяет улучшить качество образовательного процесса, повысив продуктивность студентов и преподавателей, а также упростив взаимодействие и автоматизировав многие рутинные задачи.

Целью данной работы является разработка клиентской части образовательной платформы, которая будет включать функции взаимодействия между преподавателями и студентами, автоматизированную проверку кода, а также возможности общения в рамках чатов. Особое внимание уделяется созданию такого

интерфейса, который позволит преподавателям и студентам взаимодействовать в едином пространстве, где будут доступны все образовательные инструменты и ресурсы.

Для достижения поставленной цели необходимо решить следующие задачи:

а) Проанализировать предметную область и существующие системы, выявив их сильные и слабые стороны.

б) Определить архитектурные и технологические решения, подходящие для реализации клиентской части платформы.

в) Спроектировать пользовательский интерфейс, обеспечивающий интуитивное и удобное взаимодействие для преподавателей и студентов.

г) Разработать компоненты для управления учебными структурами (институт, кафедра, группа), заданиями и чатами.

д) Интегрировать средства для автоматизированной проверки решений студентов с применением ИИ.

е) Реализовать тестирование бизнес-логики приложения для обеспечения её корректности и эффективности.

Структура пояснительной записки включает следующие разделы:

– В первом разделе рассматриваются особенности предметной области, проводится анализ существующих решений и обоснование выбора технологий и методов проектирования. Приводится обзор существующих образовательных платформ и их недостатков, а также объясняется необходимость разработки интегрированного решения.

– Во втором разделе описывается архитектура клиентской части приложения, структура пользовательского интерфейса, проектирование компонентов и их взаимодействие. Рассматриваются решения для реализации системы чатов, создания и проверки заданий, а также интеграции ИИ-анализа.

– В третьем разделе приводится описание реализации: структура кода, используемые технологии (Next.js, React, TypeScript, Redux, Auth.js), опи-

сание экрана и взаимодействий, примеры реализации различных компонентов системы.

– В заключении приводятся выводы по выполненной работе, оценивается эффективность разработанного интерфейса и функционала, а также определяются направления для дальнейшего развития и улучшения системы. Указываются перспективы внедрения ИИ в образовательные платформы для улучшения процессов оценки и взаимодействия.

1 ОБЗОР И ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Введение в предметную область

Современные образовательные процессы переживают значительные изменения под воздействием цифровых технологий. В условиях быстрого роста объемов информации и перехода на дистанционное и смешанное обучение возникает потребность в создании платформ, которые объединяют различные образовательные инструменты в единую систему. Проблемы, с которыми сталкиваются преподаватели и студенты, включают фрагментацию существующих решений: чаты для общения, отдельные системы для размещения и проверки заданий, а также инструменты для анализа решений студентов.

Существующие платформы не всегда обеспечивают интеграцию всех этих функций в одном приложении, что приводит к необходимости использования множества разных сервисов для выполнения учебных задач. В рамках образовательных процессов это усложняет взаимодействие между преподавателями и студентами, увеличивает время на организацию обучения и снижает его эффективность.

Одной из важнейших задач является создание платформы, которая объединяет все эти компоненты в одном месте, обеспечивая удобный интерфейс для студентов и преподавателей. Такая система должна включать:

- возможность создания и размещения учебных заданий;
- автоматическое тестирование решений студентов с использованием ИИ для проверки правильности кода;
- чат-функциональность для общения студентов с преподавателями и внутри групп;
- централизованный доступ к учебным материалам.

Интеграция всех этих функций в одну платформу позволит значитель-

но упростить организацию учебного процесса, улучшить взаимодействие между преподавателями и студентами, а также повысить качество обучения за счет автоматизации рутинных задач.

1.2 Анализ существующих образовательных платформ

Современные образовательные платформы, такие как Moodle, Google Classroom, Microsoft Teams для образования, а также специализированные решения, предназначенные для работы с программированием, предлагают различные функциональные возможности для взаимодействия преподавателей и студентов. Однако каждая из этих платформ имеет свои ограничения и не всегда покрывает все потребности в рамках единой системы.

1.2.1 Moodle

Moodle является одной из самых популярных образовательных платформ, используемых во многих учебных заведениях. Она предоставляет инструменты для размещения учебных материалов, организации тестов и заданий, а также ведения онлайн-курсов. Однако, несмотря на свои возможности, Moodle не предоставляет встроенных решений для автоматической проверки кода студентов, а также не включает в себя продвинутые механизмы общения в реальном времени, что делает её менее эффективной для динамичного взаимодействия в процессе обучения.

1.2.2 Google Classroom

Google Classroom предлагает простоту в использовании и позволяет интегрировать различные Google сервисы. Платформа позволяет преподавателям создавать задания, прикреплять материалы и отслеживать выполнение студентами. Однако Google Classroom не предоставляет функциональности для автоматического анализа решений, особенно в контексте программирования. Это требует интеграции с внешними инструментами, что усложняет использование системы в образовательных учреждениях.

1.2.3 Microsoft Teams for Education

Microsoft Teams, в отличие от Moodle и Google Classroom, активно используется для организации видеоконференций и групповых чатов. Он позволяет пре-

подавателям и студентам взаимодействовать в реальном времени, а также интегрирует различные сервисы Microsoft 365. Однако, как и в случае с другими платформами, Microsoft Teams не предоставляет функционала для интегрированного анализа кода студентов с использованием искусственного интеллекта, что ограничивает его возможности в обучении программированию.

1.2.4 Платформы для анализа кода

Существуют специализированные платформы, такие как CodeSignal, Codility, LeetCode, которые позволяют преподавателям и работодателям тестировать навыки программирования студентов. Эти системы используют алгоритмы для автоматической проверки решений, однако они ограничены в функционале взаимодействия с преподавателями и студентами, а также не обеспечивают централизованный доступ к учебным материалам и заданиям.

1.3 Проблемы существующих решений

Основной проблемой существующих образовательных платформ является фрагментация функционала. На данный момент нет единой платформы, которая бы эффективно объединяла создание и проверку заданий, общение преподавателей и студентов, а также использовала бы технологии ИИ для автоматизированного анализа решений студентов. Это затрудняет образовательный процесс и снижает его эффективность, особенно в условиях быстро меняющихся требований дистанционного обучения.

Таким образом, для улучшения образовательного процесса существует необходимость в разработке единой интегрированной платформы, которая бы сочетала в себе все эти компоненты и обеспечивала бы максимально удобное взаимодействие для всех участников учебного процесса.

1.4 Потребности образовательной среды

Современные образовательные процессы предъявляют высокие требования к функциональности учебных платформ. Для эффективного взаимодействия между преподавателями и студентами необходимо создавать приложения, которые обеспечивают организационную и техническую поддержку всех ключевых элементов образовательного процесса.

1.4.1 Доступность материалов и заданий

Материалы и задания должны быть доступны студентам в любое время. Приложение должно обеспечивать размещение учебных ресурсов в различных форматах (текст, видео, презентации) и упрощать их поиск и использование. Это позволяет студентам готовиться к занятиям и выполнять задания без привязки ко времени, а преподавателям — быстро обновлять и дополнять учебные модули.

1.4.2 Автоматизация проверок и оценки

Автоматизированная проверка заданий существенно ускоряет процесс получения обратной связи. Использование искусственного интеллекта для анализа кода позволяет выявлять ошибки, давать подсказки и оценивать работы без участия преподавателя. Это освобождает ресурсы преподавателя для индивидуальной поддержки студентов и более сложной экспертной оценки.

Удобная система заданий и общения

Приложение должно включать удобную систему создания и отслеживания заданий. Важно, чтобы преподаватели могли формулировать задания, прикреплять к ним материалы и получать результаты выполнения. Неотъемлемой частью также является возможность общения между участниками процесса — как в групповых, так и личных чатах, для обмена мнениями и получения поддержки.

Интеграция всех процессов в одну систему

Отдельные решения для чатов, размещения заданий и анализа кода создают фрагментированную среду. Необходима единая платформа, объединяющая все эти компоненты. Это упрощает взаимодействие, повышает удобство и эффективность обучения, а также снижает затраты на сопровождение и обучение работе с системой.

Вывод

Таким образом, при проектировании образовательной платформы следует учитывать потребности в постоянном доступе к материалам, автоматической проверке решений, поддержке взаимодействия и целостности функционала в рамках одного интерфейса.

1.5 Технологии разработки клиентской части приложения

Для реализации клиентской части платформы выбраны современные инструменты, обеспечивающие модульность, производительность, типизацию и масштабируемость интерфейса.

TypeScript

JavaScript является одним из самых популярных языков программирования для веб-разработки. Он широко используется для создания динамичных веб-страниц и приложений, поскольку позволяет работать с элементами DOM, асинхронно загружать данные и обеспечивать интерактивность пользовательских интерфейсов. Однако JavaScript имеет важный недостаток — отсутствие статической типизации. Это означает, что переменные и функции не привязываются к определённым типам данных, что может привести к ошибкам на этапе выполнения, которые трудно обнаружить в процессе разработки. Особенно это может быть проблемой в крупных приложениях, где сложно отслеживать все возможные типы данных и их изменения.

Для устранения этих проблем был разработан язык TypeScript, являющийся надмножеством JavaScript. TypeScript добавляет в JavaScript статическую типизацию, что позволяет разработчикам явно указывать типы данных для переменных и функций. Это значительно снижает вероятность ошибок и улучшает поддержку кода в будущем. Благодаря строгой типизации TypeScript помогает предотвращать баги, связанные с динамическими типами в JavaScript, и улучшает автозаполнение в редакторах кода. TypeScript распространяется как библиотека, которую можно интегрировать в проекты на JavaScript, обеспечивая совместимость с существующим кодом и позволяя постепенно внедрять типизацию без необходимости переписывать весь проект. Это особенно важно в крупных и масштабируемых приложениях, где несколько разработчиков работают с общими компонентами, и типизация помогает поддерживать консистентность кода на протяжении всего проекта.

React

React — библиотека для построения пользовательских интерфейсов, разработанная Facebook. Она широко используется в веб-разработке благодаря своей простоте, гибкости и высокой производительности. React обеспечивает декла-

тивный стиль программирования, при котором разработчик описывает, как должен выглядеть интерфейс при заданном состоянии, а библиотека самостоятельно обновляет DOM при изменениях. Это упрощает разработку сложных и динамичных интерфейсов.

Ключевые особенности:

- **Компонентный подход**: Приложение разбивается на переиспользуемые и изолированные компоненты
- **JSX-синтаксис**: Комбинация JavaScript и разметки, упрощающая написание UI
- **Virtual DOM**: Эффективное обновление только изменённых элементов страницы
- **Hooks API**: Современный способ управления состоянием и побочными эффектами

Преимущества для образовательных платформ:

- Быстрая разработка за счёт декларативности и компонентного подхода
- Большое сообщество и развитая экосистема (Next.js, Redux, React Query и др.)
- Поддержка SSR и SSG при использовании Next.js
- Простая интеграция с библиотеками и сторонними сервисами

Ограничения:

- Отсутствие встроенной архитектуры — требует выбора и настройки дополнительных инструментов
- Более низкий порог входа может привести к «разнообразию» архитектурных подходов в команде
- Без Next.js не включает такие возможности как маршрутизация, SSR и API

Angular

Angular — полноценный MVC-фреймворк, предоставляющий комплексное решение для разработки enterprise-приложений. В отличие от React, Angular накладывает строгую архитектурную модель, что обеспечивает единообразие кодовой базы в крупных проектах.

Ключевые особенности:

- **Двустороннее связывание данных**: Автоматическая синхронизация между моделью и представлением
- **Инъекция зависимостей**: Встроенный механизм для управления сервисами и их зависимостями
- **CLI-инструменты**: Генерация компонентов, сервисов и модулей через командную строку
- **TypeScript-first**: Полная поддержка статической типизации ”из коробки”

Преимущества для образовательных платформ:

- Строгая структура проекта для командной разработки
- Встроенная поддержка форм с валидацией
- Готовые решения для маршрутизации и HTTP-клиента

Ограничения:

- Высокий порог входа из-за сложной терминологии (декораторы, зоны, сервисы)
- Большой размер бандла (до 500КБ в минимальной сборке)
- Жёсткие требования к архитектуре

Vue.js

Vue.js — прогрессивный фреймворк, сочетающий подходы React и Angular. Особенно эффективен для быстрого прототипирования и проектов средней сложности.

Основные характеристики:

– **Реактивная система**: Автоматическое отслеживание зависимостей данных
– **Гибкая интеграция**: Возможность использования как через CDN, так и в составе сложных SPA
– **Single-File Components**: Объединение шаблона, логики и стилей в одном .vue-файле
– **Переходные анимации**: Встроенная поддержка анимации состояний

Сильные стороны для учебных проектов:

– Понятная документация с интерактивными примерами
– Мягкая кривая обучения для начинающих
– Компактный размер ядра (24КБ в gzip)

Ограничения:

– Относительно небольшое сообщество по сравнению с React/Angular
– Ограниченные возможности SSR без Nuxt.js
– Меньший выбор готовых UI-библиотек

Сравнительный анализ фреймворков

Таблица 1 – Сравнение характеристик фреймворков

Параметр	React + Next.js	Angular	Vue.js
Кривая обучения	Средняя	Высокая	Средняя
Сообщество	Крупное	Крупное	Растущее
Производительность	Высокая	Средняя	Высокая
Гибкость архитектуры	Высокая	Минимальная	Средняя
Поддержка SSR	Встроенная (Next.js)	Встроенная	Nuxt.js
Поддержка TypeScript	Да	Да	Да
Готовая маршрутизация	Да (Next.js)	Да	Да (Vue Router)

Ключевые выводы:

– **React + Next.js**: Предпочтителен для современных, SEO-оптимизированных приложений с гибкой архитектурой и возможностью инкрементального масштабирования
– **Angular**: Подходит для крупных enterprise-систем с чёткими архитектурными требованиями и строгой типизацией
– **Vue.js**: Оптимален для быстрого старта, MVP и небольших команд с ограниченным опытом

Для образовательной платформы выбран стек **React + Next.js**, поскольку он:

– Обеспечивает SSR и SSG «из коробки», что критично для SEO
– Позволяет гибко комбинировать клиентскую и серверную логику
– Имеет развитую экосистему и отличную интеграцию с библиотеками (Auth.js, Redux, Socket.IO)
– Упрощает масштабирование и поддержку проекта в долгосрочной перспективе

Next.js

Next.js — это популярный фреймворк для React, который значительно расширяет его возможности, предоставляя разработчикам мощные инструменты для создания высокопроизводительных веб-приложений. Одной из ключевых особенностей Next.js является поддержка рендеринга на сервере (SSR, Server-Side Rendering) и статической генерации контента (SSG, Static Site Generation). Эти подходы позволяют улучшить производительность приложений, поскольку они обеспечивают быструю загрузку страниц, оптимизированную для поисковых систем и пользователей.

