

ОПРЕДЕЛЕНИЯ, СОКРАЩЕНИЯ И ОБОЗНАЧЕНИЯ

- CSR (Client-Side Rendering) — отрисовка страницы на стороне клиента после загрузки минимального каркаса с сервера.
- SSR (Server-Side Rendering) — рендеринг веб-страниц на стороне сервера перед отправкой клиенту.
- SSG (Static Site Generation) — генерация статических HTML-страниц на этапе сборки приложения.
- JWT (JSON Web Token) — формат токена для безопасной передачи информации между сторонами в виде JSON-объекта.
- API (Application Programming Interface) — программный интерфейс, обеспечивающий взаимодействие между компонентами программного обеспечения.
- UI (User Interface) — пользовательский интерфейс.
- Socket.IO — библиотека для организации двусторонних WebSocket-соединений между клиентом и сервером с автоматическим фоллбеком.
- CSRF (Cross-Site Request Forgery) — тип атаки, при которой злоумышленник вынуждает браузер пользователя выполнить нежелательный запрос от его имени; в веб-приложениях применяется защита с помощью специальных токенов.
- XSS (Cross-Site Scripting) — уязвимость, позволяющая внедрять скрипты стороннего происхождения на страницы; одной из мер защиты является хранение JWT в HTTP-only cookie.
- SSO (Single Sign-On) — единый вход, позволяющий пользователю аутентифицироваться один раз и использовать доступ ко множеству сервисов без повторной авторизации.
- HTTP (Hypertext Transfer Protocol) — протокол передачи гипертекстовых данных между клиентом и сервером; используется для REST-запросов.

					ОПРЕДЕЛЕНИЯ, СОКРАЩЕНИЯ И ОБОЗНАЧЕНИЯ							
Изм.	Лист	№ докум.	Подп.	Дата	Разработка front-end Web-приложения – учебной среды с чатами и AI-анализом кода лабораторных работ			Лит.	Лист	Листов		
Разраб.	Бондаренко С. В.									1	86	
Руковод.	Мельников А.Б.							БГТУ им. В.Г. Шухова, ПВ-212				
Консул.												
Н. контр.	Н.Кнтр. И.О.											
Зав. Каф.	Поляков В.М.											

- REST (Representational State Transfer) — архитектурный стиль взаимодействия с API через HTTP-методы (GET, POST и др.).
- JSON (JavaScript Object Notation) — текстовый формат обмена данными, широко используемый в REST-API и для передачи полезной нагрузки токенов.
- HTML (HyperText Markup Language) — язык разметки веб-страниц, создающий структуру документа.
- CSS (Cascading Style Sheets) — язык каскадных таблиц стилей для описания внешнего вида HTML-элементов.
- SCSS (Sassy CSS) — расширение CSS с поддержкой переменных, вложенности и других возможностей препроцессора.
- CRUD (Create, Read, Update, Delete) — базовые операции над данными, которые выполняются при создании, чтении, обновлении и удалении записей.
- CLI (Command Line Interface) — интерфейс командной строки, используемый для генерации проекта и выполнения скриптов (например, Angular CLI или Next.js CLI).
- MVP (Minimum Viable Product) — минимально жизнеспособный продукт, концепция, описывающая начальную версию продукта с базовым функционалом.
- Back-end (бэкэнд) — это серверная часть приложения, которая обрабатывает логику работы системы, управляет хранением данных и предоставляет интерфейсы (API) для взаимодействия с клиентской частью.
- Front-end (фронтэнд) — это часть приложения, отвечающая за пользовательский интерфейс и его логику.
- Hooks (хуки) — в контексте React это специальные функции, позволяющие функциональным компонентам использовать состояние, жизненный цикл и другие возможности React.
- Endpoint (эндпоинт) — это уникальный URL-адрес (или URI), который представляет собой точку входа для взаимодействия с API или веб-сервисом.

					ОПРЕДЕЛЕНИЯ, СОКРАЩЕНИЯ И ОБОЗНАЧЕНИЯ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		2

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	7
1 ОБЗОР И ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	9
1.1 Введение в предметную область	9
1.2 Анализ существующих образовательных платформ	10
1.2.1 Система управления обучением — Moodle.....	10
1.2.2 Платформа дистанционного обучения — Google Classroom.....	10
1.2.3 Платформа для онлайн-обучения — Microsoft Teams.....	10
1.2.4 Специализированные платформы для анализа кода	11
1.3 Проблемы существующих решений	11
1.4 Потребности образовательной среды.....	11
1.4.1 Доступность материалов и заданий	12
1.4.2 Автоматизация проверок и оценки	12
1.4.3 Удобная система заданий и общения.....	12
1.4.4 Интеграция всех процессов в одну систему	12
1.4.5 Вывод.....	12
1.5 Технологии разработки клиентской части приложения	13
1.5.1 Язык программирования — TypeScript	13
1.5.2 Библиотека — React	14
1.5.3 Веб-фреймворк — Angular	15
1.5.4 Прогрессивный веб-фреймворк — Vue.js	15
1.5.5 Сравнительный анализ фреймворков	16
1.6 Требования к функциональности приложения	22
1.6.1 Регистрация и структура университетов	22
1.6.2 Панель преподавателя	22
1.6.3 Панель студента	23
1.7 Диаграмма вариантов использования	23
1.8 Коммуникация и взаимодействие	23

					СОДЕРЖАНИЕ							
Изм.	Лист	№ докум.	Подп.	Дата	Разработка front-end Web-приложения – учебной среды с чатами и AI-анализом кода лабораторных работ			Лит.	Лист	Листов		
Разраб.	Бондаренко С. В.										3	86
Руковод.	Мельников А.Б.							БГТУ им. В.Г. Шухова, ПВ-212				
Консул.												
Н. контр.	Н.Кнтр. И.О.											
Зав. Каф.	Поляков В.М.											

1.9	Безопасность данных	24
1.9.1	Авторизация с использованием JWT и Auth.js	24
1.9.2	Роль Auth.js.....	25
1.9.3	Меры защиты.....	26
1.10	Система создания заданий с анализом кода на основе искусственного интеллекта	26
1.10.1	Общее описание модуля	26
1.10.2	Функциональные возможности.....	27
1.10.3	Анализ кода на основе искусственного интеллекта.....	27
1.11	Анализ кода на основе DeepSeek.....	27
1.11.1	Общее описание модуля DeepSeek	27
1.11.2	Функциональные возможности DeepSeek	28
1.11.3	Принцип работы DeepSeek	28
1.12	Тестирование с использованием Jest.....	29
1.13	Выводы.....	30
2	ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОЕКТА.....	31
2.1	Архитектура клиентской части системы	31
2.1.1	Общая структура архитектуры	31
2.1.2	Архитектурная диаграмма	31
2.1.3	Слои архитектуры Feature-Sliced Design	32
2.1.4	Концепция срезов (slices)	32
2.1.5	Горизонтальное деление на сегменты (segments).....	33
2.2	Проектирование интерфейсных подсистем и экранов.....	34
2.2.1	Выделение ключевых интерфейсных подсистем	35
2.2.2	Страницы и их структура	38
2.2.3	Компоненты и принципы их структурирования	41
2.2.4	Распределение логики по слоям архитектуры	41
2.2.5	UX-решения и пользовательские сценарии	42
2.3	Проектирование взаимодействия с сервером и WebSocket	43
2.3.1	Аутентификация и управление токенами.....	44
2.3.2	Унифицированная функция отправки запросов.....	45
2.3.3	Взаимодействие через WebSocket.....	45

2.4	Интеграция ИИ-модуля DeepSeek в архитектуру проекта	46
2.4.1	Контейнеризация DeepSeek.....	46
2.4.2	Преимущества контейнеризированного подхода	46
2.4.3	Взаимодействие клиентской части с DeepSeek	47
2.4.4	Вывод.....	47
2.5	Обеспечение безопасности клиентской части	48
2.5.1	Разграничения доступа с использованием <i>middleware</i>	48
2.5.2	Роль и защита при работе с форматируемым текстом	49
2.5.3	Вывод.....	49
2.6	Покрытие бизнес-логики юнит-тестами.....	50
3	ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	52
3.1	Архитектура по Feature-Sliced Design	52
3.1.1	Слой <i>shared</i>	52
3.1.2	Слой <i>entities</i>	53
3.1.3	Слои <i>features, widgets, pages</i>	53
3.1.4	Пример виджета RegistrationUniversity	54
3.1.5	Соглашения по именованию и структуре.....	54
3.2	Собственная UI-библиотека и генерация форм	55
3.2.1	Общая идея и мотивация	55
3.2.2	Компонент <i>FormBuilder</i> : концепция и API	55
3.2.3	Типы элементов схемы и их поведение.....	57
3.2.4	Преимущества и выводы	59
3.2.5	Работа с токеном JWT	60
3.3	Модуль «Регистрация и вход».....	63
3.3.1	Модуль формирования и отправки приглашений	66
3.3.2	Последовательная диаграмма работы модуля Classrooms	68
3.3.3	Последовательная диаграмма работы модуля Chats	70
3.4	Тестирование.....	73
3.4.1	Покрытие кода в приложении	73
3.4.2	Покрытие кода в отдельной библиотеке	74
3.4.3	Методы тестирования	74
	ЗАКЛЮЧЕНИЕ	76
	СПИСОК ЛИТЕРАТУРЫ	79

ПРИЛОЖЕНИЕ А 81

ПРИЛОЖЕНИЕ Б 86

					СОДЕРЖАНИЕ	Лист
						6
Изм.	Лист	№ докум.	Подп.	Дата		

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 приложения, которое бы объединяло образовательные инструменты в едином пространстве. Приложения, решающие 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 интегрированного образовательного Web–приложения, которое объединяет функции чатов, проведения занятий и автоматического анализа решений, используя возможности ИИ. Сейчас отсутствует единое решение, которое бы эффективно сочетало в себе эти ключевые аспекты: общения через чаты, создание заданий и автоматическую проверку решений с помощью ИИ. Современные системы, как правило, фрагментированы — отдельные модули 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 объединит 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 улучшение и упрощение опыта работы преподавателей и процесса обучения студентов путем разработки front-end части интегрированного образовательного Web-приложения, обеспечивающего эффективное взаимодействие, автоматизированную проверку решений и удоб-

					ВВЕДЕНИЕ			
Изм.	Лист	№ докум.	Подп.	Дата	Разработка front-end Web-приложения – учебной среды с чатами и AI-анализом кода лабораторных работ	Лит.	Лист	Листов
Разраб.		Бондаренко С. В.						
Руковод.		Мельников А.Б.					7	86
Консул.						БГТУ им. В.Г. Шухова, ПВ-212		
Н. контр.		Н.Кнтр. И.О.						
Зав. Каф.		Поляков В.М.						

ные средства коммуникации.

Для достижения поставленной цели необходимо решить следующие задачи:

- 1) Проанализировать предметную область и существующие системы, выявив их сильные и слабые стороны;
- 2) Определить архитектурные и технологические решения, подходящие для реализации front-end части Web-приложения;
- 3) Спроектировать пользовательский интерфейс, обеспечивающий интуитивное и удобное взаимодействие для преподавателей и студентов;
- 4) Разработать компоненты для управления учебными структурами (институт, кафедра, группа), заданиями и чатами;
- 5) Интегрировать средства для автоматизированной проверки решений студентов с применением ИИ;
- 6) Реализовать тестирование бизнес-логики Web-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;
- Во втором разделе описывается архитектура front-end части Web-приложения, структура пользовательского интерфейса, проектирование компонентов и их взаимодействие;
- В третьем разделе приводится описание реализации: структура кода, используемые технологии (Next.js, React, TypeScript, Redux, Auth.js), описание экранов и взаимодействий, примеры реализации различных компонентов Web-приложения;
- В заключении приводятся выводы по выполненной работе, оценивается эффективность разработанного интерфейса и функционала, а также определяются направления для дальнейшего развития и улучшения системы.

					ВВЕДЕНИЕ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		8

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 объёмов 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 значительно упростить организацию учебного процесса, улучшить взаимодействие между

					ОБЗОР И ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ			
Изм.	Лист	№ докум.	Подп.	Дата				
Разраб.		Бондаренко С. В.			Разработка front-end Web-приложения – учебной среды с чатами и AI-анализом кода лабораторных работ	Лит.	Лист	Листов
Руковод.		Мельников А.Б.					9	86
Консул.						БГТУ им. В.Г. Шухова, ПВ-212		
Н. контр.		Н.Кнтр. И.О.						
Зав. Каф.		Поляков В.М.						