С серверным рендерингом Next.js позволяет генерировать HTML на сервере для каждой страницы перед её отправкой клиенту, что обеспечивает быстрое отображение контента. Это особенно полезно для SEO, поскольку поисковые системы могут индексировать контент сразу после его загрузки. Такой подход значительно улучшает видимость веб-приложений в поисковых системах и способствует их более высокому ранжированию.

Одним из ключевых преимуществ Next.js является автоматическая разбивка кода (code splitting). Это означает, что Next.js разделяет приложение на небольшие части, которые загружаются только по мере необходимости, что помогает сократить время загрузки страниц и улучшить пользовательский опыт. Таким образом, браузер загружает только тот код, который необходим для отображения текущей страницы, а не весь код приложения.

Кроме того, Next.js поддерживает гибкие методы рендеринга, что дает разработчикам возможность выбирать наиболее подходящий способ для каждой страницы. Статическая генерация (SSG) идеально подходит для страниц, которые не изменяются часто и могут быть сгенерированы заранее, например, блоговые записи или страницы с информацией о компании. В то время как для динамических страниц, которые требуют актуализации данных на сервере при каждом запросе, можно использовать серверный рендеринг.

Next.js также упрощает настройку маршрутизации и управление данными. Встроенная система маршрутизации автоматически генерирует страницы на основе файловой структуры, что делает создание новых страниц и маршрутов простым и интуитивно понятным. Кроме того, Next.js предоставляет инструменты для работы с API, что позволяет без труда интегрировать серверную логику в приложение.

Еще одной значимой особенностью является поддержка типизации с помощью TypeScript, что делает разработку в Next.js ещё более удобной и безопасной. Комбинация TypeScript и Next.js позволяет создавать стабильные и хорошо структурированные приложения, минимизируя количество ошибок на этапе разработки.

Важно отметить, что реализацию приложения можно было бы построить и на чистом React, однако в этом случае значительная часть функциональности, такой как маршрутизация, SSR, SSG и работа с API, потребовала бы ручной настройки и подключения дополнительных библиотек. Использование Next.js избавляет от необходимости собирать всё вручную и предоставляет готовую, хорошо спроектированную архитектуру. Таким образом, Next.js становится де-факто стандартом разработки современных React-приложений. Это не просто библиотека, а фреймворк — а значит, он предлагает определённую «протоптанную дорожку», следование которой позволяет создавать более надёжные и поддерживаемые решения.

Socket.IO

Socket.IO — это JavaScript-библиотека с открытым исходным кодом, предназначенная для реализации двустороннего взаимодействия между клиентом и сервером в режиме реального времени. Основной особенностью данной технологии является использование собственного протокола поверх WebSocket с возможностью автоматического переключения на альтернативные методы связи (long polling и др.) при отсутствии поддержки WebSocket.

Преимущества использования Socket.IO:

- **Гибкость**: Разработчики имеют полный контроль над архитектурой соединения, включая маршрутизацию сообщений, обработку событий, систему комнат (rooms) и пространств имён (namespaces).
- **Масштабируемость**: Поддержка кластеризации и горизонтального масштабирования при помощи Redis-адаптеров.
- **Совместимость с Node.js**: Socket.IO органично интегрируется в стек на основе Node.js, что упрощает реализацию единой инфраструктуры.
- **Производительность**: Низкая задержка при передаче сообщений благодаря постоянному соединению между клиентом и сервером.
- **Интеграция с Python**: Для серверной части на Python существует аналогичная библиотека *python-socketio*, которая позволяет использовать те же возможности для реализации чатов и двустороннего общения между клиентом и сервером. Это делает возможным использование Socket.IO как на front-end (с помощью Node.js), так и на back-end (с помощью Python), обеспечивая совместимость и синхронизацию данных между клиентом и сервером.

Недостатки:

- Необходимость разработки и поддержки собственной серверной инфраструктуры.
- Повышенная сложность при масштабировании без использования внешних инструментов (Redis, Kubernetes).
- Отсутствие встроенной панели мониторинга или аналитики соединений.

Socket.IO предоставляет высокий уровень кастомизации и гибкости, что делает его предпочтительным выбором в проектах, где важна точная настройка логики взаимодействия в реальном времени.

Pusher

Pusher — это облачная платформа, предоставляющая API и SDK для реализации push-уведомлений и двусторонней передачи данных в реальном времени. В отличие от Socket.IO, Pusher представляет собой managed-сервис, абстрагирующий низкоуровневые детали инфраструктуры.

Преимущества использования Pusher:

- **Упрощённая интеграция**: SDK и готовые клиентские библиотеки позволяют быстро настроить соединение и передавать события.
- **Масштабируемость**: Обеспечивается на уровне платформы без участия разработчика.
- **Надёжность**: Pusher использует устойчивую облачную инфраструктуру с балансировкой нагрузки.
- **Аналитика и мониторинг**: Панель управления предоставляет данные о соединениях, событиях и каналах.

Ограничения:

- **Платная модель**: Бесплатный тариф ограничен по числу соединений и событий, что делает использование невыгодным при росте нагрузки.
- **Зависимость от стороннего сервиса**: Потенциальные риски, связанные с отказоустойчивостью внешнего провайдера.
- **Ограниченная кастомизация**: Структура событий и поведения определяется особенностями платформы.

Pusher является удобным решением для проектов с ограниченными временными ресурсами и отсутствием внутренней серверной инфраструктуры, однако его применимость в образовательных продуктах с ограниченным бюджетом вызывает сомнения.

WebSocket в JavaScript

JavaScript предоставляет встроенный класс `WebSocket` для организации двустороннего обмена данными между клиентом и сервером. Этот класс реализует базовую функциональность протокола WebSocket, предоставляя разработчикам простой способ обмена данными в реальном времени без необходимости использовать сторонние библиотеки. Однако, несмотря на свою доступность, использование `WebSocket` требует значительных усилий для реализации различных важных аспектов взаимодействия.

К примеру, при использовании `WebSocket` разработчик должен самостоятельно реализовать:

– переподключение сокета при его падении,

– буферизацию сообщений в случае разрыва соединения,

– обработку таймаутов и ошибок,

– поддержку различных сред (например, Node.js и браузер),

– масштабирование на сервере.

Таким образом, несмотря на его доступность и гибкость, использование `WebSocket` требует написания значительного объема дополнительной логики. Это делает его менее удобным для быстрого внедрения в проект, особенно в случае масштабируемых приложений.

С учетом всех этих факторов, было принято решение отказаться от использования низкоуровневого `WebSocket` в пользу более высокоуровневых решений, таких как `Socket.IO` или `Pusher`, которые предоставляют необходимую функциональность и обрабатывают множество нюансов «из коробки», позволяя сосредоточиться на бизнес-логике приложения.

Сравнение и выбор технологии для чатов

При сравнении библиотек Socket.IO и Pusher необходимо учитывать как технические, так и организационные аспекты. Таблица 2 представляет краткое сопоставление ключевых параметров:

С учётом специфики проекта — ограниченного бюджета, необходимости полной кастомизации и тесной интеграции с Node.js-сервером — наилучшим выбором является использование **Socket.IO**. Данная библиотека предоставляет

Таблица 2 – Сравнение технологий для реализации чатов в реальном времени

Критерий	**Socket.IO**	**Pusher**
Тип решения	Open-source библиотека	Облачный managed-сервис
Контроль над архитектурой	Полный	Ограниченный
Поддержка масштабирования	Через Redis и кластеризацию	Встроенная на уровне платформы
Простота настройки	Средняя (требует сервера)	Высокая (SDK)
Затраты на использование	Бесплатно	Платная модель
Интеграция с Node.js	Нативная	Через SDK
Надёжность соединения	Высокая	Высокая
Кастомизация протокола	Да	Нет

все необходимые механизмы для реализации масштабируемой и надёжной чат-системы, при этом позволяя оптимизировать производительность без привлечения сторонних сервисов.

Более того, благодаря открытому коду, Socket.IO не ограничивает разработчика в выборе архитектурных решений, а также обеспечивает возможность расширения функционала в будущем. В условиях ограниченных ресурсов образовательной платформы такой подход оказывается наиболее целесообразным.

Auth.js

Auth.js — это библиотека для реализации аутентификации и авторизации в веб-приложениях. Она является официальным решением, рекомендуемым и поддерживаемым фреймворком Next.js, что гарантирует хорошую интеграцию и поддержку всех необходимых функций. Библиотека позволяет легко подключать сторонние провайдеры аутентификации, такие как Google, Facebook и другие, а также реализовывать собственную аутентификацию с использованием базы данных. Auth.js обеспечивает надежную защиту пользовательских данных, управление сессиями, работу с токенами и предоставляет удобные API для быстрой настройки. Это решение упрощает реализацию всех ключевых механизмов безопасности, освобождая разработчиков от необходимости погружаться в тонкости реализации.

Redux

Redux — это библиотека для управления состоянием в приложениях, основанных на React. Она используется для централизованного хранения состояния приложения, что облегчает обмен данными между компонентами и упрощает их

взаимодействие. Redux помогает избежать ”проблемы пропс-дерева”в больших приложениях, когда передача данных через множество вложенных компонентов становится сложной. Хотя Redux часто используется в более сложных приложениях, в данном проекте его роль заключается в том, чтобы сделать взаимодействие между компонентами более организованным и улучшить предсказуемость состояния приложения.

1.6 Требования к функциональности приложения

Разрабатываемое приложение представляет собой образовательную платформу, ориентированную на университетскую среду. Основная цель — предоставить единое пространство для организации учебного процесса, взаимодействия между преподавателями и студентами, а также управления учебными структурами.

Регистрация и структура университетов

Каждый университет имеет возможность зарегистрироваться на платформе и получить доступ к собственной административной панели. Через неё администраторы могут создавать внутреннюю структуру: институты, кафедры и учебные группы. Эти сущности используются как основа для управления доступом, назначения преподавателей и приглашения студентов.

Панель преподавателя

Преподаватели, закреплённые за кафедрами, получают доступ ко всем учебным группам, относящимся к соответствующей кафедре. Через панель преподавателя доступен следующий функционал:

- создание групповых чатов для любой группы своей кафедры;
- размещение учебных материалов — как в групповых чатах, так и в личных сообщениях;
- формирование и отправка заданий для студентов;
- просмотр и анализ результатов выполнения заданий;
- предоставление обратной связи студентам.

Панель студента

Студенты, присоединенные к определенным группам, имеют доступ к:

– групповым чатам своей учебной группы;
– личной переписке с преподавателями;
– материалам, отправленным преподавателями;
– заданиям, опубликованным в рамках их группы;
– форме отправки решений и получению обратной связи.

1.7 Диаграмма вариантов использования

На рисунке представлена диаграмма вариантов использования, демонстрирующая взаимодействие пользователей с системой. Каждый пользователь (актёр) обладает определённым набором действий, доступных в рамках его роли:

– **Преподаватель** — создание и управление группами и заданиями, взаимодействие со студентами в чатах, а также анализ кода лабораторных работ с использованием AI;
– **Студент** — участие в групповых и личных чатах, выполнение заданий и просмотр результатов;
– **Администратор** — управление институтами, кафедрами, группами, студентами и преподавателями, а также создание приглашений в систему.

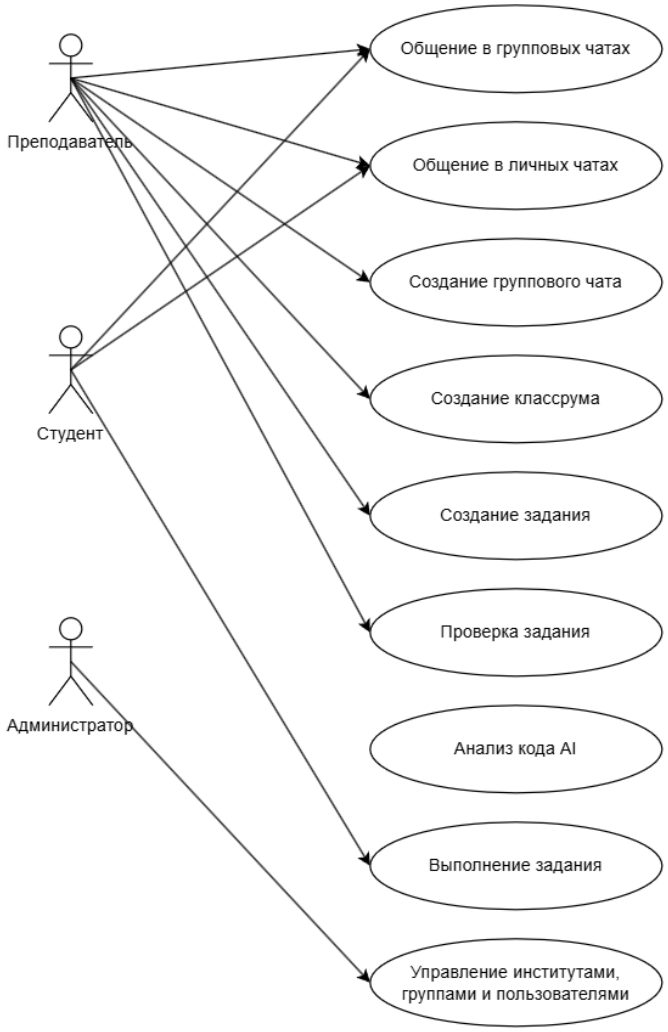

Рисунок 1 – Диаграмма вариантов использования системы для различных ролей пользователей.

1.8 Коммуникация и взаимодействие

Ключевым элементом платформы является система обмена сообщениями, включающая:

- **Групповые чаты**:
  - Общение участников учебной группы в режиме реального времени
  - Передача файлов (форматы: .jpg, .png, .mp4, .pdf, .zip и др.)
  - Привязка к группам
- **Личная переписка**:

– Обмен сообщениями один на один (студент-преподаватель или студент-студент)
– Передача файлов тех же форматов, что и в групповых чатах

Обе системы поддерживают базовые функции:

– Отображение истории сообщений
– Индикаторы прочтения сообщений
– Поиск по тексту переписки
– Уведомления о новых сообщениях

Таким образом, платформа предоставляет функциональность, охватывающую весь цикл учебной коммуникации — от административного управления структурами до взаимодействия по заданиям и материалам между преподавателями и студентами.

1.9 Безопасность данных

Авторизация с использованием JWT и Auth.js

Механизм авторизации в приложении реализован на основе JSON Web Token (JWT) и библиотеки Auth.js, которая обеспечивает безопасную и гибкую аутентификацию пользователей на стороне front-end. Процесс состоит из следующих этапов:

а) **Аутентификация пользователя**:

– Пользователь выполняет вход с помощью логина/пароля или через одного из OAuth-провайдеров (например, Google).
– Auth.js инициирует процесс аутентификации и, при успешной проверке, получает JWT.

б) **Работа с токеном**:

– Полученный JWT содержит минимальный необходимый payload (например, идентификатор пользователя, роль и срок действия).

– Токен сохраняется в `HTTP-only` cookie с флагами `Secure` и `SameSite=Strict`, что предотвращает XSS- и CSRF-атаки.

в) **Доступ к защищённым ресурсам**:

– При обращении к API, front-end автоматически прикрепляет токен к запросу.

– При недействительном или истёкшем токене, Auth.js может автоматически обновить его через refresh-токен, если он присутствует.

Роль Auth.js

Библиотека Auth.js упрощает реализацию безопасной авторизации за счёт следующих возможностей:

а) **Инкапсуляция логики**:

– Обрабатывает все основные сценарии авторизации (вход, выход, обновление токена).

– Управляет хранением и безопасной передачей токенов.

б) **Поддержка современных стандартов**:

– Генерирует CSRF-токены.

– Поддерживает стратегию Single Sign-On (SSO).

Меры защиты

Для повышения безопасности механизма авторизации реализованы дополнительные меры:

а) **JWT**:

– Access-токен действует ограниченное время (например, 5 минут), а refresh-токен — 30 дней.

б) **Auth.js**:

– Валидирует параметры входа, включая redirect URI.

Таким образом, связка JWT и Auth.js позволяет реализовать надёжный и гибкий механизм авторизации на стороне front-end с минимальной утечкой чувствительных данных.

1.10 Система создания заданий с AI-анализом кода

Общее описание модуля

Разработанная система позволяет преподавателям создавать виртуальные классрумы, выдавать задания и автоматически анализировать решения студентов с использованием AI-инструментов. Это аналог образовательной платформы (например, Google Classroom), ориентированный на технические дисциплины с программированием.

Основные функции:

– Создание виртуального класса преподавателем

– Назначение заданий с параметрами оценки

– Загрузка решений студентами

– Получение отчётов об автоматическом AI-анализе кода

– Просмотр статистики и аналитики преподавателем

Функциональные возможности

Для преподавателей:

– Создание и управление классами

– Назначение заданий

– Просмотр AI-отчётов по каждому студенту

– Сводная статистика по группе

Для студентов:

– Просмотр активных заданий и дедлайнов

– Загрузка решения

AI-анализ кода

После загрузки решения студентом система автоматически выполняет его анализ с использованием инструментов искусственного интеллекта. Проверка охватывает корректность, читаемость, соответствие заданию и уровень оригинальности кода.

На основе заданных преподавателем критериев формируется интерактивный отчёт, включающий:

- Комментарии и замечания по структуре и стилю кода
- Оценку соответствия решению поставленным требованиям

1.11 ИИ-анализ кода на основе DeepSeek

Общее описание модуля DeepSeek

DeepSeek — это современный облачный сервис для глубокого анализа исходного кода, построенный на основе передовых трансформерных моделей и нейронных сетей. Модуль предназначен для того, чтобы автоматизировать трудоёмкий процесс проверки решений студентов, снизить субъективность оценок и предоставить преподавателям максимально детализированную и качественную обратную связь.

Платформа DeepSeek способна:

- Выявлять самые разнообразные синтаксические и логические ошибки, обнаруживать неточности в реализации алгоритмов;
- Оценивать соответствие структурных блоков кода требованиям конкретного задания, обращая внимание на правильность использования функций и корректность их связи;
- Фиксировать стилевые отклонения и «code smells», которые могут усложнить поддержку и дальнейшее развитие проекта;
- Анализировать степень оригинальности решения с помощью семантического сравнения embedding-представлений, что позволяет не только обнаружить плагиат, но и оценить творческий подход студента.

Доступ к вызовам DeepSeek строго ограничен: только преподаватель, работая в защищённом интерфейсе фронтенда, может инициировать анализ кода.

Студенты видят лишь результат — отчёт с пояснениями и рекомендациями — без какой-либо информации о внутреннем устройстве сервиса.

Функциональные возможности DeepSeek

Для преподавателей:

- Возможность запускать анализ кода из единого интерфейса, не покидая окна браузера;
- Автоматически сгенерированные отчёты, где каждая найденная проблема снабжена подробным описанием и примером исправления;
- Графическое отображение метрик: сложность, уровень стиля, количество «code smells», показатели производительности;
- Механизм гибкой настройки критериев оценки — преподаватель может добавлять свои правила проверки в зависимости от характера задания;

Для студентов:

- Заявка на анализ через загрузку решения;
- Получение готового отчёта от преподавателя без прямого доступа к сервису.

Принцип работы DeepSeek

а) **Лексический и синтаксический разбор:** исходный код разбивается на токены, строится абстрактное синтаксическое дерево, на основе которого проводится первичный анализ.

б) **Семантический анализ:** алгоритмы трансформеров сопоставляют логику решения с огромной базой примеров, выявляя нетривиальные отклонения от оптимальной структуры.

в) **Расчет метрик:** рассчитываются показатели цикломатической сложности, глубины вложенности, соответствия coding standard и других параметров качества.

г) **Антиплагиатный компонент:** сравнение embedding-векторов загруженного решения с репозиторием эталонных и ранее проверенных работ для определения степени оригинальности.

д) **Формирование отчёта:** на выходе генерируется детальный JSON-документ, содержащий список найденных замечаний, метрик и чётких рекомендаций по улучшению.

1.12 Тестирование с использованием Jest

Jest — современный фреймворк для автоматизированного тестирования JavaScript/TypeScript-приложений, разработанный и поддерживаемый Facebook. Он идеально подходит для оценки бизнес-логики фронтенд-компонентов благодаря следующим преимуществам:

- **Минимальная настройка.** Работает «из коробки» без дополнительной конфигурации, что позволяет быстро приступить к написанию тестов.
- **Высокая производительность.** Параллельное выполнение тестов в изолированных средах сокращает время прогона и обеспечивает мгновенную обратную связь.
- **Отчёты по покрытию.** Встроенный сбор метрик покрытия кода помогает выявлять неохваченные участки и поддерживать высокий уровень качества.
- **Гибкое мокирование.** Лёгкая подмена модулей и функций с помощью mock-утилит упрощает эмуляцию сложных сценариев и зависимостей.
- **Snapshot-тесты.** Возможность сохранять «снимки» выходных данных функций или компонентов и автоматически отслеживать их изменения с течением времени.

Именно эти особенности делают Jest лучшим выбором для тестирования бизнес-логики: он упрощает разработку и сопровождение тестов, обеспечивает понятную диагностику ошибок и легко интегрируется в клиентскую архитектуру без лишних накладных расходов.

1.13 Выводы

В результате проведённого анализа можно сделать вывод о наличии устойчивого запроса на интегрированное образовательное приложение, способное решать сразу несколько ключевых задач. Современные платформы зачастую фокусируются либо на предоставлении учебных материалов, либо на коммуникации,

либо на автоматизации проверки знаний, при этом разрозненность этих функций создаёт неудобства для всех участников образовательного процесса.

Потребности преподавателей включают в себя удобное управление группами и заданиями, возможность оперативной обратной связи, загрузку и распространение материалов. Студентам, в свою очередь, важно иметь стабильный и понятный доступ к заданиям, личным сообщениям и учебным ресурсам, а также возможность взаимодействовать с преподавателями и одногруппниками в привычном цифровом формате.

Предлагаемое приложение должно закрыть этот разрыв, обеспечив единую среду, в которой объединены функции управления учебным процессом, общения, публикации и проверки заданий. Такой подход позволит повысить качество образовательного взаимодействия, сократить технические барьеры и обеспечить более высокую степень вовлечённости пользователей.

Таким образом, на основании проведённого анализа подтверждается необходимость разработки новой системы, в которой ключевые элементы образовательной среды будут интегрированы в одно приложение, удовлетворяющее современным требованиям пользователей.

2 ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОЕКТА

2.1 Архитектура клиентской части системы

Клиентская часть разрабатываемой системы реализована в виде **одностраничного приложения (SPA)** с использованием фреймворка **Next.js**, поддерживающего гибкий рендеринг — как на стороне клиента (CSR), так и на стороне сервера (SSR). Такое решение позволяет обеспечить как высокую производительность и отзывчивость пользовательского интерфейса, так и оптимизацию индексации содержимого поисковыми системами за счёт серверного рендеринга.

2.1.1 Общая структура архитектуры

Для организации структуры проекта был применён подход **Feature-Sliced Design (FSD)** — современная парадигма проектирования front-end приложений, ориентированная на модульность, масштабируемость и соответствие предметной области. В отличие от традиционных архитектур, основанных на технических слоях (например, разделение на компоненты, страницы или сервисы), FSD предполагает смысловое разделение приложения на функциональные модули, которые отражают реальные пользовательские сценарии и бизнес-логику.

Каждый модуль FSD отвечает за строго ограниченную область и содержит всё необходимое для своей работы: представление, поведение, взаимодействие с API и внутренние модели. Это способствует повышению читаемости и повторному использованию кода, а также упрощает сопровождение и развитие системы в долгосрочной перспективе.

2.1.2 Архитектурная диаграмма

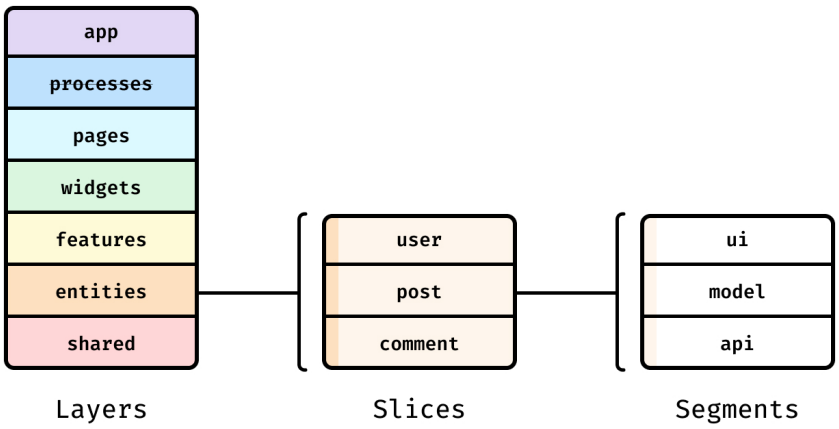

Рисунок 2 – Схема архитектуры клиентской части (FSD + Next.js)

На диаграмме представлены основные уровни и элементы архитектуры приложения. Следует отметить, что каждый слой в данной структуре ориентирован на строгое разграничение ответственности. Компоненты нижних уровней не имеют информации о вышестоящих слоях, что позволяет реализовать принцип инверсии зависимостей и минимизировать связанность между модулями.

2.1.3 Слои архитектуры Feature-Sliced Design

Таблица 3 – Слои архитектуры Feature-Sliced Design

Слой	**Описание и назначение**
`app`	Точка входа в приложение: глобальные стили, маршрутизация, провайдеры состояния, интеграции с внешними сервисами.
`pages`	Страницы, связанные с маршрутизацией. Формируются из виджетов и не содержат бизнес-логики.
`widgets`	Крупные элементы интерфейса, отражающие пользовательские сценарии, например, чат, список заданий, панель управления.
`features`	Изолированные пользовательские функции, такие как авторизация, отправка сообщений или регистрация. Могут включать бизнес-логику и вызовы API.
`entities`	Базовые предметные сущности предметной области, включающие типы, схемы, API и UI-представление.
`shared`	Универсальные компоненты, утилиты и типы, переиспользуемые во всём проекте.