преподавателями и студентами, а также повысить качество обучения за счёт автоматизации рутинных задач.

1.2 Анализ существующих образовательных платформ

Современные образовательные платформы, такие как Moodle [1], Google Classroom [2], Microsoft Teams для образования [3], 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 Система управления обучением — Moodle

Moodle является одной из самых популярных образовательных платформ, используемых во многих учебных заведениях [1]. 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 Платформа дистанционного обучения — Google Classroom

Google Classroom предлагает простоту в использовании и позволяет интегрировать различные Google сервисы [2]. 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 Платформа для онлайн-обучения — Microsoft Teams

Microsoft Teams, в отличие от Moodle и Google Classroom, активно используется для организации видеоконференций и групповых чатов [3]. Он позволяет преподавателям и студентам взаимодействовать в реальном времени, а

					ОБЗОР И ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		10

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 Специализированные платформы для анализа кода

Существуют специализированные платформы, такие как CodeSignal [4], Codility [5] и LeetCode [6], 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.

					ОБЗОР И ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		11

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.

1.4.3 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.

1.4.4 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.

1.4.5 Вывод

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

					ОБЗОР И ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		12

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.

1.5.1 Язык программирования — 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. Для получения большей информации смотрите документацию [7].

					ОБЗОР И ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		13

1.5.2 Библиотека — React

React — библиотека для построения пользовательских интерфейсов, разработанная Facebook (см. документацию [8]). Она широко используется в веб-разработке благодаря своей простоте, гибкости и высокой производительности. React обеспечивает декларативный 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.

Можно выделить следующие 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 включаются такие возможности, как маршрутизация, SSR и API.

					ОБЗОР И ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		14

1.5.3 Веб-фреймворк — Angular

Angular — полноценный MVC-фреймворк, предоставляющий комплексное решение для разработки enterprise-приложений [9]. 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 [10];
- 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 500 КБ в минимальной сборке);
- Жёсткие требования к архитектуре.

1.5.4 Прогрессивный веб-фреймворк — Vue.js

Vue.js — прогрессивный фреймворк, сочетающий подходы React и Angular [11]. Особенно эффективен для быстрого прототипирования и проектов средней сложности.

Основные характеристики:

					ОБЗОР И ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		15

- 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 (24 КБ в gzip).

Ограничения:

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

1.5.5 Сравнительный анализ фреймворков

На основе сравнения характеристик трёх популярных JavaScript-фреймворков (см. таблицу 1.1) можно сделать следующие выводы: React с Next.js демонстрирует высокую производительность и даёт гибкую архитектуру, а встроенная поддержка SSR/SSG и SEO-оптимизация делают этот стек идеальным выбором для масштабируемых образовательных приложений. Angular, с другой стороны, благодаря строгой типизации и продуманной структуре, подходит для крупных enterprise-систем, где важны единообразие и безопасность кода. Vue.js сочетает в себе простоту освоения и возможность расширения экосистемы, что отлично для быстрого запуска MVP и небольших команд.

					ОБЗОР И ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		16

Таблица 1.1 – 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)

Фреймворк SSR/SSG — Next.js

Фреймворк для React объединяет в себе серверный рендеринг и «гидратацию» клиентского кода, обеспечивая быструю и плавную работу приложений. Благодаря гидратации браузеру отправляется только необходимый на текущий момент минимальный объём JavaScript, а по мере навигации и взаимодействия фреймворк автоматически подгружает дополнительные скрипты, расширяя уже отрендеренный контент. Это снижает время первого отображения страницы и ускоряет переходы между разделами, что особенно важно для крупных проектов с большими бандлами.

В документации [12] выделяют несколько режимов серверного рендеринга:

- SSR (Server-Side Rendering) — классический серверный рендеринг, при котором HTML-страница полностью генерируется на сервере при каждом запросе и отправляется клиенту. Пользователь сразу видит готовый контент без ожидания выполнения JavaScript.
- SSG (Static Site Generation) — статическая генерация страниц на этапе билда проекта: все HTML-файлы готовятся заранее и хранятся на сервере, что позволяет отдавать их мгновенно и без лишних вычислений [13].
- ISR (Incremental Static Regeneration) — расширение SSG, дающее возможность автоматически пересобирать отдельные страницы через заданные интервалы времени, сохраняя преимущества статической отдачи и обновляя устаревший контент [13].

Такой многообразный набор инструментов делает Next.js крайне гибким:

					ОБЗОР И ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		17

разработчики могут подобрать оптимальный способ рендеринга для каждой страницы — от полностью статических лендингов до динамических разделов с актуальными данными. В результате ваше приложение получает высокую производительность, хорошую индексацию поисковиками и комфортный пользовательский опыт без избыточной передачи данных.

Библиотека WebSocket — Socket.IO

JavaScript-библиотека Socket.IO с открытым исходным кодом предназначена для реализации двустороннего взаимодействия между клиентом и сервером в режиме реального времени. В соответствии с документацией [14] 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.

Недостатки:

- Необходимость разработки и поддержки собственной серверной инфраструктуры;

					ОБЗОР И ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		18

- Повышенная сложность при масштабировании без использования внешних инструментов (Redis, Kubernetes);
- Отсутствие встроенной панели мониторинга или аналитики соединений.

Сервис обмена событиями в реальном времени — Pusher

Pusher — это облачная платформа, предоставляющая API и SDK для реализации push-уведомлений и двусторонней передачи данных в реальном времени [15]. 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.

Двухнаправленный обмен сообщениями — WebSocket в JavaScript

JavaScript предоставляет встроенный класс *WebSocket* для организации двустороннего обмена данными между клиентом и сервером [16]. Этот класс реализует базовую функциональность протокола WebSocket, предоставляя

					ОБЗОР И ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		19

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 объёма дополнительной логики. Это делает его менее удобным для быстрого внедрения в проект, особенно в случае масштабируемых приложений.

С учётом 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. В таблице 1.2 представлено 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 все необходимые механизмы для реализации масштабируемой и надёжной чат-системы, при этом позволяя оптимизировать производительность без привлечения сторонних сервисов.

					ОБЗОР И ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		20

Таблица 1.2 – 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	Через Redis и кластеризацию	Встроенная на уровне платформы
Простота настройки	Средняя (требует сервера)	Высокая (SDK)
Затраты на	Бесплатно	Платная модель
Интеграция с Node.js	Нативная	Через SDK
Надёжность	Высокая	Высокая
Кастомизация	Да	Нет

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.

Библиотека аутентификации — 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 (см. [17]), а также реализовывать собственную аутентификацию с использованием базы данных. Auth.js обеспечивает надёжную 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.

Библиотека управления состоянием приложения — Redux

Redux — это библиотека для управления состоянием в приложениях, основанных на React [18]. 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 простым.

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.

1.6.1 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.

1.6.2 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;

					ОБЗОР И ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		22

- просмотр и анализ результатов выполнения заданий;
- предоставление обратной связи студентам.

1.6.3 Панель студента

Студенты, присоединённые к определённым 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 А.1 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 искусственного интеллекта;
- 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.

1.8 Коммуникация и взаимодействие

Ключевым элементом платформы является система обмена сообщениями, включающая:

1) Групповые чаты:

- Общение участников учебной группы в режиме реального времени;

					ОБЗОР И ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		23

- Передача файлов (форматы: .jpg, .png, .mp4, .pdf, .zip и др.);
- Привязка к группам.

2) 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

В данном разделе приводится обзор ключевых принципов и практик, обеспечивающих конфиденциальность, целостность и доступность данных в приложении.

1.9.1 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:

1) Аутентификация пользователя:

- Пользователь выполняет вход с помощью логина/пароля или через одного из OAuth-провайдеров (например, Google);

					ОБЗОР И ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		24

- Auth.js инициирует процесс аутентификации и, при успешной проверке, получает JWT.

2) 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.

3) 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.

1.9.2 Роль Auth.js

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

1) Инкапсуляция логики:

- Обработывает все основные сценарии авторизации (вход, выход, обновление токена);
- Управляет хранением и безопасной передачей токенов.

2) Поддержка современных стандартов:

- Генерирует CSRF-токены;
- Поддерживает стратегию Single Sign-On (SSO).

					ОБЗОР И ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		25

1.9.3 Меры защиты

Для повышения безопасности механизма авторизации реализованы дополнительные меры:

- 1) JWT: Access-токен действует ограниченное время (например, 5 минут), а refresh-токен — 30 дней;
- 2) 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 анализом кода на основе искусственного интеллекта

В данном разделе описаны основные функциональные возможности анализа кода на основе искусственного интеллекта для преподавателей и студентов.

1.10.1 Общее описание модуля

Разработанная система позволяет преподавателям создавать виртуальные классрумы, выдавать задания и автоматически анализировать решения студентов с использованием инструментов искусственного интеллекта. 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 анализе кода на основе искусственного интеллекта;
- Просмотр статистики и аналитики преподавателем.

					ОБЗОР И ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		26

1.10.2 Функциональные возможности

Для преподавателей представлены следующие функциональные возможности:

- Создание и управление классами;
- Назначение заданий;
- Просмотр отчётов искусственного интеллекта по каждому студенту;
- Сводная статистика по группе.

У студентов возможностей меньше, а именно:

- Просмотр активных заданий и дедлайнов;
- Загрузка решения.

1.10.3 Анализ кода на основе искусственного интеллекта

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 Анализ кода на основе DeepSeek

1.11.1 Общее описание модуля DeepSeek

DeepSeek — это облачная платформа для глубинного анализа исходного кода, основанная на передовых архитектурах трансформеров и нейронных сетей [19]. Ее учебный модуль автоматизирует ручную проверку студенческих работ, минимизирует субъективность оценивания и предоставляет преподавателям детальную, содержательную обратную связь.

					ОБЗОР И ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		27

- 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;
- 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: напрямую вызвать проверку кода искусственным интеллектом нельзя — проверка доступна только при отправке решения студентом.

1.11.2 Функциональные возможности DeepSeek

Общий функционал разделён на две части: для преподавателей и для студентов. Для преподаватель:

- 1) Автоматически сгенерированные отчёты, где каждая найденная проблема снабжена подробным описанием и примером исправления;
- 2) Механизм гибкой настройки критериев оценки — преподаватель может добавлять свои правила проверки в зависимости от характера задания.

Студент, в свою очередь, имеет следующие возможности:

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

1.11.3 Принцип работы DeepSeek

- 1) Лексический и синтаксический разбор: исходный код разбивается на токены, строится абстрактное синтаксическое дерево, на основе которого проводится первичный анализ.
- 2) Семантический анализ: алгоритмы трансформеров сопоставляют логику решения с огромной базой примеров, выявляя нетривиальные отклонения от оптимальной структуры.

					ОБЗОР И ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		28

- 3) Расчёт метрик: рассчитываются показатели цикломатической сложности, глубины вложенности, соответствия coding standard и других параметров качества.
- 4) Антиплагиатный компонент: сравнение embedding-векторов загруженного решения с репозиторием эталонных и ранее проверенных работ для определения степени оригинальности.
- 5) 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 (см. документацию [20]). 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.

					ОБЗОР И ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		29

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.

					ОБЗОР И ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		30

2 ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОЕКТА

2.1 Архитектура клиентской части системы

Клиентская часть разрабатываемой системы реализована в виде одностороннего 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) [21] — 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

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

					ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОЕКТА			
Изм.	Лист	№ докум.	Подп.	Дата				
Разраб.	Бондаренко С. В.				Разработка front-end Web-приложения – учебной среды с чатами и AI-анализом кода лабораторных работ	Лит.	Лист	Листов
Руковод.	Мельников А.Б.						31	86
Консул.						БГТУ им. В.Г. Шухова, ПВ-212		
Н. контр.	Н.Кнтр. И.О.							
Зав. Каф.	Поляков В.М.							