2.1.4 Концепция срезов (slices)

Ключевым элементом архитектурного подхода Feature-Sliced Design является понятие **срезов** (англ. *slices*). Под срезом понимается логически обособленный модуль, реализующий завершённую часть функциональности приложения. Каждый срез может содержать собственные модели данных, визуальные компоненты, бизнес-логику, а также механизмы взаимодействия с внешними источниками данных.

Срезы группируются по смысловому признаку и могут располагаться в рамках различных уровней архитектуры: `features`, `entities`, `widgets`, `pages`. Такое структурирование обеспечивает лучшую декомпозицию кода, повышает его читабельность и облегчает модульное тестирование.

Например:

- `features/login` — срез, реализующий сценарий авторизации пользователя;
- `entities/task` — срез, содержащий всё, что связано с сущностью «задание»;
- `widgets/ChatWindow` — срез, объединяющий функциональность и интерфейс чат-интерфейса;
- `pages/home` — срез, реализующий главную страницу приложения.

Таким образом, построение приложения через срезы позволяет выстраивать архитектуру, ориентированную не на реализацию, а на назначение функциональных компонентов, что делает проект удобным для масштабирования и сопровождения.

2.1.5 Горизонтальное деление на сегменты (segments)

Каждый срез, независимо от своего уровня, может быть дополнительно разделён на **сегменты** (англ. *segments*) — логические подкатегории, структурирующие содержимое среза по назначению кода. В отличие от слоёв, которые представляют вертикальную иерархию, сегменты формируют горизонтальное деление и обеспечивают внутреннюю организацию модулей.

Наиболее распространённые типы сегментов включают:

- `ui` — визуальные компоненты и стили, определяющие отображение данных;
- `model` — модели данных, хранилища состояния, типизация и бизнес-логика;
- `api` — функции для работы с внешними сервисами, включая описание типов запросов и маппинг ответов;
- `lib` — вспомогательные функции и библиотеки, используемые в пределах данного среза;
- `config` — конфигурационные файлы и переключатели функциональности (feature flags).

Такой подход обеспечивает предсказуемую структуру каждого среза, делает код самодокументируемым и способствует упрощению навигации в большом проекте. Также, при необходимости, разработчики могут вводить дополнительные сегменты в рамках слоёв `shared` или `app`, не нарушая целостности архитектуры.

2.1.6 Преимущества выбранного подхода

Применение архитектуры Feature-Sliced Design в контексте разрабатываемого клиентского приложения позволило достичь следующих результатов:

- Чёткое разграничение обязанностей между модулями и слоями;
- Улучшенная масштабируемость проекта без деградации структуры;
- Повышенная модульность, обеспечивающая лёгкость в тестировании и повторном использовании кода;
- Создание условий для быстрой и эффективной интеграции новых членов команды в разработку;
- Архитектура, ориентированная на задачи и бизнес-логику, а не на технические детали.

В совокупности данные свойства делают архитектурное решение устойчивым к росту функциональности, улучшая поддержку и развитие системы в долгосрочной перспективе.

2.2 Проектирование интерфейсных подсистем и экранов

Одной из ключевых задач при проектировании клиентской части является логическое и функциональное разделение интерфейса на подсистемы, каждая из которых реализует отдельный аспект пользовательского взаимодействия. Такое разделение позволяет обеспечить модульность, переиспользуемость компонентов и устойчивость к изменениям.

Проект разрабатывается в архитектуре Feature-Sliced Design, что накладывает дополнительную дисциплину на организацию экранов и компонентов: все подсистемы формируются из `entities`, `features`, `widgets` и собираются в `pages`, а общая инфраструктура — в слое `shared`.

Выделение ключевых интерфейсных подсистем

Клиентская часть разработанной платформы организована в виде набора функционально обособленных интерфейсных подсистем, каждая из которых отвечает за определённый аспект пользовательского взаимодействия и бизнес-логики. Такое разграничение позволяет повысить масштабируемость и сопровождаемость системы, а также упростить процесс тестирования и внедрения новых функций.

На основании анализа требований к функциональности приложения и сценариев использования пользователями различных ролей (администратор, преподаватель, студент), были выделены следующие ключевые подсистемы.

а) **Подсистема авторизации и регистрации**

Отвечает за обеспечение безопасного входа в систему, регистрацию новых пользователей и управление сессиями. Аутентификация реализована с применением библиотеки `Auth.js` и технологии JSON Web Token (JWT), что позволяет надёжно разграничивать доступ к различным разделам интерфейса в зависимости от роли пользователя.

Регистрация в системе представлена в виде трёх пользовательских сценариев, адаптированных под особенности образовательного процесса:

- Первый сценарий реализован для новых организаций (институтов) и сопровождается созданием административной учётной записи. На этом этапе формируется корневая структура управления учреждением.
- Второй и третий сценарии предназначены для регистрации преподавателей и студентов соответственно. Оба сценария доступны исключительно по индивидуальным приглашениям, что обеспечивает контроль над составом участников образовательного процесса и предотвращает несанкционированный доступ.

Подсистема тесно связана с механизмами контроля прав доступа и маршрутизации, определяя поведение интерфейса в зависимости от текущего статуса пользователя.

б) **Подсистема управления университетом**

Реализует административную логику, связанную с конфигурацией организационной структуры образовательного учреждения.

Основными функциями данной подсистемы являются:

- Создание и удаление структурных единиц — институтов, кафедр, учебных групп;
- Управление персоналом: добавление и блокировка преподавателей и студентов;
- Генерация приглашений для входа новых участников на платформу с конкретной ролью;
- Отображение данных по структуре учреждения.

Визуально подсистема представлена в виде панели управления с множеством таблиц, форм и интерактивных элементов, обеспечивающих быстрый доступ к ключевым административным операциям. Все действия защищены авторизацией и доступны только пользователям с соответствующими правами доступа.

в) **Подсистема работы с заданиями и отправкой решений**

Данная подсистема предназначена для организации учебной деятельности: выдачи заданий преподавателями, загрузки решений студентами и автоматизированного анализа этих решений.

Основной интерфейс включает:

- Панель создания и редактирования заданий с параметрами проверки;
- Представление активных и завершённых заданий для студентов;
- Историю отправок с отображением результатов и статуса проверки.

Задания связаны с группами. Система также предоставляет базовую аналитику по результатам выполнения.

г) **Подсистема обмена сообщениями (чаты)**

В рамках образовательного процесса большое значение имеет возможность коммуникации. Подсистема реализует обмен сообщениями как в рамках учебной группы, так и в формате личной переписки. Технически реализация основана на технологии WebSocket с использованием библиотеки `Socket.IO`, что обеспечивает мгновенную доставку сообщений и минимальную задержку при передаче данных.

Основной функционал включает:

- Подключение к соответствующим «комнатам» (группам или диалогам);
- Отправку и приём текстовых сообщений;
- Отображение истории переписки;
- Поддержку вложений и индикаторов прочтения.

Доступ к системе чатов осуществляется только после успешной авторизации, что исключает участие анонимных пользователей и обеспечивает безопасность переписки.

д) **Подсистема AI-анализа решений**

Одной из уникальных особенностей платформы является использование искусственного интеллекта для автоматической оценки студенческих заданий.

Подсистема предназначена для получения и визуализации результатов AI-анализа, включающих:

- Оценку корректности кода;
- Проверку на соответствие заданию;
- Выявление потенциальных ошибок и некорректных конструкций;
- Комментарии, рекомендации и текстовые пояснения.

Результаты анализа отображаются в виде отчёта с возможностью преподавателя оставить дополнительные замечания. Таким образом, снижается нагрузка на преподавателя и повышается объективность оценивания.

Каждая из указанных подсистем обладает чётко определёнными входными и выходными данными, а также взаимодействует с другими модулями системы. Например, подсистема работы с заданиями напрямую связана как с AI-анализом, так и с интерфейсами преподавателя и студента, а система чатов — с механизмами авторизации и маршрутизации. Такое проектирование обеспечивает гибкость, надёжность и чёткую масштабируемость клиентской архитектуры.

Страницы и их структура

Разработка интерфейсной части веб-приложения требует не только реализации функциональных компонентов, но и проектирования логически связанных экранов, отражающих ключевые сценарии взаимодействия пользователя с системой. В рамках платформы каждая страница представляет собой самостоятельный интерфейсный модуль, обслуживающий одну или несколько бизнес-задач, соответствующих определённой роли: студент, преподаватель, администратор.

Процесс формирования страниц реализован с применением маршрутизации, встроенной в фреймворк `Next.js`, что обеспечивает высокую производительность и поддержку серверного рендеринга. Страницы не только представляют визуальный уровень приложения, но и координируют работу между компонентами пользовательского интерфейса, бизнес-логикой и хранилищем состояния.

Архитектурно страницы собираются из обособленных функциональных элементов, разработанных согласно принципам FSD: пользовательские действия реализуются в слое `features`, отображаемые сущности формируются на базе `entities`, а объединение этих блоков происходит внутри `widgets`. Такой подход позволяет повысить согласованность, переиспользуемость и модульность кода, а также снижает зависимость между различными частями интерфейса.

Ниже приведён перечень ключевых страниц, отражающих основную логику пользовательского взаимодействия.

– **Страница авторизации**

Отвечает за вход пользователя в систему. Содержит форму для ввода

учётных данных, а также реализует логику валидации, передачи данных на сервер, обработки ошибок и сохранения сессионного токена. После успешной авторизации пользователь перенаправляется на главную страницу, соответствующую его роли.

– **Страница заданий**
Представляет собой ключевой интерфейс для организации и выполнения учебной деятельности. Интерфейс страницы включает:

  – Список классов и учебных групп, к которым привязан пользователь;
  – Перечень активных заданий в рамках каждой группы;
  – Доступ к подробному описанию заданий, срокам сдачи и параметрам оценивания;
  – Отправку решений и просмотр результатов, включая отчёты AI-анализа.

Для преподавателя дополнительно предоставляется интерфейс управления заданиями, а также доступа к аналитике по группам и студентам.

– **Административная панель института**
Данная страница является основным рабочим инструментом пользователя с ролью администратора. Интерфейс включает:

  – Управление иерархией образовательного учреждения (институты, кафедры, группы);
  – Назначение и блокировка пользователей (студентов и преподавателей);
  – Просмотр структуры учреждения в табличной форме;
  – Генерация и отправка приглашений на регистрацию;
  – Журнал событий и контроль активности пользователей.

Все действия на данной странице требуют повышенного уровня доступа и сопровождаются системой уведомлений о результатах операций.

– **Страница чатов**

Реализует коммуникационную составляющую платформы. Пользователь получает доступ к:

– Перечню активных диалогов (личных и групповых);
– Истории сообщений в рамках выбранного чата;
– Форме для отправки сообщений и файлов;
– Интерактивным элементам: индикаторы доставки, статус прочтения, поиск по переписке.

Для преподавателей также предусмотрена возможность создания новых групповых чатов для своих учебных групп.

Все функциональные страницы приложения, за исключением экранов регистрации и входа, используют единый шаблон компоновки `AppLayout`, обеспечивающий целостность визуального восприятия и унификацию пользовательского опыта. Данный шаблон включает в себя общие элементы интерфейса — верхнюю панель навигации, боковое меню и основной контейнер для отображения содержимого, который динамически наполняется в зависимости от текущего маршрута.

Использование общего каркаса позволяет сохранить структурную согласованность между различными разделами системы, облегчает адаптацию пользователей к интерфейсу и упрощает внедрение изменений. Кроме того, архитектурное разделение логики и представления на уровне страниц способствует инкапсуляции ответственности, а также повышает читаемость и сопровождаемость кода. В рамках маршрутизации обеспечивается централизованное управление доступом, фильтрацией и визуализацией данных с учётом ролей пользователей.

Таким образом, структура страниц приложения отражает как технические требования архитектуры, так и практическую ориентацию на удобство и эффективность работы конечных пользователей.

Компоненты и принципы их структурирования

Компонентная модель проекта выстроена на основе принципов повторного использования, инкапсуляции и чёткого разделения ответственности между уров-

нями абстракции. Все компоненты, применяемые в рамках клиентского интерфейса, условно делятся на два основных класса: общие (универсальные) и специфические (бизнес-ориентированные).

– **Общие компоненты** (`shared/ui`) представляют собой переиспользуемые элементы пользовательского интерфейса, не зависящие от предметной области. К ним относятся кнопки, поля ввода, модальные окна, индикаторы загрузки, элементы навигации, уведомления и другие базовые визуальные элементы. Такие компоненты широко применяются на всех уровнях интерфейса и не содержат бизнес-логики.
– **Специфические компоненты**, разрабатываемые в слоях `entities` и `widgets`, предназначены для реализации прикладной логики и отображения конкретных сущностей системы. Примерами являются компоненты отображения сообщений в чате, карточек заданий, панели управления преподавателя, таблиц пользователей и др. Они обладают внутренним состоянием и часто включают обращение к хранилищу или API.

Такое структурное разграничение существенно упрощает масштабирование проекта, облегчает поддержку и повторное использование элементов, а также способствует разделению труда между разработчиками.

Распределение логики по слоям архитектуры

Функциональная логика клиентской части системы строго распределяется по слоям архитектуры Feature-Sliced Design, что обеспечивает высокую модульность и инкапсуляцию поведения. Каждому слою соответствует свой уровень ответственности:

– В слое `entities` сосредоточена модель предметной области: типизация, структура сущностей, атомарные компоненты отображения, такие как `Registration`, `Department`, `Group`. Данный слой реализует описание и базовое представление данных без привязки к конкретным действиям пользователя.
– Слой `features` содержит реализацию отдельных действий, составляющих пользовательские сценарии: отправка сообщений, регистрация,

загрузка задания, подтверждение действия и т.д. Эти модули инкапсулируют конкретные шаги взаимодействия пользователя с интерфейсом, часто включая локальное состояние и вызовы к API. `Features` могут быть использованы многократно и комбинироваться для построения более сложных сценариев.

– Слой `widgets` представляет собой реализацию полноценных пользовательских сценариев — законченных интерфейсных блоков, решающих определённую задачу. Примеры: интерфейс чата, панель с заданиями, административный модуль управления группами. Каждый виджет объединяет несколько фич и сущностей, обеспечивая завершённую и логически связанную единицу поведения.

– Слой `pages` выполняет роль точки входа и финальной сборки пользовательских сценариев. Здесь происходит выбор и компоновка виджетов в зависимости от маршрута, роли пользователя и контекста сессии. Кроме того, на уровне страниц задаются глобальные обёртки, обеспечиваются ограничения доступа, инициализируются загрузки данных и подключаются необходимые провайдеры. Таким образом, `pages` являются связующим слоем между навигацией и пользовательским опытом.

Такое строгое распределение обязанностей по слоям позволяет исключить дублирование логики, минимизировать связанность между модулями и обеспечить чёткую иерархию ответственности.

UX-решения и пользовательские сценарии

Для повышения удобства и доступности платформы, особенно в условиях использования её разными категориями пользователей, были реализованы следующие решения в области пользовательского опыта (UX):

– **Централизованная навигация** — через универсальный макет, включающий боковую и верхнюю панели, интерфейс остаётся единообразным и интуитивно понятным вне зависимости от текущего маршрута.

– **Toast-уведомления** — реализация мгновенной обратной связи при выполнении действий: успешная отправка формы, ошибка сети, получение новых сообщений.

– **Обработка пустых состояний и ошибок** — предусмотрены интерфейсы для ситуаций отсутствия данных, ошибок загрузки или недоступности сервера.

В результате, пользователь получает предсказуемый и непрерывный опыт взаимодействия с системой вне зависимости от своей роли и уровня подготовки.

Вывод

Проектирование интерфейсной части приложения основывается на чётком структурном и функциональном разграничении компонентов, ориентированном на принципы модульности и масштабируемости. Использование архитектуры Feature-Sliced Design позволяет изолировать бизнес-логику, визуальные компоненты и маршрутизацию, что делает интерфейс легко расширяемым и сопровождаемым.

Реализованная организация интерфейса, объединяющая единый шаблон компоновки, повторно используемые компоненты и специфические бизнес-модули, способствует формированию целостного пользовательского опыта. Выбранные UX-решения обеспечивают удобство и логичность навигации, а также высокую отзывчивость системы при взаимодействии с пользователем.

Интерфейсная часть проекта построена на модульной архитектуре, основанной на бизнес-функциях. Подсистемы выделены логически, а их реализация изолирована в независимые модули, что повышает удобство поддержки, расширения и переиспользования компонентов.

2.3 Проектирование взаимодействия с сервером и WebSocket

Клиентская часть приложения активно взаимодействует с сервером для получения и отправки данных, а также поддерживает постоянное соединение с помощью WebSocket в рамках подсистемы обмена сообщениями. При проектировании механизма взаимодействия были учтены требования безопасности, стабильности соединения, обработки ошибок, а также необходимость автоматического обновления сессионных данных пользователя.

Аутентификация и управление токенами

Для обеспечения защищённого доступа к функциональности платформы используется система авторизации с применением JSON Web Token (JWT). Управление сессией пользователя реализовано через библиотеку `Auth.js`, которая выполняет роль промежуточного слоя между клиентом и системой хранения токенов.

Поскольку access token не хранится в открытом виде на стороне клиента, его получение и обновление возможны только через специальные механизмы обращения к серверу. При первичном входе пользователя токен сохраняется в защищённой cookie. Для последующего взаимодействия клиенту необходимо перед каждым сетевым запросом удостовериться в актуальности токена.

В рамках проектирования был реализован механизм автоматической валидации access token. Каждый раз перед отправкой запроса выполняется проверка срока его действия. В случае, если срок истёк, инициируется обращение к серверу с целью его обновления через `Auth.js`. После получения нового токена он автоматически обновляется в cookie, и запрос выполняется повторно.

Дополнительно, учитывая асинхронную природу работы клиента, был предусмотрен механизм защиты от многократного обновления токена в случае параллельных запросов. Реализована единая точка обращения к логике проверки и обновления токена. Если несколько запросов запускаются одновременно, и access token требует обновления, то все они получают результат единого процесса рефреша, исключая избыточные сетевые обращения. Это позволяет сократить нагрузку на сервер и избежать конфликтов при замене токенов.

Унифицированная функция отправки запросов

Для стандартизации сетевого взаимодействия была разработана функция `sendRequest`, инкапсулирующая всю логику подготовки и отправки запросов к серверу. Она реализует следующие функции:

- Преобразование данных из формата JavaScript в JSON (`JSON.stringify`);
- Добавление заголовков, включая авторизационный `Authorization: Bearer`;
- Обработка возможных ошибок и повторная попытка в случае обновле-

ния токена;

– Поддержка различных HTTP-методов.

Данный подход позволяет централизованно управлять всей логикой сетевого взаимодействия и снижает вероятность ошибок при интеграции новых клиентских модулей.

Взаимодействие через WebSocket

Для реализации функциональности обмена сообщениями и других сценариев, требующих обновления данных в режиме реального времени, в клиентской части приложения применяется технология WebSocket. В отличие от традиционного HTTP-взаимодействия, WebSocket обеспечивает постоянное двустороннее соединение между клиентом и сервером, позволяя оперативно обмениваться событиями без необходимости постоянного опроса сервера.

В рамках проекта используется библиотека `Socket.IO`, которая предоставляет высокоуровневую обёртку над стандартным WebSocket-протоколом и значительно упрощает реализацию клиентского взаимодействия. Преимуществами `Socket.IO` являются:

– Поддержка автоматического переподключения при обрыве соединения;

– Передача структурированных событий с именами и аргументами;

– Интеграция с механизмами авторизации и middleware;

– Гибкость при работе с пространствами имён и комнатами;

– Совместимость с fallback-транспортами (в случае недоступности WebSocket).

На стороне клиента был разработан и реализован специализированный хук `useSocket`, инкапсулирующий всю логику работы с соединением. Он обеспечивает:

– Инициализацию соединения с сервером по заданному адресу;

– Отправку и приём событий с типизированной структурой данных;

– Автоматическую подписку и отписку от необходимых каналов;

– Управление жизненным циклом подключения;

– Обработку ошибок и логирование сетевых событий.

Для каждого подключённого пользователя создаётся уникальное пространство взаимодействия, определяемое его ролью и идентификатором. Это позволяет реализовать маршрутизацию сообщений между конкретными участниками чата (включая как личные, так и групповые диалоги), а также централизованно управлять доступом к отдельным коммуникационным потокам.

Особое внимание в проектировании было уделено вопросу авторизации в рамках WebSocket-сессии. В момент установления соединения клиент передаёт access token в параметрах подключения. На стороне сервера выполняется проверка подлинности токена, после чего соединение активируется. Однако, с учётом ограниченного срока действия access token, реализована логика автоматического переподключения. При обнаружении истечения токена:

а) Инициируется обновление токена через заранее определённый механизм;

б) Закрывается текущее соединение;

в) После получения нового токена создаётся новое подключение с обновлёнными параметрами.

Таким образом, WebSocket-подсистема функционирует устойчиво и прозрачно для конечного пользователя, обеспечивая бесперебойную передачу сообщений даже в условиях потери соединения или истечения сессии. Благодаря использованию `Socket.IO` удалось добиться высокой надёжности, расширяемости и лёгкости сопровождения реализации в рамках модульной архитектуры клиентской части.

Вывод

Взаимодействие клиентской части с сервером реализовано с учётом требований к безопасности, надёжности и масштабируемости. Реализованные механизмы автоматического обновления токенов, централизованная отправка запросов и продуманная интеграция WebSocket-соединения позволяют обеспечить стабильную работу интерфейса и корректную обработку всех пользовательских сценариев в режиме реального времени.

2.4 Интеграция ИИ-модуля DeepSeek в архитектуру проекта

Контейнеризация DeepSeek

Для обеспечения независимого жизненного цикла и лёгкой масштабируемости ИИ-компонента DeepSeek развёртывается в виде изолированного Docker-контейнера. Такой подход позволяет:

- Быстро запускать и останавливать сервис без влияния на основное приложение.
- Поддерживать разные версии DeepSeek параллельно, экспериментируя с обновлениями моделей.
- Мигрировать между хостами и облачными средами с минимальными изменениями конфигурации.

Контейнер содержит все необходимые зависимости: предобученные трансформерные модели, библиотеки для анализа AST и семантической обработки, а также механизмы формирования отчётов. При этом фронтенд остаётся полностью изолированным от этих деталей — ему достаточно знать только адрес и формат запросов.

Взаимодействие клиентской части с DeepSeek

Клиентская часть приложения, реализованная на React и Next.js, отправляет HTTP-запросы через единый API-шлюз. Основная логика взаимодействия упрощена до следующих действий:

- Преподаватель выбирает работу студента и нажатием кнопки инициирует анализ.
- Фронтенд направляет корректно сформированный запрос к API, где указаны идентификатор работы и необходимые параметры оценки.
- По готовности отчёта фронтенд периодически проверяет статус анализа и загружает результаты в привычном формате.
- Все шаги интегрированы в единый пользовательский поток: преподаватель не видит инфраструктуры контейнеров, а получает только готовый отчёт.

Такая схема взаимодействия гарантирует простоту клиентской логики и минимальную связность с внутренним устройством ИИ-сервиса.

Преимущества контейнеризированного подхода

Контейнеризация DeepSeek даёт следующие ключевые плюсы:

- **Изоляция нагрузки:** анализ кода выполняется в отдельном окружении, не влияя на отзывчивость пользовательского интерфейса.
- **Горизонтальное масштабирование:** при необходимости обрабатывать большое число запросов можно запускать несколько инстансов контейнера.
- **Упрощённое сопровождение:** обновление ИИ-компонента сводится к выпуску нового Docker-образа без правок в фронтенде.
- **Гибкость развертывания:** контейнеры можно запускать как локально, так и в облаке, сохраняя одинаковую конфигурацию.

Вывод

Таким образом, архитектура остаётся прозрачной на уровне клиентского приложения, а DeepSeek надёжно инкапсулирован и готов к дальнейшему эволюционному развитию.

2.5 Обеспечение безопасности клиентской части

Безопасность пользовательского взаимодействия является важнейшей составляющей архитектуры клиентской части платформы. В условиях, когда доступ к различным модулям приложения осуществляется на основе ролей, а взаимодействие с данными сопровождается отображением пользовательского контента, особое внимание уделяется как управлению доступом, так и защите от потенциальных атак, включая межсайтовое выполнение скриптов (XSS). В данной подсистеме реализован комплекс механизмов, направленных на защиту данных и поведения интерфейса со стороны клиента.

Механизм разграничения доступа с использованием Middleware

Одним из ключевых компонентов обеспечения безопасности клиентской части является система контроля доступа на основе промежуточного слоя —

middleware. В рамках архитектуры Next.js, middleware представляет собой функцию, исполняемую при каждом запросе к защищённым маршрутам. Она позволяет перехватывать обращения к страницам до их рендеринга и на этой стадии выполнять необходимые проверки: наличие токена, его валидность, а также права пользователя.

В контексте реализуемой платформы, при обращении пользователя к любой защищённой странице клиентская логика через middleware извлекает JWT-токен из cookies и дешифрует его содержимое, получая так называемый payload — полезную нагрузку токена. Внутри неё содержится вся необходимая информация о пользователе: уникальный идентификатор, срок действия сессии и, что особенно важно, роль в системе (admin, teacher, student).

На основе этой информации middleware выполняет следующие действия:

- Если пользователь не авторизован (отсутствует валидный токен) — происходит автоматический редирект на страницу входа.
- Если пользователь авторизован, но не обладает достаточными правами — осуществляется перенаправление на главную страницу или отображается сообщение об отказе в доступе.
- Если пользователь обладает необходимой ролью — доступ к ресурсу предоставляется, и страница загружается с соответствующим контентом.

Таким образом, механизм middleware обеспечивает надёжную фильтрацию обращений к различным частям интерфейса, предотвращая несанкционированный доступ и обеспечивая соответствие поведения приложения политике разграничения прав.

Роль и защита при работе с форматируемым текстом

Дополнительным вектором потенциальной угрозы в клиентских приложениях является отображение форматируемого текста, особенно если пользователь имеет возможность редактировать его содержимое. В таких случаях возрастает риск внедрения вредоносных скриптов, замаскированных под обычный HTML.

Для решения данной задачи в проекте используется специализированная библиотека tiptap — расширяемый редактор форматированного текста, основанный на ProseMirror. Одним из ключевых преимуществ tiptap является

контроль над тем, какие HTML-теги и атрибуты допускаются к интерпретации и отображению. Таким образом, даже если пользователь попытается вставить опасный HTML-код (например, `<script>` или инъекцию с обработчиком событий), редактор проигнорирует или удалит такие элементы на этапе парсинга.

Технически это реализуется следующим образом:

– При вводе текстов редактор не сохраняет «сырые» HTML-строки, а формирует безопасное представление контента на основе строго описанных схем;

– При рендеринге текста из базы или состояния редактор отображает только те элементы, которые были описаны как допустимые;

– Расширения (extensions), добавляемые к `tiptap`, позволяют точно контролировать список разрешённых действий и структур (например, разрешить только `<strong>`, `<em>`, `<ul>` и `<code>`).

Таким образом, даже при наличии активной формы редактирования форматируемого текста, пользовательская среда остаётся защищённой от внедрения опасного контента. Редактор фактически выступает в роли фильтра, строго ограничивающего возможный набор HTML-инструкций.