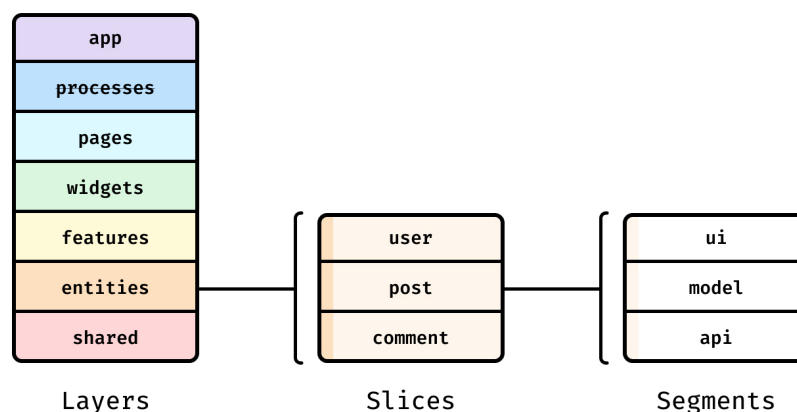


Рисунок 2.1 – Схема архитектуры клиентской части (FSD + Next.js)

принцип инверсии зависимостей и минимизировать связанность между модулями.

2.1.3 Слои архитектуры Feature-Sliced Design

В таблице 2.1 представлены слои архитектуры Feature-Sliced Design, сгруппированные по уровню абстракции и ответственности.

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.

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.

Таблица 2.1 – Слои архитектуры Feature-Sliced Design

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

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 ти-

пов запросов и маппинг ответов;

- *lib* — вспомогательные функции и библиотеки, используемые в пределах данного среза;
- *config* — конфигурационные файлы и переключатели функциональности.

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*.

					ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОЕКТА	Лист
Изм.	Лист	№ докум.	Подп.	Дата		34

2.2.1 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.

1) 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.

					ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОЕКТА	Лист
						35
Изм.	Лист	№ докум.	Подп.	Дата		

2) Подсистема управления университетом

Реализует административную логику, связанную с конфигурацией организационной структуры образовательного учреждения (структура компонентов показана на Рис. А.4).

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.

3) Подсистема работы с заданиями и отправкой решений

Данная подсистема предназначена для организации учебной деятельности (структура компонентов представлена на Рис. А.5).

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.

					ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОЕКТА	Лист
						36
Изм.	Лист	№ докум.	Подп.	Дата		

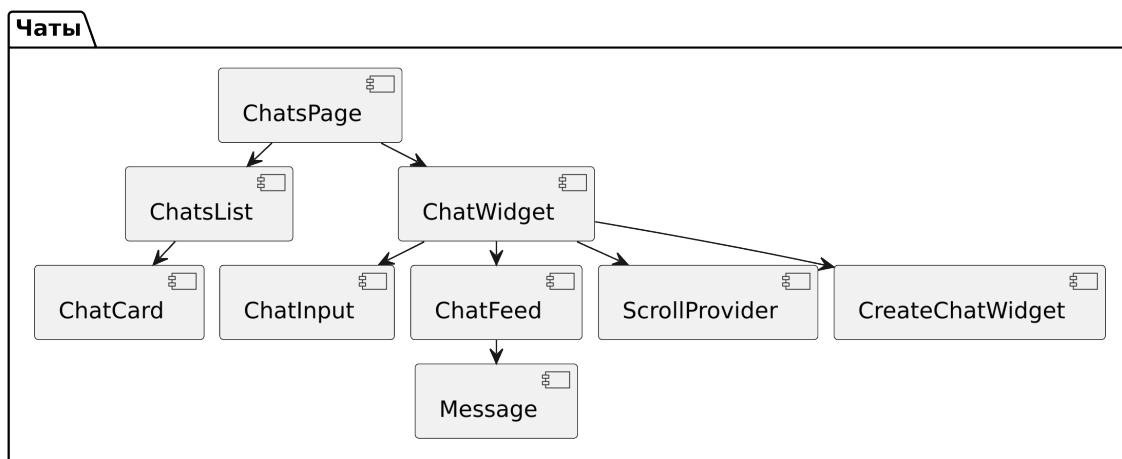


Рисунок 2.2 – Диаграмма компонентов системы чатов

4) Подсистема обмена сообщениями (чаты)

В рамках образовательного процесса большое значение имеет возможность коммуникации (структура компонентов отображена на Рис. 2.2).

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.

5) Подсистема анализа решений на основе искусственного интеллекта

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

Подсистема предназначена для получения и визуализации результатов анализа на основе искусственного интеллекта, включающих:

- Оценку корректности кода;
- Проверку на соответствие заданию;
- 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 с анализом на основе искусственного интеллекта, а система чатов — с механизмами авторизации и маршрутизации. Такое проектирование обеспечивает гибкость, надёжность и чёткую масштабируемость клиентской архитектуры.

2.2.2 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

					ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОЕКТА	Лист
Изм.	Лист	№ докум.	Подп.	Дата		38

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 анализа на основе искусственного интеллекта.

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);

					ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОЕКТА	Лист
						39
Изм.	Лист	№ докум.	Подп.	Дата		

- Просмотр структуры учреждения в табличной форме;
- Генерацию и отправку 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.

					ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОЕКТА	Лист
Изм.	Лист	№ докум.	Подп.	Дата		40

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

2.2.3 Компоненты и принципы их структурирования

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

2.2.4 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

					ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОЕКТА	Лист
Изм.	Лист	№ докум.	Подп.	Дата		41

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.

2.2.5 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, включаю-

					ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОЕКТА	Лист
Изм.	Лист	№ докум.	Подп.	Дата		42

щий 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

					ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОЕКТА	Лист
Изм.	Лист	№ докум.	Подп.	Дата		43

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.

2.3.1 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.

1) Аутентификация пользователя:

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

2) 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.

3) Доступ к защищённым ресурсам:

- при обращении к API front-end автоматически прикрепляет токен к запросу;
- при недействительном или истёкшем токене *Auth.js* обновляет его через refresh-токен, если он присутствует.

					ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОЕКТА	Лист
Изм.	Лист	№ докум.	Подп.	Дата		44

2.3.2 Унифицированная функция отправки запросов

Для стандартизации сетевого взаимодействия была разработана функция *sendRequest*, инкапсулирующая логику подготовки и отправки HTTP-запросов:

- преобразование данных в JSON через *JSON.stringify*;
- добавление заголовков, включая *Authorization: Bearer*;
- обработка ошибок и повторная попытка после обновления токена;
- поддержка различных HTTP-методов.

2.3.3 Взаимодействие через WebSocket

Для реализации обмена сообщениями в реальном времени используется библиотека *Socket.IO*, обеспечивающая:

- автоматическое переподключение при обрыве соединения;
- передачу структурированных событий с именами и аргументами;
- интеграцию с middleware для авторизации;
- работу с пространствами имён и комнатами;
- fallback-транспорты при недоступности WebSocket.

На стороне клиента реализован хук *useSocket*, который:

- инициализирует соединение с сервером;
- отправляет и получает события с типизированными данными;
- подписывается и отписывается от каналов;
- управляет жизненным циклом подключения и логирует события;
- обрабатывает ошибки соединения.

При установлении WebSocket-соединения клиент передаёт access-token в параметрах, сервер проверяет его и активирует соединение. В случае истечения срока действия токена:

- 1) инициируется его обновление;
- 2) текущее соединение закрывается;
- 3) создаётся новое соединение с обновлённым токеном.

					ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОЕКТА	Лист
Изм.	Лист	№ докум.	Подп.	Дата		45

Вывод

Реализованные механизмы автоматического обновления токенов, единая функция отправки запросов и продуманная интеграция WebSocket через *Socket.IO* обеспечивают безопасность, надёжность и масштабируемость взаимодействия клиентской части с сервером в режиме реального времени.

2.4 Интеграция ИИ-модуля DeepSeek в архитектуру проекта

Для обеспечения эффективной работы ИИ-компонента в образовательной платформе реализована модульная архитектура с чётким разделением ответственности. Последующие подразделы детализируют ключевые аспекты интеграции: стратегию контейнеризации для изоляции сервиса, механизмы взаимодействия с клиентской частью и системные преимущества выбранного подхода. Основное внимание уделено сохранению прозрачности работы ИИ для конечных пользователей при обеспечении гибкости разработки и эксплуатации.

2.4.1 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.

2.4.2 Преимущества контейнеризированного подхода

Контейнеризация DeepSeek даёт следующие ключевые плюсы:

- Изоляция нагрузки: анализ кода выполняется в отдельном окружении, не влияя на отзывчивость интерфейса;

					ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОЕКТА	Лист
Изм.	Лист	№ докум.	Подп.	Дата		46

- Горизонтальное масштабирование: при большом числе запросов можно запускать несколько экземпляров контейнера;
- Упрощённое сопровождение: обновление ИИ-компонента сводится к выпуску нового Docker-образа без правок во фронтенде;
- Гибкость развёртывания: контейнеры можно запускать локально и в облаке с одинаковой конфигурацией.

2.4.3 Взаимодействие клиентской части с DeepSeek

Клиентская часть приложения, реализованная на React и Next.js, отправляет HTTP-запросы к back-end части Web-приложения при прикреплении решения задания студентом, далее back-end автоматически начинает проверку решения при помощи анализа на основе искусственного интеллекта. Результат проверки приходит пользователю при запросе на получения детальной информации по задаче, выполненной студентом.

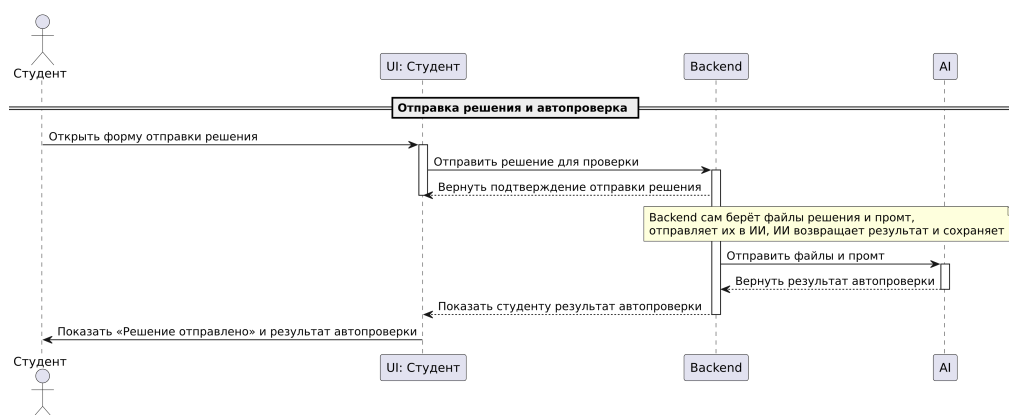


Рисунок 2.3 – Схема взаимодействия клиентской части (React/Next.js) с модулем DeepSeek

На рисунке 2.3 представлена схема взаимодействия клиентской части с модулем DeepSeek.

2.4.4 Вывод

Контейнеризация DeepSeek обеспечивает полную независимость остальных компонентов приложения от ИИ-модуля, позволяя развёртывать и обновлять его без влияния на другие сервисы. Взаимодействие через API бэкенда создаёт своего рода «чёрный ящик» для клиента, что упрощает инкапсуляцию

					ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОЕКТА	Лист
Изм.	Лист	№ докум.	Подп.	Дата		47

логики и даёт гибкость в распределении и масштабировании нагрузки на контейнер с ИИ.

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.

2.5.1 Разграничения доступа с использованием *middleware*

Одним из ключевых компонентов обеспечения безопасности клиентской части является система контроля доступа на основе промежуточного слоя — *middleware* [22]. 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 полезную нагрузку (payload) — уникальный идентификатор, срок действия сессии и 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

					ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОЕКТА	Лист
Изм.	Лист	№ докум.	Подп.	Дата		48

предоставляется, и страница загружается с соответствующим контентом.

Таким образом, механизм *middleware* обеспечивает надёжную фильтрацию обращений к различным частям интерфейса, предотвращая несанкционированный доступ и соблюдая политику разграничения прав.

2.5.2 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 библиотека *tiptap* — расширяемый редактор форматированного текста на основе *ProseMirror*. Одним из ключевых преимуществ *tiptap* является контроль над тем, какие HTML-теги и атрибуты допускаются к отображению. Таким образом, даже если пользователь попытается вставить опасный код (например, `<script>` или обработчик событий), редактор удалит такие элементы на этапе парсинга.