Вывод

Комплекс реализованных решений позволяет эффективно защитить клиентскую часть приложения как от внешнего вмешательства, так и от ошибочного доступа со стороны пользователей. Механизм middleware обеспечивает надёжную проверку подлинности сессии и прав доступа до загрузки страниц, а редактор `tiptap` гарантирует безопасность при работе с форматируемым текстом. Такое сочетание архитектурных и прикладных решений способствует созданию устойчивой и безопасной пользовательской среды.

2.6 Покрытие бизнес-логики юнит-тестами

Наш подход к обеспечению надёжности клиентского приложения фокусируется на обязательном юнит-тестировании бизнес-логики при помощи Jest. Тестирование UI-компонентов считается вторичным: написание и поддержка сравнений HTML-вывода часто оказывается более трудоёмким и хрупким, чем простая визуальная валидация. Визуальный осмотр интерфейса преподавателем или

дизайнером даёт более быстрый и надёжный результат без лишних накладных расходов.

Основные принципы нашего подхода:

- **Фокус на бизнес-логике.** Юнит-тесты покрывают функции, отвечающие за валидацию данных, расчёт оценок и другие критичные механизмы, гарантируя корректность работы независимо от изменений UI.
- **Минимизация тестов UI.** Модульные тесты интерфейсов не используются: динамика верстки и частые мелкие правки приводят к избыточным провалам тестов и дополнительным усилиям на их поддержку.
- **Простота поддержки.** Благодаря отказу от snapshot-тестирования HTML структура кода остаётся гибкой, а команда освобождает время на развитие функциональности вместо постоянной правки тестов.
- **Интеграция с CI/CD.** Автоматический запуск тестов бизнес-логики при каждом пуше позволяет мгновенно обнаруживать регрессии и поддерживать стабильность продукта.
- **Визуальная проверка.** Для окончательной валидации интерфейса используется ручной осмотр ключевых страниц после сборки, что даёт уверенность в корректности отображения без сложных технических средств.

Такой подход обеспечивает надёжность самой логики приложения и упрощает работу с UI: вместо громоздких автоматизированных тестов на вёрстку мы применяем человеческую экспертизу для финальной проверки внешнего вида и пользовательского опыта.

3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

3.1 Архитектура по Feature-Sliced Design

Feature-Sliced Design (FSD) — это гибкий набор рекомендаций и подходов по логической организации клиентского кода, основанных на выделении независимых функциональных слоёв и зон ответственности. В отличие от строгих стандартов, FSD предоставляет разработчикам свободу выбора конкретных решений, сохраняя при этом единый общий каркас структуры. Такая архитектура повышает читаемость, масштабируемость и тестируемость приложения, а также упрощает командную разработку и поддержку кода.

3.1.1 Слой `shared`

Слой `shared` служит хранилищем нижнего уровня для общих и переиспользуемых компонентов, утилит и ресурсов, не зависящих от конкретной бизнес-логики:

- **UI-компоненты общего назначения**: простые React-компоненты (кнопки, лоадеры, таблицы, модальные окна), которые не содержат бизнес-логики и могут использоваться в различных контекстах;
- **Провайдеры контекста**: `DragAndDropFilesProvider`, `EnterKeyHandlerProvider`, а также другие контексты, обеспечивающие единообразную работу с событиями и состояниями на уровне всего приложения;
- **Утилиты и хелперы**: функции для форматирования дат и чисел, генерации уникальных идентификаторов, работы с localStorage и пр.;
- **Кастомные хуки общего назначения**: `useSearchParamsListener`, `useWindowSize`, `usePreviousValue` и другие, позволяющие сократить дублирование кода;
- **SVG-иконки и графика**: импорт через SVGR для единообразного подключения и управления атрибутами SVG;

– **Компоненты навигации и управления состоянием**: `PaginationComponent` для управления пагинацией через URL-параметры, `Breadcrumbs`, `Tabs` и т. д.

Каждый элемент слоя `shared` сопровождается подробной документацией и примерами использования, что ускоряет обучение новых участников команды и снижает риски ошибок.

3.1.2 Слой `entities`

Слой `entities` отвечает за интеграцию с внешними сервисами и описывает доменные модели:

– `api/`: тонкий слой-абстракция над HTTP-клиентами (`fetch`/`axios`), где функции названы в соответствии с операциями Swagger/OpenAPI (например, `getUserProfile`, `createOrder`);
– `types/`: TypeScript-интерфейсы и типы для запросов и ответов (E.g. `UserProfileResponseType`, `OrderCreateRequestBodyType`);
– `models/`: классы и мапперы (mapper) для преобразования сырых данных из API в объекты, удобные для работы в приложении;
– `services/`: дополнительные обёртки для работы с локальным кэшем (IndexedDB, localStorage) или реализации retry-логики и таймаутов;

Кроме того, здесь же располагаются функции для обработки ошибок и логирования, а также константы (URL-ы, ключи хранящихся данных) для единообразия использования.

3.1.3 Слои `features`, `widgets`, `pages`

Главные рабочие слои приложения, отвечающие за реализацию конкретной функциональности:

а) **`pages`**: маршрутизация и верхний уровень страницы. Здесь описываются пути, разрешения доступа (guards), асинхронная загрузка данных на уровне страницы и выбор необходимого набора виджетов;

б) **`widgets`**: презентационные и «умные» компоненты, разделённые по Smart/Presentational-паттерну:

- *Presentational*: исключительно UI и пропсы, без прямой работы с API и глобальным состоянием;
- *Smart*: обёртки, получающие данные через хуки из `features`, передающие результат в презентационные компоненты;

в) **`features`**: бизнес-логика и состояние, реализованное в виде кастомных хуков, функций-редьюсеров, слайсов Redux или Zustand:

- `hooks.ts`: главный хук-фабрика (например, `useRegistrationUniversity`);
- `schema.ts`: схемы валидации через `Yup` или `Zod`;
- `utils.ts`: вспомогательные функции и вычисляемые селекторы;
- `store.ts`: подключение к Redux/Redux Toolkit или настройка Zustand;

Пример процесса разработки новой функции: сначала добавляем маршрут и контекст в `pages`, затем создаём презентационный виджет, выносим всю логику в хук в `features`, настраиваем валидацию и состояние, и наконец интегрируем с API через слой `entities`.

3.1.4 Пример: RegistrationUniversity

Листинг 1: RegistrationUniversityWidget

```
// Presentationкомпонент-
export function RegistrationUniversityWidget() {
  const { formDataRef, isError, setIsError, onSubmit, isLoading } =
  useRegistrationUniversity();

  return (
  <Card>
  <CardContent>
  <RegistrationForm
```

```
    schema={universitySchema({ password: formDataRef.current?.password })}
    onChangeFormData={data => (formDataRef.current = data)}
    onChangeIsError={setIsError}
    />
    <Button
    text={isLoading ? 'Загрузка...' : 'Зарегистрировать'}
    disabled={isError.length > 0 || isLoading}
    onClick={onSubmit}
    />
    </CardContent>
    </Card>
    );
  }
```

Листинг 2: useRegistrationUniversity

```
// Smartкомпонент-: хукфабрика-
export function useRegistrationUniversity() {
  const formDataRef = useRef<RegistrationData>();
  const [isError, setIsError] = useState<string[]>([]);
  const [isLoading, setIsLoading] = useState(false);
  const router = useRouter();

  const onSubmit = async () => {
    const data = formDataRef.current;
    if (!data) return;
    setIsLoading(true);
    try {
      const res = await registerUniversity(data);
      if (res.ok) {
        router.push('/admin');
      } else {
        // обработка ошибок от API
        setIsError([res.errorMessage || 'Неизвестная ошибка']);
      }
    } catch (e) {
      setIsError(['Сетевая ошибка, попробуйте позже']);
    } finally {
      setIsLoading(false);
    }
  };

  return { formDataRef, isError, setIsError, onSubmit, isLoading };
}
```

3.1.5 Соглашения по неймингу и структуре

Для поддержания высокого уровня единого стиля и предсказуемости структуры проекта:

- Имена функций API и типов дублируют бекенд-операции;
- Компоненты и хуки получают префиксы по зоне ответственности: `RegistrationForm`, `useFilesUpload` и т. д.;
- В каждом каталоге `features/FeatureName` обязателен минимум файлов: `index.ts`, `hooks.ts`, `schema.ts`, `utils.ts`;
- Все слои должны быть описаны в README с диаграммой и ссылками на примеры использования;
- Код ревью включает проверку соответствия FSD-подходу и правил TypeScript;

3.2 Собственная UI-библиотека и генерация форм

3.2.1 Общая идея и мотивация

Для обеспечения единого стилистического и функционального каркаса клиентского приложения была разработана собственная библиотека компонентов и утилит, распространяемая через npm-пакет. В её составе присутствуют:

- SCSS-миксины для реализации адаптивных сеток и гибкой типографики;
- Набор готовых UI-компонентов (например, `Button`, `Tag`, `ScrollProvider`), не содержащих бизнес-логику;
- Провайдеры глобальных состояний и контекстов (например, для управления скроллом или обработкой событий клавиатуры);
- Унифицированный генератор форм `FormBuilder`, позволяющий описывать структуру и поведение сложных форм через декларативную схему.

Применение данной библиотеки ускоряет процессы разработки и упрощает поддержку интерфейса, так как все ключевые решения собраны в централизованном модуле с единым API и консистентной документацией.

Компонент `FormBuilder`: концепция и API

Ключевым элементом библиотеки является компонент `FormBuilder`. Он реализует маршрутизацию данных и событий между декларативной схемой формы и её полями. Основные принципы работы:

а) Пользователь задаёт схему параметров формы, представляющую собой массив объектов с полем `type` и соответствующим набором `props`.

б) `FormBuilder` инициализирует внутреннее состояние формы и передаёт каждому полю текущие значения и функции обработки изменений.

в) При срабатывании события изменения значения поле уведомляет `FormBuilder`, который обновляет общую модель данных и вызывает коллбэк `onChange`.

Определение схемы формы Схема формы описывается функцией, возвращающей массив объектов. Каждый объект содержит:

– `type`: идентификатор типа элемента (например, `input_field`, `array_fields`);

– `props`: набор свойств, необходимых для рендеринга и обработки (имя поля, текст метки, дополнительные параметры).

Листинг 3: Пример описания схемы формы

```
export function inviteTeacherScheme(): FORM_BUILDER_SCHEMA {
  return [
  {
    type: 'input_field',
    props: {
      name: 'email',
      labelText: 'Email'
    }
  },
  {
    type: 'input_field',
    props: {
      type: 'select',
      name: 'department_id',
      ownerInputComponent: <DepartmentSelectField />
    }
  }
```

```
    ];
  }
```

Листинг 4: Использование `FormBuilder`

```
<FormBuilder
schema={inviteTeacherScheme()}
onChange={onChangeFormData}
/>
```

Компонент `FormBuilder` автоматически распределяет данные между полями и собирает итоговый объект формы, передавая его через `onChange`.

Типы элементов схемы и их поведение

Ниже приведены ключевые типы схем, поддерживаемые `FormBuilder`, и описание их функциональности.

INPUT_FIELD_SCHEMA Тип `input_field` отвечает за отображение и управление единичным полем ввода.

- `name`: ключ в итоговом объекте данных.
- `labelText`: отображаемая метка поля.
- `hintText` (опционально): текст подсказки под полем.
- `type` (опционально): уточняет тип поля (например, `select`, `datetime`).
- `ownerInputComponent` (опционально): пользовательский компонент, принимающий `value`, `onChange`, `isError`, `onBlur`.

Листинг 5: Пример INPUT_FIELD_SCHEMA

```
const schema: INPUT_FIELD_SCHEMA = {
  type: 'input_field',
  props: {
    name: 'username',
    labelText: 'Имя пользователя',
    hintText: 'Введите ваш логин'
  }
};
```

ARRAY_FIELDS_SCHEMA Тип `array_fields` предназначен для формирования массива однотипных входных полей. Все вложенные `input_field` будут собраны в массив.

- `name`: имя массива в итоговой модели.
- `children`: массив схем элементов, входящих в каждую запись.

Листинг 6: Пример ARRAY_FIELDS_SCHEMA

```
const schema: ARRAY_FIELDS_SCHEMA = [
{
  type: 'array_fields',
  props: {
    name: 'subjects',
    children: [
    {
      type: 'input_field',
      props: {
        name: 'subjectName',
        labelText: 'Название предмета'
      }
    }
    ]
  }
}
];
```

FORM_WRAPPER_SCHEMA Тип `form_wrapper` служит для группировки полей без сброса счётчика массивов. Внутренние поля записываются как вложенный объект.

- `name`: имя ключа в итоговом объекте.
- `children`: схема вложенных элементов.

Листинг 7: Пример FORM_WRAPPER_SCHEMA

```
const schema: FORM_WRAPPER_SCHEMA = [
{
  type: 'form_wrapper',
  props: {
    name: 'teacherInfo',
    children: [
    {
```

```
        type: 'input_field',
        props: {
          name: 'firstName',
          labelText: 'Имя'
        }
      },
      {
        type: 'input_field',
        props: {
          name: 'lastName',
          labelText: 'Фамилия'
        }
      }
      ]
    }
  }
  ];
```

BLOCK_WRAPPER_SCHEMA Тип `block_wrapper` позволяет визуально группировать элементы без изменения структуры данных: поля в нём обрабатываются как часть текущего массива или объекта.

Листинг 8: Пример BLOCK_WRAPPER_SCHEMA

```
  const schema: BLOCK_WRAPPER_SCHEMA = [
  {
    type: 'block_wrapper',
    props: {
      children: [
      {
        type: 'input_field',
        props: {
          name: 'code',
          labelText: 'Код'
        }
      }
      ]
    }
  }
  ];
```

REACT_NODE_SCHEMA Тип `react_node` предназначен для вставки произвольного React-элемента в произвольном месте формы.