Технически это реализуется следующим образом:

- при вводе содержимого редактор не сохраняет «сырые» HTML-строки, а формирует безопасное представление согласно заданным схемам;
- 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 тегов и атрибутов.

Таким образом, даже при наличии активной формы редактирования форматируемого текста пользовательская среда остаётся защищённой от внедрения опасного контента.

2.5.3 Вывод

Комплекс реализованных решений позволяет эффективно защитить клиентскую часть приложения как от внешнего вмешательства, так и от ошибочного доступа пользователей. Механизм *middleware* обеспечивает проверку сес-

					ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОЕКТА	Лист
Изм.	Лист	№ докум.	Подп.	Дата		49

сии и прав доступа до загрузки страниц, а редактор *tiptap* гарантирует безопасность при работе с форматлируемым текстом. Такое сочетание архитектурных и прикладных средств создаёт устойчивую и безопасную пользовательскую среду.

2.6 Покрывание 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

					ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОЕКТА	Лист
Изм.	Лист	№ докум.	Подп.	Дата		50

мы применяем человеческую экспертизу для финальной проверки внешнего вида и пользовательского опыта.

					ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОЕКТА	Лист
						51
Изм.	Лист	№ докум.	Подп.	Дата		

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), не содержащие бизнес-логику и используемые в различных контекстах;
- Провайдеры контекста: *DragAndDropFilesProvider*, *EnterKeyHandlerProvider* и другие, обеспечивающие единообразную работу с событиями и состояниями по всему приложению;
- 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, сокращающие дублирование кода;
- SVG-иконки и графика: импорт через SVGR для единообразного подключения и управления атрибутами SVG;

					ПРОГРАММНАЯ РЕАЛИЗАЦИЯ			
Изм.	Лист	№ докум.	Подп.	Дата				
Разраб.	Бондаренко С. В.				Разработка front-end Web-приложения – учебной среды с чатами и AI-анализом кода лабораторных работ	Лит.	Лист	Листов
Руковод.	Мельников А.Б.						52	86
Консул.						БГТУ им. В.Г. Шухова, ПВ-212		
Н. контр.	Н.Кнтр. И.О.							
Зав. Каф.	Поляков В.М.							

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

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 (например, *UserProfileResponseType*, *OrderCreateRequestBodyType*);
- *models/*: классы и *mapper*-функции для преобразования сырых данных из API в удобные объекты;
- *services/*: обёртки для работы с локальным кэшем (*IndexedDB*, *localStorage*) и реализации *retry*-логики и таймаутов.

3.1.3 Слои *features*, *widgets*, *pages*

Главные рабочие слои приложения, отвечающие за реализацию конкретной функциональности:

- 1) *pages*: маршрутизация и верхний уровень страниц, описывающий пути, *guards* для доступа, асинхронную загрузку данных и выбор виджетов;
- 2) *widgets*: презентационные и «умные» компоненты по Smart/Presentational-паттерну:
 - *Presentational Component* — презентационный компонент, содержащий исключительно UI-логику и пропсы, без работы с API и глобальным состоянием;
 - *Smart Hook* — умный хук, содержащий всю бизнес-логику и передающий необходимые данные в презентационные компоненты.
- 3) *features*: бизнес-логика и состояние в виде кастомных хуков, редьюсеров, слайсов *Redux* или *Zustand*:

					ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	Лист
						53
Изм.	Лист	№ докум.	Подп.	Дата		

- *hooks.ts*: главный хук-фабрика (например, *useRegistrationUniversity*);
- *schema.ts*: схемы форм;
- *utils.ts*: вспомогательные функции и селекторы;
- *store.ts*: подключение к Redux/Redux Toolkit.

3.1.4 Пример виджета RegistrationUniversity

В листинге 3.1 представлен презентационный компонент, отвечающий за отображение формы регистрации университета.

Листинг 3.1 – RegistrationUniversityWidget

```

1 // Presentationкомпонент-
2 export function RegistrationUniversityWidget() {
3   const { formDataRef, isError, setIsError, onSubmit, isLoading } =
4     useRegistrationUniversity();
5
6   return (
7     <div>...</div>
8   );
9 }

```

В листинге 3.2 представлен smart-hook, содержащий логику взаимодействия пользователя с формой регистрации университета и обработки отправки запроса на регистрацию.

Листинг 3.2 – useRegistrationUniversity

```

1 // Smartкомпонент-: хукфабрика-
2 export function useRegistrationUniversity() {
3   const formDataRef = useRef<RegistrationData>();
4   const [isError, setIsError] = useState<string[]>([]);
5   const [isLoading, setIsLoading] = useState(false);
6   const router = useRouter();
7
8   const onSubmit = async () => {
9     ...
10  };
11  return { formDataRef, isError, setIsError, onSubmit, isLoading };
12 }

```

3.1.5 Соглашения по именованию и структуре

Для поддержания единого стиля и предсказуемости структуры проекта:

					ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		54

- Имена функций API интерфейса и типов дублируют backend-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 описаны в README с диаграммой и примерами использования;
- Код-ревью включает проверку соответствия FSD-подходу и правилам 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:

- 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.

3.2.2 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:

- 1) Пользователь задаёт схему параметров формы, представляющую собой массив объектов с полем *type* и соответствующим набором *props*.

					ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		55

- 2) *FormBuilder* инициализирует внутреннее состояние формы и передаёт каждому полю текущие значения и функции обработки изменений.
- 3) При срабатывании события изменения значения поле уведомляет *FormBuilder*, который обновляет общую модель данных и вызывает коллбэк *onChange*.

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).

Пример описания схемы формы приведён в листинге 3.3.

Листинг 3.3 – Пример описания схемы формы

```

1  export function inviteTeacherScheme(): FORM_BUILDER_SCHEMA {
2      return [
3          {
4              type: 'input_field',
5              props: {
6                  name: 'email',
7                  labelText: 'Email'
8              }
9          },
10         {
11             type: 'input_field',
12             props: {
13                 type: 'select',
14                 name: 'department_id',
15                 ownerInputComponent: <DepartmentSelectField />
16             }
17         }
18     ];
19 }

```

Пример использования компонента *FormBuilder* приведён в листинге 3.4.

Листинг 3.4 – Использование *FormBuilder*

```

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

```


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

3.2.3 Типы элементов схемы и их поведение

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

Схема *INPUT_FIELD_SCHEMA* 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*.

Пример использования схемы представлен в листинге 3.5.

Листинг 3.5 – Пример *INPUT_FIELD_SCHEMA*

```
1 const schema: INPUT_FIELD_SCHEMA = {  
2   type: 'input_field',  
3   props: {  
4     name: 'username',  
5     labelText: 'Имя пользователя',  
6     hintText: 'Введите ваш логин'  
7   }  
8 };
```

Схема *ARRAY_FIELDS_SCHEMA* предназначена 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.

Пример схемы показан в листинге 3.6.

					ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		57

Листинг 3.6 – Пример *ARRAY_FIELDS_SCHEMA*

```

1  const schema: ARRAY_FIELDS_SCHEMA = [
2    {
3      type: 'array_fields',
4      props: {
5        name: 'subjects',
6        children: [
7          {
8            type: 'input_field',
9            props: {
10              name: 'subjectName',
11              labelText: 'Название предмета'
12            }
13          }
14        ]
15      }
16    }
17  ];

```

Схема *FORM_WRAPPER_SCHEMA* служит для группировки полей без сброса счётчика массивов. Внутренние поля записываются как вложенный объект.

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

Пример схемы показан в листинге 3.7.

Листинг 3.7 – Пример *FORM_WRAPPER_SCHEMA*

```

1  const schema: FORM_WRAPPER_SCHEMA = [{
2    type: 'form_wrapper',
3    props: {
4      name: 'teacherInfo',
5      children: [
6        {
7          type: 'input_field',
8          props: { name: 'firstName', labelText: 'Имя' }
9        },
10       {
11         type: 'input_field',
12         props: { name: 'lastName', labelText: 'Фамилия' }
13       }
14     ]
15   }
16 }];

```

					ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		58

Схема *BLOCK_WRAPPER_SCHEMA* позволяет визуально группировать элементы без изменения структуры данных: поля в нём обрабатываются как часть текущего массива или объекта. Пример схемы показан в листинге 3.8.

Листинг 3.8 – Пример *BLOCK_WRAPPER_SCHEMA*

```
1  const schema: BLOCK_WRAPPER_SCHEMA = [  
2    {  
3      type: 'block_wrapper',  
4      props: {  
5        children: [  
6          {  
7            type: 'input_field',  
8            props: {  
9              name: 'code',  
10             labelText: 'Код'  
11           }  
12         ]  
13       }  
14     }  
15   ]  
16 ];
```

Схема *REACT_NODE_SCHEMA* предназначена для вставки произвольного React-элемента в форму. Пример схемы показан в листинге 3.9.

Листинг 3.9 – Пример *REACT_NODE_SCHEMA*

```
1  const schema: REACT_NODE_SCHEMA = [  
2    {  
3      type: 'react_node',  
4      props: {  
5        node: <CustomSeparator />  
6      }  
7    }  
8  ];
```

3.2.4 Преимущества и выводы

Использование компонента *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.

3.2.5 Работа с токеном JWT

Для передачи и обновления JWT в сессии используется функция обратного вызова *jwt*, принимающий параметр *trigger*, позволяющий определить сценарий обработки. Логика работы примерно следующая: при первом входе пользователя (*trigger* = *signIn*) в токен записываются поля *access_token*, *refresh_token* и прочие метаданные; при последующих запросах проверяется срок жизни *access_token*, и при необходимости инициируется процесс его обновления (*trigger* = *update*). Если же токен ещё валиден, возвращается неизменная структура. Пример реализации приведён в листинге 3.10.

Листинг 3.10 – JWT-callback с учётом trigger

```

1  async jwt({ token, trigger, user, session }): Promise<JWT> {
2    // Срабатывает при первичной аутентификации (signIn)
3    if (trigger === 'signIn') {
4      return { ...user, error: null };
5    }
6    // Если токена ещё нет например(, при восстановлении сессии из куки)
7    if (token = null) {
8      return { ...session?.user, error: 'another' } as JWT;
9    }
10   // При запросе обновления (trigger = 'update')
11   if (trigger === 'update') {
12     return await refreshingProcess(token);
13   }
14   // Во всех остальных случаях токен( валиден), возвращаем прежнее
15   ↪ состояние
16   return { ...token, error: null };
17 }

```

Ниже на рисунке 3.1 показана вся последовательная диаграмма, иллюстрирующая проверку срока жизни JWT на клиенте и, при необходимости, получение нового токена по refresh-токену. Эта схема помогает понять, как именно Auth.js (или NextAuth.js) взаимодействует с бэкендом для устойчивого хранения и своевременного обновления токенов без лишних повторных запросов.

					ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		60

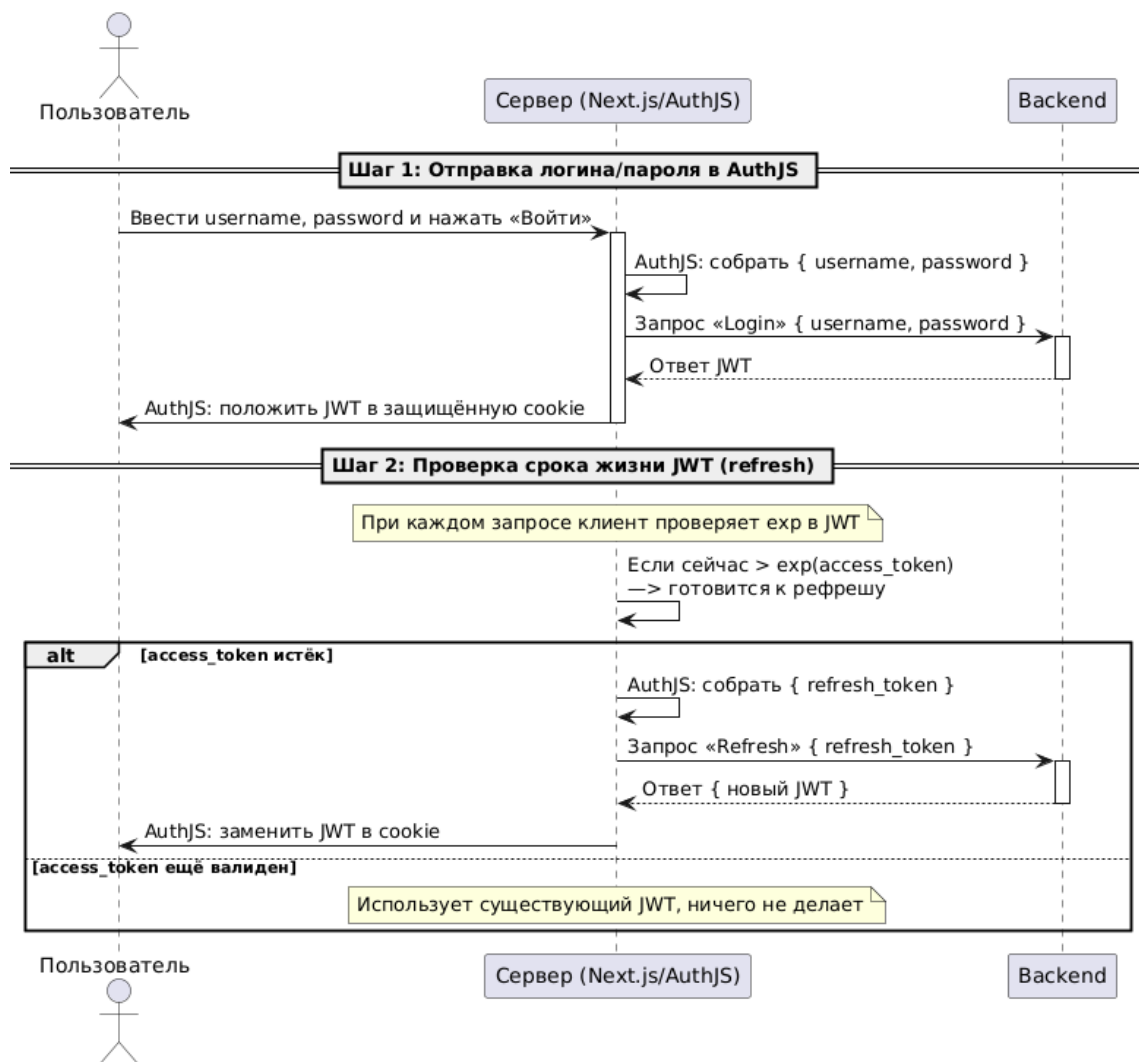


Рисунок 3.1 – Схема процесса проверки и обновления JWT через refresh-token

На рисунке 3.1 можно выделить два основных этапа: аутентификация пользователя и процесс использования токена.

При входе пользователя в систему происходит следующая последовательность действий:

- 1) пользователь вводит *username* и *password* в приложение,
- 2) Auth.js формирует объект с учётными данными и отправляет запрос на бэкэнд (метод *login*),
- 3) бэкэнд возвращает пару токенов: *access_token* и *refresh_token*,
- 4) Auth.js сохраняет полученный JWT в защищённую cookie или в хранилище клиента.

После успешной аутентификации при каждом запросе выполняется проверка срока жизни токена и, при необходимости, его обновление:

- 1) при каждом запросе клиент проверяет поле *exp* (срок жизни) в *access_token*,
- 2) если текущий момент времени превысил *exp(access_token)*, начинается подготовка к рефрешу,
- 3) в блоке *alt* (альтернативный сценарий) на диаграмме показано, что в случае истёкшего access-токена:
 - Auth.js собирает объект с *refresh_token*,
 - отправляется запрос «Refresh» на бэкенд, передаётся *refresh_token*,
 - бэкенд возвращает новый *access_token* и, при необходимости, обновлённый *refresh_token*,
 - Auth.js заменяет старый JWT в cookie на новый.
- 4) если же *access_token* ещё валиден, Auth.js просто использует существующий токен и не делает дополнительных запросов (промежуточные уведомления внизу диаграммы).

Таким образом, схема на рисунке 3.1 демонстрирует, что клиент всегда сначала пробует воспользоваться существующим *access_token*, проверяя его валидность. Только если проверка не проходит, выполняется последовательность обновления, благодаря чему повышается отказоустойчивость и исключается ситуация «гонки» при параллельных запросах на обновление.

Важным дополнением к этой концепции является механизм предотвращения *race condition* при одновременном запросе нескольких API-методов, обнаруживающих, что access-токен просрочен. В таких случаях на клиенте сохраняется единственный промис обновления, который переиспользуется всеми последующими запросами до получения ответа от бэкенда. Пример реализации этого механизма приведён в листинге 3.11.

Листинг 3.11 – Механизм предотвращения *race condition* при рефреше токена

```
1 export type RefreshPromiseStateType = Promise<JWT | null> | null;
```

					ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		62

```

2  let tokenPromiseState: RefreshPromiseStateType = null;
3  export const setTokenPromiseState = (
4    promise: RefreshPromiseStateType
5  ): void => {
6    tokenPromiseState = promise;
7  };
8  export const getTokenPromiseState = (): RefreshPromiseStateType => {
9    return tokenPromiseState;
10 };
11
12 let tokenState: JWT | null = null;
13 export const setTokenState = (newTokenState: JWT | null): void => {
14   tokenState = newTokenState;
15 };
16 export const getTokenState = (): JWT | null => {
17   return tokenState;
18 };
19
20 let timeoutState: NodeJS.Timeout | null = null;

```

На основе приведённой логики обеспечивается централизованная обработка авторизации и обновления токенов без дублирования кода в разных частях приложения. Кроме того, использование одной общей очереди запросов к API для обновления *access_token* предотвращает нежелательные состояния гонки и лишние обращения к бэкенду.

3.3 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.

Выбор формы регистрации осуществляется в зависимости от типа пользователя, переданного через параметры URL. Пример логики выбора виджета приведён в листинге 3.12.

Листинг 3.12 – Выбор виджета регистрации по типу

```

1  const getForm = () => {
2    const type = getSearchParams(REGISTRATION_TYPE_PARAM_NAME) as
    ↪ RegistrationTypesType;
3    const inviteId = getInviteId();
4    switch (type) {
5      case 'teacher':
6        return <RegistrationTeacherWidget inviteId={inviteId} />;

```

					ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		63

```

7      case 'student':
8          return <RegistrationStudentWidget inviteId={inviteId} />;
9      case 'university':
10         default:
11             return <RegistrationUniversityWidget />;
12     }
13 };

```

На рисунке 3.2 показан интерфейс страницы регистрации университета. Пользователь должен ввести название учебного заведения, фамилию, имя и отчество контактного лица, email администрации университета, а также задать и подтвердить пароль. После заполнения всех обязательных полей и нажатия кнопки «Зарегистрировать университет» инициируется отправка данных на сервер.

Рисунок 3.2 – Страница регистрации университета: поля для названия института, контактного лица (ФИО), email администрации, пароля и подтверждения пароля.

Листинг 3.13 демонстрирует структуру виджета регистрации преподавателя. Здесь реализована обработка валидности приглашения и логика отобра-

					ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		64

жения состояния формы в зависимости от результата асинхронной проверки.

Листинг 3.13 – Виджет регистрации преподавателя

```
1 export function RegistrationTeacherWidget({ inviteId }:  
  ⇨ RegistrationPropsType) {  
2   const { initData, onSubmit, isError, setIsError, formDataRef } =  
3     useRegistrationStudentAndTeacher<typeof registerTeacher>({  
4       inviteId,  
5       registrationRequest: registerTeacher  
6     });  
7  
8   if (initData === undefined) {  
9     return 'Loading';  
10  }  
11  
12  if (initData === null) {  
13    // Неверное приглашение или оно истекло  
14    return 'Error';  
15  }  
16  
17  // Дальнейшая логика отображения формы регистрации преподавателя  
18  ...  
19 }
```

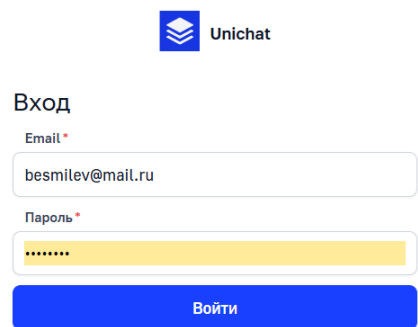


Рисунок 3.3 – Экран входа в систему: поля для ввода Email и пароля, а также кнопка «Войти».

На рисунке 3.3 представлен экран входа в систему. Для авторизации пользователь вводит Email и пароль, после чего нажимает кнопку «Войти». В слу-

					ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		65

чае успешной авторизации система перенаправляет его в личный кабинет; при некорректных учётных данных отображается сообщение об ошибке.

3.3.1 Модуль формирования и отправки приглашений

В административной панели университета реализован отдельный модуль, который позволяет формировать и отправлять приглашения как для преподавателей, так и для студентов. Этот модуль работает по следующему сценарию:

1) Инициация приглашения:

- администратор вводит в UI адрес электронной почты и идентификатор кафедры (для приглашения преподавателя) либо адрес электронной почты и идентификатор группы (для приглашения студента),
- после заполнения необходимых полей администратор нажимает кнопку «Сформировать ссылку».

2) Отправка запроса на бэкенд:

- UI передаёт запрос на бэкенд, например, «Запрос на приглашение преподавателя (требуется: email, department_id)» или «Запрос на приглашение студента (требуется: email, group_id)»,
- бэкенд проверяет права администратора и корректность введённых данных, генерирует уникальную ссылку приглашения и возвращает её обратно в UI.

3) Получение ответа и отображение ссылки:

- UI получает от бэкенда объект с полем *invite_link*,
- UI отображает администратору уведомление «Приглашение отправлено» и показывает полученную ссылку, которую можно скопировать и отправить по электронной почте.

На рисунке 3.4 приведена последовательная диаграмма, иллюстрирующая полный процесс формирования и отправки приглашений: в верхней части — сценарий для преподавателя, в нижней части — сценарий для студента.

На рисунке 3.4 можно выделить следующие ключевые этапы:

					ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		66

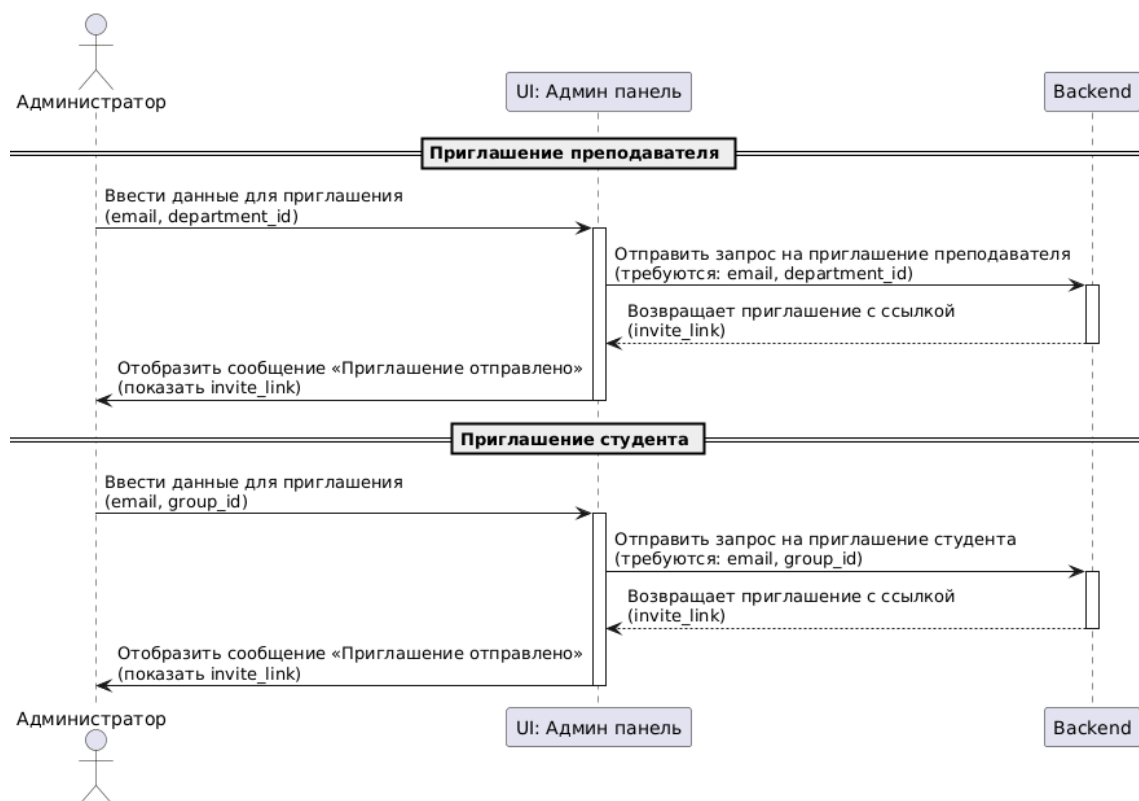


Рисунок 3.4 – Схема процесса формирования и отправки приглашений преподавателям и студентам

– Приглашение преподавателя:

- 1) администратор вводит в UI email и *department_id*,
- 2) UI отправляет запрос на бэкенд «Приглашение преподавателя (требуется: email, department_id)»,
- 3) бэкенд проверяет данные, создаёт приглашение и возвращает уникальную ссылку (*invite_link*),
- 4) UI отображает сообщение «Приглашение отправлено» и показывает *invite_link*.