Листинг 9: Пример REACT_NODE_SCHEMA

```
const schema: REACT_NODE_SCHEMA = [
{
  type: 'react_node',
  props: {
    node: <CustomSeparator />
  }
}
];
```

Преимущества и выводы

Использование декларативного `FormBuilder` существенно снижает сложность создания многоуровневых форм:

- Консистентность API при описании разных типов полей;
- Возможность единообразной валидации и управления ошибками;
- Лёгкость расширения — добавление новых типов полей или обёрток сводится к регистрации нового `type` и соответствующего рендерера;
- Повышение читаемости кода: структура формы полностью отражена в схеме без дублирования логики в компонентах.

3.3 Интеграция Auth.js

Для реализации авторизации в приложении использовалась библиотека Auth.js (ранее известная как NextAuth.js). Данный пакет рекомендован в официальной документации Next.js и обеспечивает гибкую работу с JWT-сессиями, однако требует дополнительной настройки для поддержки типизации, обновления токенов и обработки ошибок.

Типизация данных сессии

Для корректной работы TypeScript необходимо расширить интерфейсы, предоставляемые Auth.js. Пример декларации модулей:

Листинг 10: Типизация JWT и Session в Auth.js

```
import { AuthJWTErrorType } from "@/features/next-auth/types";

export declare module "next-auth/jwt" {
  interface JWT {
    access_token?: string;
```

```
    refresh_token?: string;
    token_type?: string;
    error?: AuthJWTErrorType;
  }
}

export declare module "next-auth" {
  interface Session {
    user: {
      access_token?: string;
      refresh_token?: string;
      token_type?: string;
      error?: AuthJWTErrorType;
    };
  }
  interface User {
    access_token?: string;
    refresh_token?: string;
    token_type?: string;
  }
}
```

Конфигурация провайдера и функция `authorize`

Основная логика входа пользователя реализуется в методе `authorize`, где учётные данные передаются на сервер, и в случае успешного ответа возвращается объект пользователя:

Листинг 11: Реализация функции authorize

```
async authorize(credentials, _req) {
  if (credentials.username && credentials.password) {
    const userData = {
      username: credentials.username as string,
      password: credentials.password as string,
    };

    const authResponse = await login(userData);
    if (authResponse?.success) {
      return authResponse.data;
    } else {
      throw new InvalidLoginError(
      `Backend server error: ${JSON.stringify(authResponse?.data)}`
      );
    }
  }
  throw new InvalidLoginError('Unknown authentication error');
```

```
}
```

JWT-callback и обновление токенов

Для передачи и обновления JWT в сессии используется callback `jwt`, принимающий параметр `trigger`, позволяющий определить сценарий обработки:

Листинг 12: JWT-callback с учётом trigger

```
async jwt({ token, trigger, user, session }): Promise<JWT> {
  if (trigger === 'signIn') {
    return { ...user, error: null };
  }
  if (token == null) {
    return { ...session?.user, error: 'another' } as JWT;
  }
  if (trigger === 'update') {
    return await refreshingProcess(token);
  }
  return { ...token, error: null };
},
```

Важным аспектом является недопущение состояния гонки при одновременном запросе обновления токена: для этого на клиенте сохраняется единственный промис обновления, который переиспользуется при множественных запросах:

Листинг 13: Механизм предотвращения race condition при рефреше токена

```
import { JWT } from "next-auth/jwt";

export type RefreshPromiseStateType = Promise<JWT | null> | null;
let tokenPromiseState: RefreshPromiseStateType = null;

export const setTokenPromiseState = (
promise: RefreshPromiseStateType
): void => {
  tokenPromiseState = promise;
};

export const getTokenPromiseState = (): RefreshPromiseStateType => {
  return tokenPromiseState;
};

let tokenState: JWT | null = null;
export const setTokenState = (newTokenState: JWT | null): void => {
  tokenState = newTokenState;
```

```
};
export const getTokenState = (): JWT | null => {
  return tokenState;
};

let timeoutState: NodeJS.Timeout | null = null;
```

Проверка сессии в Middleware

Для защиты маршрутов на уровне middleware необходимо предварительно проверять наличие валидной сессии и доступность токена:

Листинг 14: Пример проверки сессии в Middleware

```
const session = (await auth()) as Session | undefined;
const tokenPayload = getAccessTokenPayload(
session?.user?.access_token
);
if (
session?.user?.access_token &&
(isTokenAvailable(session.user.access_token) ||
isTokenAvailable(session.user.refresh_token)) &&
!session.user.error &&
tokenPayload
) {
  // доступ разрешён
} else {
  // перенаправление на страницу входа
}
```

Данный подход обеспечивает централизованную обработку авторизации и автоматическое обновление токенов без дублирования логики в разных частях приложения.

3.4 Модуль «Регистрация и вход»

В системе предусмотрены три сценария регистрации: для университета, студентов и преподавателей. В URL-параметрах `invite_id` передаются данные приглашения для последующей валидации и передачи на бэкенд.

Листинг 15: Выбор виджета регистрации

```
const getForm = () => {
    const type = getSearchParams(REGISTRATION_TYPE_PARAM_NAME) as
  ↪ RegistrationTypesType;
    const inviteId = getInviteId();
```

```
    switch (type) {
        case 'teacher':
            return <RegistrationTeacherWidget inviteId={inviteId} />;
        case 'student':
            return <RegistrationStudentWidget inviteId={inviteId} />;
        case 'university':
        default:
            return <RegistrationUniversityWidget />;
    }
};
```

Далее `invite_id` передаётся в соответствующий виджет, который запрашивает на сервере данные приглашения. В случае неверного или просроченного ID отображается сообщение об ошибке.

Листинг 16: RegistrationTeacherWidget

```
export function RegistrationTeacherWidget({ inviteId }:
  ↪ RegistrationPropsType) {
    const { initData, onSubmit, isError, setIsError, formDataRef } =
        useRegistrationStudentAndTeacher<typeof registerTeacher>({
            inviteId,
            registrationRequest: registerTeacher
        });

    if (initData === undefined) {
        return 'Loading';
    }

    if (initData === null) {
        // Неверное приглашение или оно истекло
        return 'Error';
    }

  ...
}
```

3.5 Административная панель университета

Административная панель университета разделена на несколько секций, каждая из которых отвечает за управление соответствующими данными: институтами, кафедрами, группами, преподавателями и студентами. Также предусмотрён модуль для формирования и отправки приглашений.

Страница списка с таблицей данных

На страницах списка отображается компонент `Table` с переданной конфигурацией `header` и `body`, а также обрабатываются события клика по строкам для навигации к деталям записи:

Листинг 17: Компонент страницы списка институтов

```
export function AdminInstitutePage() {
    const { data, body, onClickRow, header } = useListAdminInstitute();

    return (
        <div className={AdminListPageStyle.AdminListPage}>
            <div className={AdminListPageStyle.table}>
                <Table
                    header={header}
                    body={body}
                    onClickRow={onClickRow}
                />
            </div>
            <PaginationComponent totalCount={data.total_count} />
        </div>
    );
}
```

Страница просмотра/редактирования записи

На странице детализации отображаются текущие данные и предоставляется возможность их изменения через генератор форм и кнопку сохранения. Также реализована операция удаления записи:

Листинг 18: Компонент страницы детализации института

```
export function AdminInstituteDetailPage({ id }: { id: number }) {
    const { onUpdate, onDelete, onChangeFormData, data } =
  ↪ useDetailAdminInstitute(id);

    return (
        <div className={AdminDetailPageStyle.AdminDetailPage}>
            <div className={AdminDetailPageStyle.content}>
                <h3 className={AdminDetailPageStyle.header}>
                    Данные института
                </h3>
                <FormBuilder<InstitutePatchType>
                    formDataDefault={data}
                    clearForm={true}
                    onChange={onChangeFormData}
```

```
                    schema={instituteSchema()}
                />
                <Button
                    size="large"
                    hierarchy="primary"
                    text="Сохранить"
                    width="fill"
                    onClick={onUpdate}
                />
            </div>
            <div className={AdminDetailPageStyle.action}>
                <h3 className={AdminDetailPageStyle.header}>
                    Операции
                </h3>
                <Button
                    size="small"
                    hierarchy="primary"
                    warning={true}
                    iconLeft={<Trash01SVG />}
                    text="Удалить"
                    onClick={onDelete}
                    width="fill"
                />
            </div>
        </div>
    );
}
```

Страница создания новой записи

Компонент создания использует `FormBuilder` с соответствующей схемой и отправляет форму по событию нажатия кнопки:

Листинг 19: Компонент страницы создания института

```
export function AdminInstituteCreatePage() {
    const { onChangeFormData, onSend } = useCreateAdminInstitute();

    return (
        <div className={AdminDetailPageStyle.AdminDetailPage}>
            <div className={AdminDetailPageStyle.content}>
                <h1 className={AdminDetailPageStyle.header}>
                    Создание института
                </h1>
                <FormBuilder<InstitutePostType>
                    schema={instituteSchema()}
                    onChange={onChangeFormData}
                />
```

```
                <Button
                    onClick={onSend}
                    text="Создать"
                    size="large"
                    width="fill"
                />
            </div>
        </div>
    );
}
```

Модальные виджеты для создания приглашений

Для формирования приглашений используются два модальных компонента, управляемые состоянием `isActiveModalWindow`:

Листинг 20: Выбор модального окна приглашения

```
const getModalWindow = () => {
    switch (isActiveModalWindow) {
        case 'teacher':
            return <CreateTeacherInvites onClose={() =>
  ↪ setIsActiveModalWindow(undefined)} />;
        case 'students':
            return <CreateStudentsInvites onClose={() =>
  ↪ setIsActiveModalWindow(undefined)} />;
    }
};
```

Кнопки для вызова модальных окон:

Листинг 21: Кнопки вызова модальных приглашений

```
<ActionField
    title="Пригласить преподавателя"
    subtitle="Сформировать ссылку для регистрации преподавателя"
    onClick={() => setIsActiveModalWindow('teacher')}
/>
<ActionField
    title="Пригласить студентов"
    subtitle="Сформировать ссылку для регистрации студентов"
    onClick={() => setIsActiveModalWindow('students')}
/>
```

3.6 Модуль «Classrooms» (Виртуальные классы)

Модуль «Classrooms» обеспечивает полноценное управление виртуальными классами на клиенте, включая:

- загрузку и отображение списка классов с поддержкой пагинации,
- получение детальных данных класса и связанных заданий,
- создание нового класса и перенаправление на его страницу,
- интеграцию с AI-сервисом для анализа кода лабораторных работ.

Бизнес-логика модуля реализована ортогонально к UI через кастомные React-хуки. Ниже приведены их реализации без лишних импортов, с подробным описанием каждого шага.

Хук `useClassrooms`

Хук отвечает за загрузку списка классов и обработку параметров пагинации из строки URL. Он:

а) Читает `skip` и `limit` через слушатель URL-параметров;

б) Вызывает API-функцию `getClassroomsList` с этими параметрами;

в) Сохраняет результат запроса во внутреннее состояние `list` и возвращает его для UI;

г) Управляет состоянием загрузки и ошибок (не показано на этом уровне, так как хук возвращает только данные).

Листинг 22: Кастомный хук useClassrooms

```
export function useClassrooms() {
    const [list, setList] = useState<ClassroomListType | undefined | null>(
  ↪ undefined);
    const { getSearchParams } = useSearchParamsListener();

    useEffect(() => {
        const limit = Number(getSearchParams(LIMIT_SEARCH_PARAM_NAME) ||
  ↪ LIMIT_SEARCH_PARAM_DEFAULT);
        const skip = Number(getSearchParams(SKIP_SEARCH_PARAM_NAME) ||
  ↪ SKIP_SEARCH_PARAM_DEFAULT);
        getClassroomsList({ skip, limit }).then(setList);
    }, [
```

```
        getSearchParams(LIMIT_SEARCH_PARAM_NAME),
        getSearchParams(SKIP_SEARCH_PARAM_NAME)
    ]);

    return { list };
}
```

Этот хук позволяет любому компоненту, вызывающему его, получить актуальный список классов без дублирования логики запроса и парсинга параметров.

Хук `useClassroomDetail`

Хук инкапсулирует две независимые операции:

- получение детальной информации о классе по его идентификатору `id`;
- загрузку списка заданий, привязанных к данному классу, также с пагинацией.

Он инициирует оба запроса одновременно и возвращает два состояния: `classroom` и `list` (массив заданий).

Листинг 23: Кастомный хук useClassroomDetail

```
export function useClassroomDetail(id: number) {
    const [classroom, setClassroom] = useState<ClassroomDetailType |
  ↪ undefined | null>(undefined);
    const [list, setList] = useState<TaskListType | undefined | null>(
  ↪ undefined);
    const { getSearchParams } = useSearchParamsListener();

    useEffect(() => {
        const limit = Number(getSearchParams(LIMIT_SEARCH_PARAM_NAME) ||
  ↪ LIMIT_SEARCH_PARAM_DEFAULT);
        const skip = Number(getSearchParams(SKIP_SEARCH_PARAM_NAME) ||
  ↪ SKIP_SEARCH_PARAM_DEFAULT);

        getClassroomDetail(id).then(setClassroom);
        getTasksList({ classroomId: id, skip, limit }).then(setList);
    }, [
        id,
        getSearchParams(LIMIT_SEARCH_PARAM_NAME),
        getSearchParams(SKIP_SEARCH_PARAM_NAME)
    ]);

    return { classroom, list };
```

```
}
```

Таким образом, компонент, использующий этот хук, получает и данные по самому классу, и список заданий для отображения в UI, не заботясь о деталях запросов.