– Приглашение студента:

- 1) администратор вводит в UI email и *group_id*,
- 2) UI отправляет запрос на бэкенд «Приглашение студента (требуется: email, group_id)»,

- 3) бэкенд проверяет данные, создаёт приглашение и возвращает уникальную ссылку (*invite_link*),
- 4) UI отображает сообщение «Приглашение отправлено» и показывает *invite_link*.

Таким образом, схема на рисунке 3.4 демонстрирует единый алгоритм работы модуля: ввод данных в UI, отправка запроса на бэкенд, генерация и возврат уникальной ссылки, отображение ссылки администратору.

3.3.2 Последовательная диаграмма работы модуля Classrooms

Ниже приведена последовательная диаграмма, иллюстрирующая полный жизненный цикл взаимодействия преподавателя, студента, и сервиса искусственного интеллекта при работе с виртуальными классами и заданиями. На рисунке А.3 показаны все основные этапы: создание класса, создание задания с указанием промта для автопроверки, получение списков заданий студентом, отправка решения студентом, автоматическая проверка с помощью искусственного интеллекта и выставление оценки преподавателем.

На рисунке А.3 можно выделить следующие ключевые этапы:

1) Преподаватель создаёт класс:

- преподаватель открывает форму создания класса в своем интерфейсе (UI: Преподаватель),
- UI отправляет запрос на бэкенд с данными нового класса,
- бэкенд возвращает подтверждение успешного создания (например, ID нового класса),
- UI отображает преподавателю сообщение «Класс создан».

2) Преподаватель создаёт задание с возможностью задать промт для проверки искусственным интеллектом:

- преподаватель переходит в форму создания задания, указывая вместе с условием текста задания промт для проверки искусственным интеллектом,

					ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		68

- UI отправляет запрос на бэкенд с данными задания и промтом,
- бэкенд возвращает подтверждение успешного сохранения задания,
- UI отображает преподавателю сообщение «Задание создано».

3) Студент получает список доступных заданий:

- студент открывает интерфейс (UI: Студент) и запрашивает список заданий для конкретного класса,
- UI отправляет запрос на бэкенд с ID класса,
- бэкенд возвращает массив доступных заданий,
- UI отображает студенту список заданий.

4) Студент отправляет решение на проверку:

- студент открывает форму отправки решения, выбирая конкретное задание,
- UI отправляет файлы решения и метаданные (например, ID задания, ID студента) на бэкенд,
- бэкенд возвращает подтверждение успешной загрузки решения,
- UI отображает студенту сообщение «Решение отправлено».

5) Автопроверка работы искусственным интеллектом:

- бэкенд получает файлы решения и ранее заданный промт к заданию,
- бэкенд отправляет файлы и промт во внешний сервис с использованием искусственного интеллекта,
- сервис с использованием искусственного интеллекта выполняет анализ кода (например, проверку корректности, стилизованных нарушений и т. д.) и возвращает результат вместе с комментариями,
- бэкенд сохраняет результат автопроверки и передаёт его UI обоим ролям:

					ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	Лист
						69
Изм.	Лист	№ докум.	Подп.	Дата		

- a) UI Студента: отображается результат автопроверки (оценка искусственного интеллекта, комментарии),
- b) UI Преподавателя: отображается результат автопроверки (для последующей ручной проверки и выставления итоговой оценки).

6) Преподаватель ставит оценку, и студент получает её:

- преподаватель открывает форму выставления оценки (UI: Преподаватель) для конкретного решения,
- UI отправляет в бэкенд оценку и комментарий преподавателя,
- бэкенд сохраняет оценку, возвращает подтверждение сохранения,
- UI отображает преподавателю сообщение «Оценка сохранена», а UI Студента — обновлённую финальную оценку.

Таким образом, последовательная диаграмма на рисунке А.3 демонстрирует весь цикл взаимодействий: от создания класса и задания преподавателем до получения студентом финальной оценки после автопроверки и ручного выставления оценки преподавателем.

3.3.3 Последовательная диаграмма работы модуля Chats

Модуль «Chats» обеспечивает обмен сообщениями в реальном времени между пользователями системы (преподавателями и студентами) посредством WebSocket-соединения. Ниже на рисунке А.2 приведена последовательная диаграмма, демонстрирующая ключевые этапы работы чата: открытие приложения, выбор беседы, отправка и приём сообщений, а также обработку истечения JWT и повторное подключение.

На рисунке А.2 выделены следующие этапы:

1) Открытие приложения:

- пользователь переходит в раздел «Чаты», UI боковой панели запрашивает и отображает список доступных бесед (из кэша или по HTTP),

					ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		70

- при загрузке страницы «Чат» UI извлекает *access_token* из защищённой cookie (AuthJS),
- после получения токена устанавливается WebSocket-соединение с сервером, передавая JWT в заголовке, бэкенд подтверждает подключение.

2) Выбор чата в боковой панели:

- пользователь кликает на одну из бесед (параметр *chatId*), боковая панель передаёт этот *chatId* компоненту «Чат»,
- UI «Чат» выполняет HTTP GET-запрос к эндпоинту */chats/{chatId}/messages?page=1* для получения первой страницы сообщений,
- бэкенд возвращает список сообщений, которые UI отображает в области истории сообщений.

3) Отправка нового сообщения:

- пользователь пишет текст сообщения и нажимает «Отправить» в UI «Чат»,
- UI «Чат» эмиттит событие *send_message* по WebSocket, передавая объект *{chat: chatId, message: {...}}*,
- бэкенд принимает событие, сохраняет сообщение и возвращает подтверждение приёма,
- UI «Чат» временно отображает отправленное сообщение (optimistic UI) со статусом «отправляется» до получения фактического сообщения от сервера.

4) Получение нового сообщения (*new_message*):

- когда любой участник (в том числе другой пользователь) отправляет сообщение, сервер по WebSocket рассылает событие *new_message* всем подписанным участникам данного чата,

					ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	Лист
						71
Изм.	Лист	№ докум.	Подп.	Дата		

- UI «Чат» получает событие *new_message { chat, message }* и добавляет новое сообщение в конец списка истории,
- боковая панель получает обновлённый *chatId* для обновления списка бесед (например, переставить текущую беседу наверх) и отображает актуальный список чатов с учётом новых сообщений.

5) Редактирование сообщения (*edit_message*):

- если пользователь (автор сообщения) редактирует ранее отправленное сообщение, UI «Чат» отправляет серверу по WebSocket событие *edit_message { chat, message }*,
- сервер обновляет текст сообщения и рассылает событие *edit_message* всем участникам беседы,
- UI «Чат» находит сообщение по его *id* или *local_id* и обновляет текст,
- после этого боковая панель получает сигнал «передать *chatId*» для обновления порядка бесед и отображает обновлённый список чатов.

6) Отметка сообщения как прочитанного (*read_message*):

- когда пользователь прокручивает историю и дожидается видимости новых сообщений, UI «Чат» отправляет событие *read_message { chat, message }* по WebSocket,
- сервер обновляет статус сообщения на «прочитано» и уведомляет других участников через событие *read_message*,
- UI «Чат» и боковая панель получают это событие, обновляют статус соответствующего сообщения и обновляют отображение списка чатов (например, убрать бейдж «новых сообщений»).

7) Обработка истечения JWT и переподключение:

- если при работе WebSocket возникает ошибка авторизации (например, *access_token* истёк), UI «Чат» получает событие *connect_error* от библиотеки Socket.IO,

					ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		72

- компонент проверяет поле *exp(access_token)* локально, при истечении:
- а) UI «Чат» отправляет *refresh_token* на AuthJS/Next.js, получая новый *access_token*,
- б) после получения нового токена выполняется повторное подключение WebSocket, передавая обновлённый JWT,
- в) сервер WebSocket подтверждает новый сеанс подключения.
- боковая панель и UI «Чат» возобновляют подписку на события и продолжают обмен сообщениями без потери данных.

Таким образом, последовательная диаграмма на рисунке А.2 отражает полный цикл работы модуля «Chats»: от открытия приложения и установления защищённого соединения до приёма, отправки, редактирования и пометки сообщений, а также автоматического обновления JWT и переподключения WebSocket. Все события обрабатываются как в компоненте UI «Чат», так и в боковой панели, чтобы гарантировать актуальность списка бесед и статусов сообщений.

3.4 Тестирование

В ходе разработки были протестированы модули как основного приложения, так и используемой при разработке библиотеки.

3.4.1 Покрытие кода в приложении

Таблица 3.1 – Покрытие кода front-end приложения.

File	%Stmts	%Branch	%Funcs	%Lines
All files	95.65	77.77	100.00	100.00
features/Chats/lib	100.00	100.00	100.00	100.00
mergeMessages.ts	100.00	100.00	100.00	100.00
shared/ui/TextEditor/lib	90.47	68.42	100.00	100.00
processTextToTiptap.ts	94.11	72.22	100.00	100.00
processTiptapToText.ts	75.00	0.00	100.00	100.00

В таблице 3.1 приведены четыре основные метрики покрытия:

					ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		73

- %Stmts (Statements) — процент операторов кода, выполненных в ходе тестов;
- %Branch (Branches) — процент ветвей условных операторов, затронутых тестами;
- %Funcs (Functions) — процент функций, вызванных хотя бы одним тестом;
- %Lines (Lines) — процент строк кода, исполненных тестами.

Для ключевых модулей (например, *mergeMessages.ts*) все метрики равны 100 %.

3.4.2 Покрытие кода в отдельной библиотеке

Таблица 3.2 – Покрытие кода библиотеки компонентов.

File	%Stmts	%Branch	%Funcs	%Lines
All files	52.89	48.76	20.68	52.65
lib/dict/getDeepValue.ts	86.36	73.33	100.00	85.71
lib/dict/setDeepValue.ts	100.00	100.00	100.00	100.00
ui/DateTimePicker/lib/changeInterval.ts	100.00	96.29	100.00	100.00

В таблице 3.2 показано, что основные утилитные функции библиотеки (*getDeepValue*, *setDeepValue*, *changeInterval*) полностью или почти полностью покрыты.

3.4.3 Методы тестирования

- Модульное тестирование (unit testing): для каждой функции и компонента написаны независимые тесты, покрывающие: граничные и некорректные входные данные (*undefined*, пустые массивы), типичные сценарии и пограничные случаи.
- TDD-подход (Test-Driven Development): реализация функций по циклу «*test* → *fail* → написать минимальный код → *test* → *pass* → рефакторинг».
- Покрытие ветвлений (branch coverage): каждый сценарий условных операторов (*if/else*, тернарные выражения, *switch*) проверяется отдельными тестами.

- Round-trip-тесты: для преобразований «текст → HTML → текст» (функции *processTextToTiptap* / *processTiptapToText*) проверяется обратимость и корректность в сложных случаях (вложенные теги, переносы строк).
- Инструментация и сбор покрытия: запуск командой *npx jest --coverage* автоматически оборачивает счётчиками все исходники и собирает метрики Statements, Branches, Functions и Lines.

					ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		75

ЗАКЛЮЧЕНИЕ

В рамках проведённой дипломной работы было улучшено и упрощено взаимодействие преподавателей и студентов в образовательном процессе университета посредством разработки клиентской части интегрированного образовательного приложения. Основная цель заключалась в повышении качества опыта работы преподавателей и обучения студентов, обеспечивая единое пространство для коммуникаций, автоматизированной проверки заданий и управления учебными ресурсами.

Для достижения поставленной цели были выполнены следующие задачи:

- 1) проанализирована предметная область и существующие образовательные системы, выявлены их сильные и слабые стороны;
- 2) определены архитектурные и технологические решения (Next.js, React, TypeScript, Redux, Auth.js, Socket.IO) для реализации гибкой, надёжной и удобной клиентской части приложения;
- 3) спроектирован пользовательский интерфейс, обеспечивающий интуитивное и удобное взаимодействие для преподавателей и студентов (навигация, адаптивная вёрстка, единые UI-компоненты);
- 4) разработаны компоненты для управления учебными структурами (институты, кафедры, группы), заданиями и чатами, включая административную панель, что упростило работу преподавателей при ведении учебного процесса;
- 5) интегрированы средства автоматизированной проверки решений студентов с применением ИИ (DeepSeek) в модуле виртуальных классов, что повысило качество и скорость проверки лабораторных работ;
- 6) реализовано тестирование бизнес-логики клиентского приложения с использованием Jest, что подтвердило корректность и стабильность основных функций.

					ЗАКЛЮЧЕНИЕ			
Изм.	Лист	№ докум.	Подп.	Дата				
Разраб.	Бондаренко С. В.				Разработка front-end Web-приложения – учебной среды с чатами и AI-анализом кода лабораторных работ	Лит.	Лист	Листов
Руковод.	Мельников А.Б.						76	86
Консул.						БГТУ им. В.Г. Шухова, ПВ-212		
Н. контр.	Н.Кнтр. И.О.							
Зав. Каф.	Поляков В.М.							

Таким образом, все поставленные задачи выполнены, а основная цель достигнута: предложенные архитектурные решения обеспечили удобство работы преподавателей и улучшили опыт обучения студентов. Разработанный интерфейс продемонстрировал эффективность в упрощении коммуникаций, автоматизации рутинных задач и повышении качества образовательного процесса.

С практической точки зрения, данное клиентское приложение обладает следующими преимуществами:

- быстрая разработка за счёт переиспользования компонентов и генерации форм, что позволяет преподавателям оперативно адаптировать интерфейс под учебный процесс;
- надёжная безопасность благодаря механизму управления сессиями (JWT, Auth.js) и авторизации;
- удобство сопровождения за счёт модульной структуры и чёткого разделения ответственности, что облегчает расширение и поддержку проекта;
- расширяемость и готовность к новым сценариям благодаря гибкой конфигурации интерфейса, что позволяет быстро внедрять новые образовательные инструменты;
- высокая устойчивость системы, подтверждённая результатами тестирования бизнес-логики, что гарантирует стабильность работы приложения для всех участников образовательного процесса.

Перспективными направлениями дальнейшего развития являются:

- Расширение функционала анализа кода на основе искусственного интеллекта за счёт надстроек для автоматической проверки студенческих работ по заданным паттернам;
- добавление офлайн-режима работы с последующей синхронизацией изменений при восстановлении связи, что улучшит опыт студентов в условиях нестабильного интернета;
- разработка мобильных клиентских приложений для повышения доступности приложения и удобства работы преподавателей и студентов на разных устройствах;

					ЗАКЛЮЧЕНИЕ	Лист
						77
Изм.	Лист	№ докум.	Подп.	Дата		

- внедрение системы аналитики пользовательской активности для оптимизации интерфейса, оценки эффективности учебного процесса и принятия обоснованных решений по улучшению образовательной среды.

Итоги работы подтверждают, что предложенные решения действительно упростили работу преподавателей, улучшили опыт обучения студентов и повысили качество образовательного процесса. Открываются новые возможности для дальнейших исследований и совершенствования системы.

					ЗАКЛЮЧЕНИЕ			
<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подп.</i>	<i>Дата</i>	<i>Разработка front-end Web-приложения – учебной среды с чатами и AI-анализом кода лабораторных работ</i>	<i>Лит.</i>	<i>Лист</i>	<i>Листов</i>
<i>Разраб.</i>		<i>Бондаренко С. В.</i>						
<i>Руковод.</i>		<i>Мельников А.Б.</i>					78	86
<i>Консул.</i>						<i>БГТУ им. В.Г. Шухова, ПВ-212</i>		
<i>Н. контр.</i>		<i>Н.Кнтр. И.О.</i>						
<i>Зав. Каф.</i>		<i>Поляков В.М.</i>						

СПИСОК ЛИТЕРАТУРЫ

1. Moodle Docs: Main Page. — Accessed: 2025-06-10. https://docs.moodle.org/4/en/Main_page.
2. Google Classroom Documentation. — Accessed: 2025-06-10. <https://edu.google.com/intl/ru/products/classroom/>.
3. Microsoft Teams for Education Documentation. — Accessed: 2025-06-10. <https://learn.microsoft.com/education>.
4. CodeSignal. — Accessed: 2025-06-10. <https://en.wikipedia.org/wiki/CodeSignal>.
5. Codility. — Accessed: 2025-06-10. <https://en.wikipedia.org/wiki/Codility>.
6. LeetCode. — Accessed: 2025-06-10. <https://en.wikipedia.org/wiki/LeetCode>.
7. *TypeScript*. TypeScript Handbook. — Accessed: 2025-06-10. <https://www.typescriptlang.org/docs/handbook/intro.html>.
8. *React*. Getting Started with React. — Accessed: 2025-06-10. <https://reactjs.org/docs/getting-started.html>.
9. *Angular*. What is Angular? — Accessed: 2025-06-10. <https://angular.io/guide/what-is-angular>.
10. *Angular*. Dependency Injection. — Accessed: 2025-06-10. <https://angular.io/guide/dependency-injection>.
11. *Vue.js*. Introduction to Vue.js. — Accessed: 2025-06-10. <https://vuejs.org/v2/guide/>.
12. Next.js Server-Side Rendering. — Accessed: 2025-06-10. <https://nextjs.org/docs/basic-features/pages#server-side-rendering>.
13. Next.js Static Generation and ISR. — Accessed: 2025-06-10. <https://nextjs.org/docs/basic-features/data-fetching/get-static-props>.
14. Socket.IO Documentation. — Accessed: 2025-06-10. <https://socket.io/docs/v4/>.
15. Pusher Documentation. — Accessed: 2025-06-10. <https://pusher.com/docs>.

					СПИСОК ЛИТЕРАТУРЫ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		79

16. Docs M. W. WebSocket API. — Accessed: 2025-06-10. [https : / / developer.mozilla.org/docs/Web/API/WebSocket](https://developer.mozilla.org/docs/Web/API/WebSocket).
17. NextAuth.js Documentation. — Accessed: 2025-06-10. <https://next-auth.js.org/getting-started/introduction>.
18. Redux – Getting Started. — Accessed: 2025-06-10. <https://redux.js.org/introduction/getting-started>.
19. DeepSeek Documentation: Module Overview. — Accessed: 2025-06-10. <https://deepseek.ai/docs/overview>.
20. Jest Documentation. — Accessed: 2025-06-10. <https://jestjs.io/docs/getting-started>.
21. Feature-Sliced Design Documentation. — Accessed: 2025-06-10. [https : //feature-sliced.design/docs/introduction](https://feature-sliced.design/docs/introduction).
22. Middleware. — Accessed: 2025-06-10. <https://nextjs.org/docs/advanced-features/middleware>.

					СПИСОК ЛИТЕРАТУРЫ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		80

ПРИЛОЖЕНИЕ А

Диаграммы

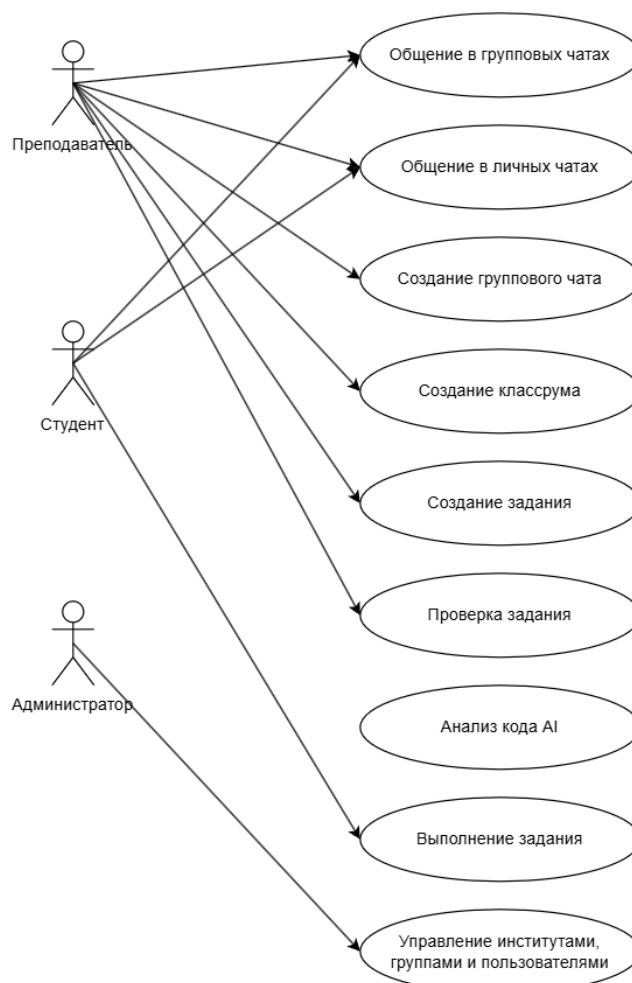


Рисунок А.1 – Диаграмма вариантов использования системы для различных ролей пользователей.

					ПРИЛОЖЕНИЕ Б			
Изм.	Лист	№ докум.	Подп.	Дата				
Разраб.	Бондаренко С. В.				Разработка front-end Web-приложения – учебной среды с чатами и AI-анализом кода лабораторных работ	Лит.	Лист	Листов
Руковод.	Мельников А.Б.						81	86
Консул.						БГТУ им. В.Г. Шухова, ПВ-212		
Н. контр.	Н.Кнтр. И.О.							
Зав. Каф.	Поляков В.М.							