Хук `useClassroomCreate`

Хук управляет процессом создания нового класса:

а) Хранит ссылку на объект формы `formDataRef` через `useRef`;

б) Обрабатывает коллбэк `onChangeFormData`, обновляя `formDataRef.current`;

в) При вызове `onSubmit` отправляет текущие данные на сервер и, в случае успеха, перенаправляет пользователя на страницу нового класса.

Листинг 24: Кастомный хук useClassroomCreate

```
export function useClassroomCreate() {
    const router = useRouter();
    const formDataRef = useRef<ClassroomPostType>();

    const onChangeFormData = (newFormData: ClassroomPostType) => {
        formDataRef.current = newFormData;
    };

    const onSubmit = async () => {
        if (!formDataRef.current) return;
        const response = await postClassroom(formDataRef.current);
        if (response !== null) {
            router.replace(ROUTES_CONFIG.CLASSROOMS_DETAIL_SLUG_PAGE +
  ↪ response);
        }
    };

    return { onChangeFormData, onSubmit };
}
```

Хук минимизирует код в компонентах: весь процесс подготовки, валидации и навигации после создания вынесен внутрь.

Хук `useLabAnalysis`

Хук для интеграции AI-анализа кода лабораторных работ:

– Хранит объект `formData` и предоставляет `setFormData` для передачи данных (код или файл) из компонента;

– При вызове метода `analyze` отправляет `formData` на API (функция `postAnalyzeCode`);

– Сохраняет результат в `result` и ошибку в `error`.

Листинг 25: Кастомный хук useLabAnalysis

```
export function useLabAnalysis() {
    const [formData, setFormData] = useState<FormData | null>(null);
    const [result, setResult] = useState<any>(null);
    const [error, setError] = useState<string | null>(null);

    const analyze = async () => {
        if (!formData) return;
        try {
            const res = await postAnalyzeCode(formData);
            if (res.ok) setResult(res.data);
            else setError(res.error_message || 'Ошибка анализа');
        } catch {
            setError('Сетевая ошибка анализа');
        }
    };

    return { formData, setFormData, result, error, analyze };
}
```

3.7 Модуль «Chats» (Чаты)

Модуль «Chats» предназначен для организации обмена сообщениями в режиме реального времени между участниками системы (преподавателями и студентами). Ключевой технологией является библиотека Socket.IO, обеспечивающая надёжное WebSocket-соединение с возможностью автоматического восстановления. Вся бизнес-логика работы с соединением, событиями и состоянием сообщений инкапсулирована в отдельных функциях и хуках — это позволяет UI-компонентам оставаться «тонкими» и фокусироваться только на отображении.

В этом подразделе приведены основные элементы модуля «Chats»:

– хук `useSocket` для управления жизненным циклом WebSocket-соединения;

– компаратор `messagesComp` и функция `mergeMessages` для аккуратного упорядочивания и объединения потоков сообщений;
– функция `addMessages` для ввода свежих и локально отправленных сообщений в общее состояние;
– функция `sendMessage`, реализующая стратегию optimistic UI и синхронизацию с сервером;
– компонент `ScrollProvider` для двусторонней (вперёд и назад) пагинации сообщений.

Хук `useSocket`

Хук `useSocket` реализует следующие задачи:

а) Аутентификация через JWT: периодическое обновление токена доступа и передача его в заголовках при установке соединения;
б) Установка соединения с сервером по Socket.IO, включая указание URL, пользовательских заголовков и пути для WebSocket-подключения;
в) Подписка на базовые события `connect`, `disconnect`, `connect_error` для управления флагом `isConnected`;
г) Динамическая регистрация пользовательских обработчиков событий через параметр `onEvents`;
д) Предоставление методов `emit`, `disconnect`, `reconnect`, `connect` для внешнего управления соединением.

Процесс работы хука выглядит следующим образом:

– При монтировании компонента запускается эффект для получения сессии пользователя посредством функции `getClientSession`. Результат (JWT-токен) сохраняется в локальном состоянии и автоматически обновляется через заданный таймаут `TOKEN_UPDATE_TIMEOUT`. Это позволяет избежать простоя соединения из-за истёкшего токена и гарантирует, что при переподключении будет использован свежий токен авторизации.
– Второй эффект отвечает за само подключение к Socket.IO. Он создаёт новый экземпляр `io(url, options)`, где в

`options.extraHeaders` передаётся поле `Authorization`. Затем регистрируются «родные» события соединения и пользовательские события, переданные через проп `onEvents`. После этого ссылка на сокет сохраняется в рефе `socketRef`. Возврат функции очистки гарантирует, что при размонтировании или изменении зависимостей соединение будет корректно разорвано.

– Метод `emit` инкапсулирует вызов `socket.emit`, проверяя предварительно существование объекта сокета. Методы `disconnect`, `reconnect` и `connect` позволяют при необходимости вручную управлять соединением, например, при смене URL или после обработки ошибки.

Листинг 26: Кастомный хук useSocket

```
export function useSocket({ url, headers, onEvents }) {
    const socketRef = useRef(null);
    const [isConnected, setIsConnected] = useState(false);
    const [token, setToken] = useState(null);

    // 1. Обновление токена доступа с фиксированным интервалом
    useEffect(() => {
        const updateAuth = () => getClientSession().then(setToken);
        updateAuth();
        const timer = setInterval(updateAuth, TOKEN_UPDATE_TIMEOUT);
        return () => clearInterval(timer);
    }, []);

    // 2. Установка и подписка на WebSocket
    useEffect(() => {
        const socket = io(url, { extraHeaders: { Authorization: token?.
  ↪ access_token }, path: '/ws/socket.io' });
        socket.on('connect',    () => setIsConnected(true));
        socket.on('disconnect', () => setIsConnected(false));
        socket.on('connect_error', () => setIsConnected(false));
        // Регистрируем пользовательские события
        Object.entries(onEvents || {}).forEach(([event, handler]) => {
            socket.on(event, handler);
        });
        socketRef.current = socket;
        return () => socket.disconnect();
    }, [url, token, JSON.stringify(headers), JSON.stringify(onEvents)]);

    // 3. Эмиттер и ручное управление
```

```
    const emit      = (...args) => socketRef.current?.emit(...args);
    const disconnect = ()      => socketRef.current?.disconnect();
    const reconnect  = ()      => { socketRef.current?.disconnect(); socket.
  ↪ connect(); };
    const connect    = ()      => socketRef.current?.connect();

    return { emit, disconnect, reconnect, connect, isConnected };
}
```

В результате хук `useSocket` становится универсальным инструментом: он подходит для любых namespaces и событий, достаточно определить схему типов для `OnEvents` и `EmitEvents`.

Корректное объединение потоков сообщений

Передача и получение сообщений в чатах осуществляется в виде потоков, которые могут содержать старые и новые элементы. Для упорядочивания и исключения дубликатов разработаны:

- `messagesComp`: компаратор, сравнивающий два сообщения по серверному `id` или по локальному `local_id`, а затем по полю `created_at`;
- `mergeMessages`: редуктор, объединяющий два списка `base` и `insert` в хронологическом порядке, сохраняя уникальность.

Компаратор сначала проверяет полное совпадение идентификаторов — это позволяет найти случаи, когда локально отправленное сообщение (ещё без `id` от сервера) соответствует уже подтверждённому сервером сообщению по `local_id`. Если идентификаторы различны, они сравниваются по временной метке `created_at`. Такая стратегия обеспечивает корректную сортировку и «слияние» потоков.

Листинг 27: Компаратор сообщений messagesComp

```
const messagesComp = (a, b) => {
    // 1) Идентификаторы совпадают → одинаковое сообщение
    if ((a.id && b.id && a.id === b.id) ||
        (a.local_id && b.local_id && a.local_id === b.local_id)) {
        return 0; // EQUAL
    }
    // 2) Сравнение по дате создания
    return new Date(a.created_at) < new Date(b.created_at) ? -1 : 1;
};
```

Листинг 28: Функция слияния mergeMessages

```
export function mergeMessages({ base, insert, comp }) {
    const result = [];
    let i = 0, j = 0;
    // Идём по двум спискам параллельно, добавляя меньший элемент
    while (i < base.length && j < insert.length) {
        if (comp(base[i], insert[j]) <= 0) result.push(base[i++]);
        else result.push(insert[j++]);
    }
    // Добавляем остатки
    return result.concat(base.slice(i)).concat(insert.slice(j));
}
```

Добавление сообщений и оптимистичный UI

Для плавного UX при отправке сообщений используется механизм optimistic UI:

а) При отправке формируется временное сообщение `tempMsg` с полем `local_id` и статусом `'sending'`;

б) Это сообщение сразу добавляется в общий поток через `addMessages`;

в) После подтверждения от сервера (приём события `'message_sent'`) локальное сообщение заменяется на настоящий объект с серверным `id` благодаря функции слияния.

Листинг 29: Функция addMessages

```
function addMessages(newMsgs) {
    setMessages(prev => mergeMessages({
        base: prev || [],
        insert: Array.isArray(newMsgs) ? newMsgs : [newMsgs],
        comp: messagesComp
    }));
}
```

Листинг 30: Отправка сообщения sendMessage

```
function sendMessage(message) {
    const localId = generateLocalId();
    const tempMsg = { ...message, local_id: localId, status: 'sending',
  ↪ created_at: new Date().toISOString() };
    // 1) Мгновенно отображаем в UI
    addMessages(tempMsg);
    // 2) Отправляем на сервер, сервер вернёт событие о доставке
```

```
    emit('send_message', message);
}
```

Такой подход позволяет пользователю видеть своё сообщение немедленно, даже если сеть медленная, и обеспечивает плавную замену локального объекта на серверный при получении подтверждения.

Двусторонняя пагинация через ScrollProvider

Для загрузки как старых, так и новых сообщений используется компонент `ScrollProvider`. Он вызывает соответствующие колбэки при достижении верха или низа скролла, что позволяет:

- при прокрутке вверх дозагружать более ранние сообщения;
- при прокрутке вниз обновлять список свежими сообщениями;
- сохранять позицию скролла при подгрузке исторических или новых данных.

Листинг 31: Использование ScrollProvider

```
<ScrollProvider onScrollTop={loadOlder} onScrollBottom={loadNewer} accuracy
  ↪ ={200}>
    {messages.map(m => <ChatMessageItem key={m.id || m.local_id} {...m} />)
  ↪ }
</ScrollProvider>
```

Комбинация хука `useSocket`, функций `mergeMessages`/`addMessages` и компонента `ScrollProvider` создаёт единый модуль чатов, отвечающий требованиям надёжности, масштабируемости и приятного UX.

ЗАКЛЮЧЕНИЕ

В рамках проведённой дипломной работы была спроектирована и реализована клиентская часть веб-приложения для управления образовательным процессом в университете. Основная цель заключалась в создании масштабируемой и легко расширяемой архитектуры интерфейса, обеспечивающей высокую консистентность, читаемость и удобство поддержки. С этой целью были выполнены следующие ключевые мероприятия:

– Исследован и успешно применён модульный подход, позволивший чётко разделить проект на уровни ответственности и упростить навигацию по коду.

– Разработана собственная библиотека пользовательского интерфейса, включающая набор визуальных компонентов и механизм автоматической генерации форм на основе декларативных описаний. Эта библиотека обеспечила единообразие стилистики и сократила время на создание новых интерфейсных элементов.

– Внедрена система аутентификации и авторизации с поддержкой долгосрочных сессий, способная автоматически обновлять токены доступа без вмешательства пользователя и обеспечивать надёжную защиту маршрутов с учётом разных типов пользователей.

– Реализован универсальный механизм регистрации для различных ролей участников (администрации университета, преподавателей и студентов), включающий проверку приглашений и гибкую обработку ошибок серверной части.

– Создана административная панель с возможностями просмотра, создания, редактирования и удаления данных университета, включая управление структурой институтов, кафедр и учебных групп. Интерфейс обеспечил интуитивный доступ к основным операциям благодаря адаптированным формам и табличным представлениям.

– Построен модуль виртуальных классов, обеспечивающий создание и организацию учебных групп, назначение заданий и сбор обратной связи. Интеграция автоматизированного анализа решений студентов с использованием AI-технологий позволила существенно повысить качество проверки кода лабораторных работ.

– Разработана система мгновенного обмена сообщениями в реальном времени, включающая надёжное WebSocket-соединение, устойчивое к временным сбоям сети, а также удобные механизмы загрузки истории переписки в обе стороны и оптимистичного обновления интерфейса при отправке новых сообщений.

Проведённая работа подтверждает достижение поставленных задач: предложенные архитектурные решения обеспечили гибкость и расширяемость, а использование современных технологий и инструментов ускорило реализацию функционала и повысило устойчивость системы.

С практической точки зрения, клиентское приложение демонстрирует следующие преимущества:

– Повышенная скорость разработки за счёт переиспользования компонентов и генерации форм;

– Улучшенная безопасность благодаря продуманной схеме управления сессиями и авторизацией;

– Удобство сопровождения за счёт модульности и чёткого разделения ответственности;

– Расширяемость и поддержка новых сценариев благодаря гибкому подходу к конфигурации интерфейса.

В дальнейшем целесообразно рассмотреть следующие направления развития:

– Расширение возможностей автоматизированного анализа решений с учётом различных языков программирования и более глубокой семантической проверки;

– Добавление функционала офлайн-режима с последующей синхронизацией изменений;

– Разработку мобильных клиентских приложений для повышения доступности и удобства работы на мобильных устройствах;
– Внедрение системы аналитики и мониторинга пользовательской активности для оптимизации интерфейса и оценки эффективности процессов обучения.

Подведённые итоги свидетельствуют о высокой эффективности предложенного подхода и открывают перспективы дальнейших исследований и развития системы.

СПИСОК ЛИТЕРАТУРЫ

4 Список литературы

5 Приложения