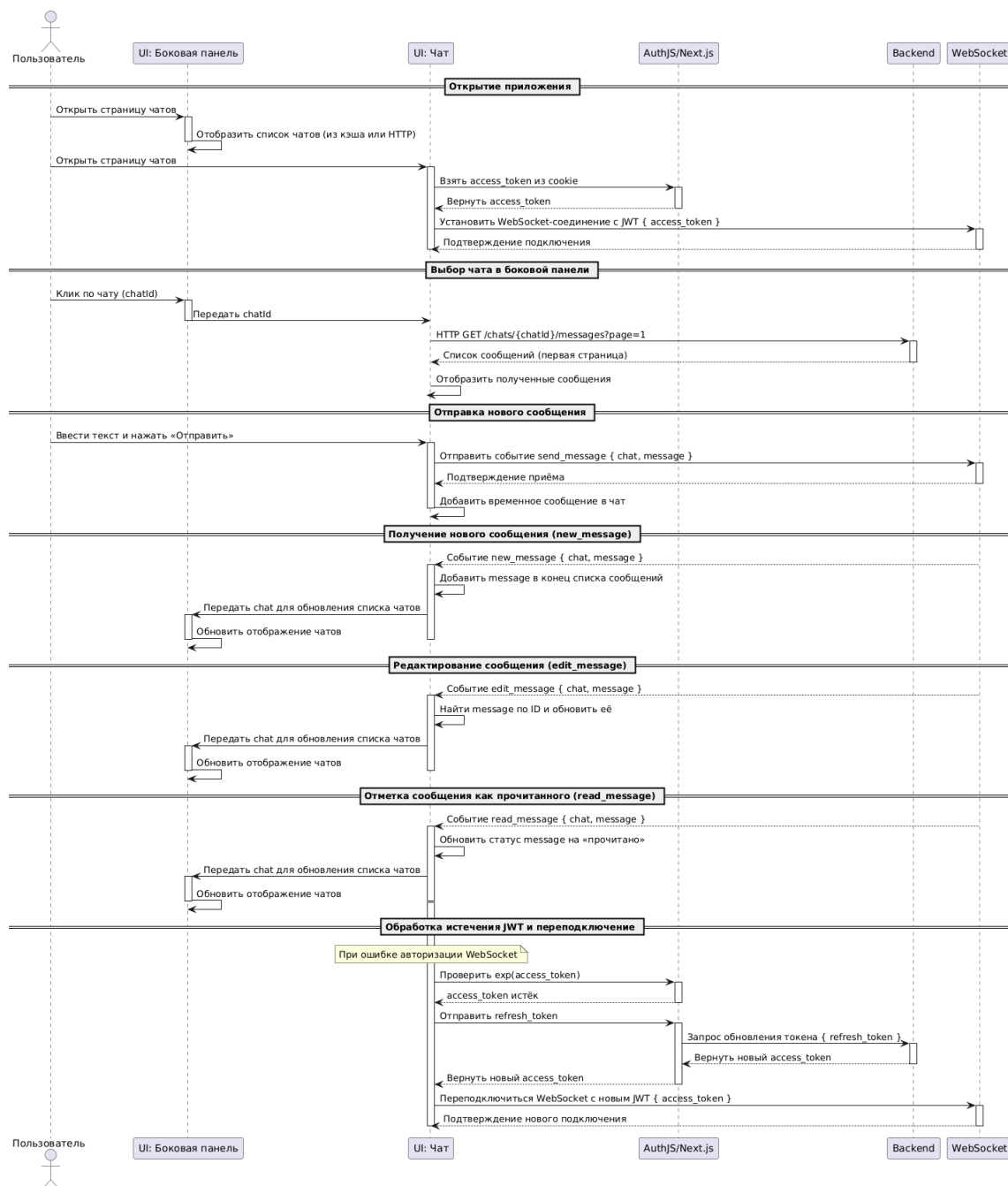


Рисунок А.2 – Схема взаимодействия клиента (UI: Боковая панель и UI: Чат), AuthJS (Next.js), бэкенда и WebSocket при работе модуля «Chats».

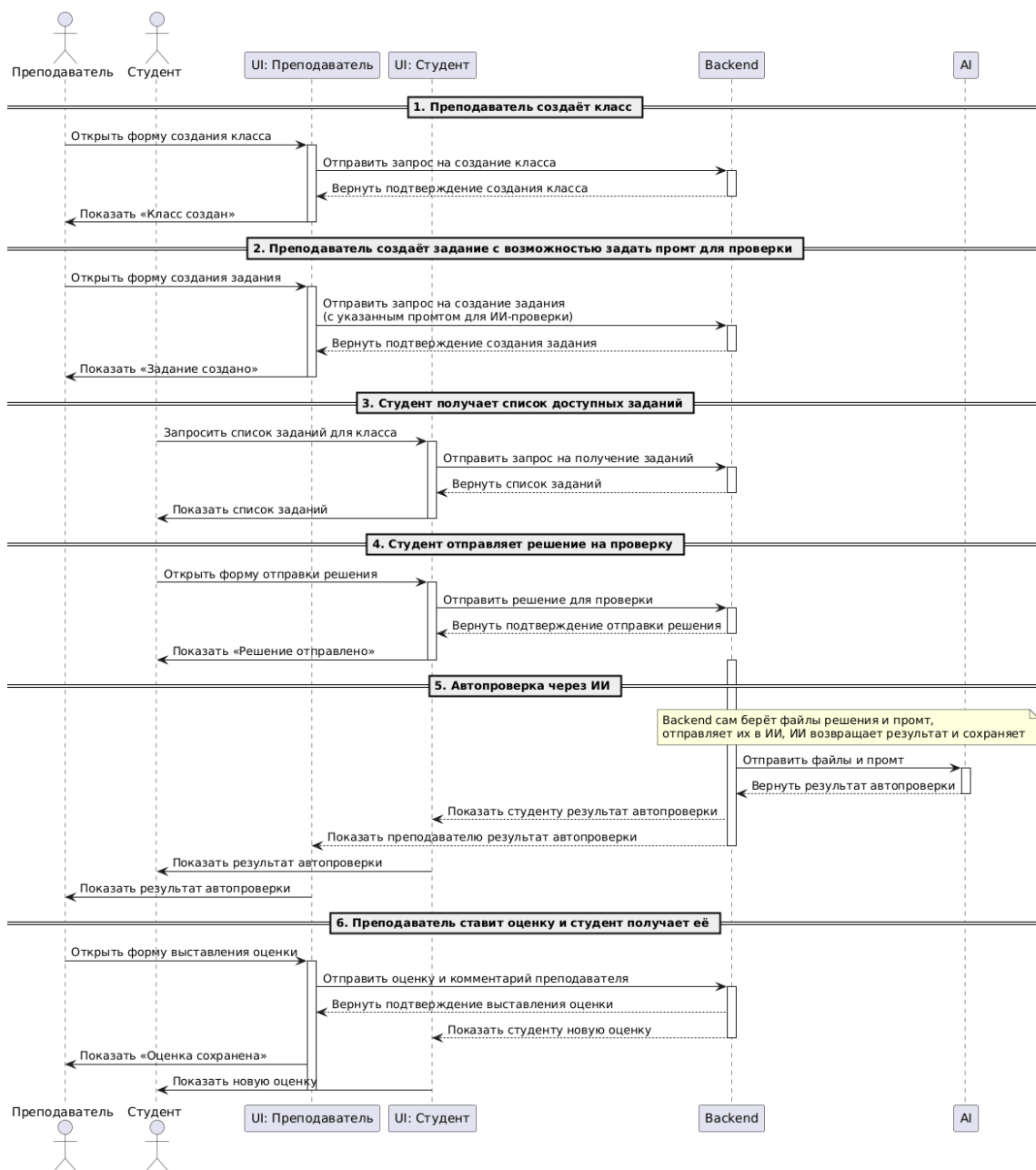


Рисунок А.3 – Схема взаимодействия преподавателя, студента, и искусственного интеллекта при работе с виртуальными классами

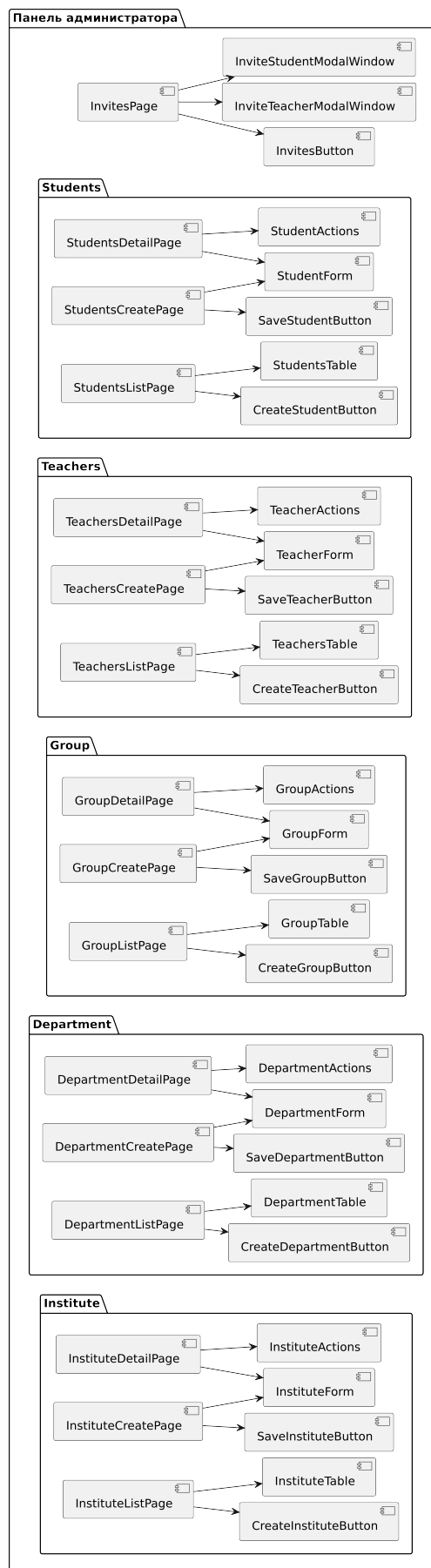


Рисунок А.4 – Диаграмма компонентов административной панели

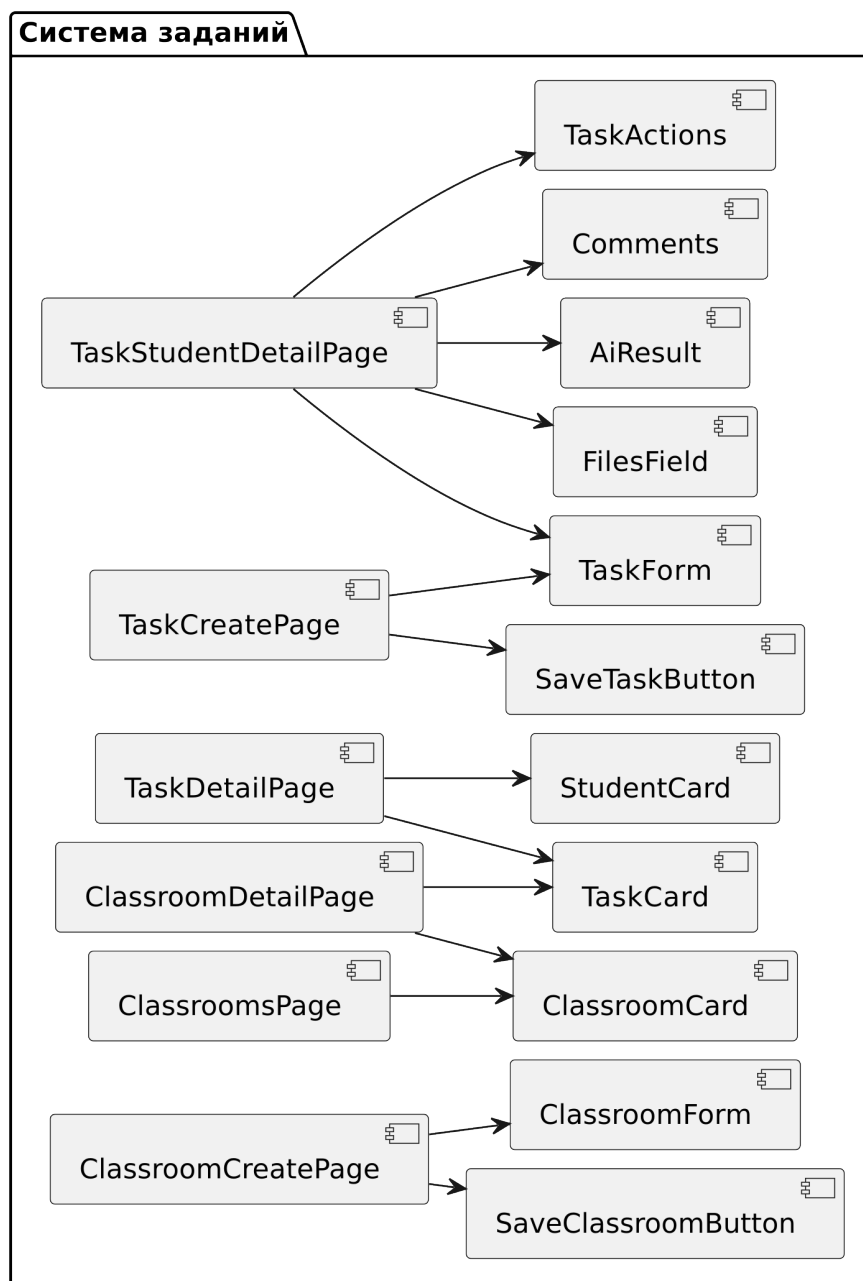


Рисунок А.5 – Диаграмма компонентов системы заданий

ПРИЛОЖЕНИЕ Б

Исходный код программы

По результатам данной работы был реализовано front-end часть приложения. Ознакомится с исходным кодом приложения можно по ссылке на репозиторий GitHub.

					ПРИЛОЖЕНИЕ Б	Лист
Изм.	Лист	№ докум.	Подп.	Дата		86