

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ им. В.Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)

Институт информационных технологий и управляющих систем
Кафедра программного обеспечения вычислительной техники и
автоматизированных систем
Направление подготовки 09.03.04 – Программная инженерия
Направленность (профиль) образовательной программы Разработка
программно-информационных систем

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

на тему:

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

Студент: Бондаренко Сергей Владимирович
Зав. кафедрой: канд. техн. наук, доц. Поляков В.М.
Руководитель: Мельников А.Б.

К защите допустить:

Зав. кафедрой _____ **/Поляков В.М./**

«_____» _____ 2025 г.

Белгород 2025 г.

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ им. В.Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)

Институт информационных технологий и управляющих систем
Кафедра программного обеспечения вычислительной техники и
автоматизированных систем
Направление подготовки 09.03.04 – Программная инженерия
Направленность (профиль) образовательной программы Разработка
программно-информационных систем

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

на тему:

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

Студент: Бондаренко Сергей Владимирович
Зав. кафедрой: канд. техн. наук, доц. Поляков В.М.
Руководитель: Мельников А.Б.

К защите допустить:

Зав. кафедрой _____ **/Поляков В.М./**

«_____» _____ 2025 г.

Белгород 2025 г.

ОПРЕДЕЛЕНИЯ, СОКРАЩЕНИЯ И ОБОЗНАЧЕНИЯ

- ИИ (AI) — искусственный интеллект.
- SPA (Single Page Application) — одностраничное веб-приложение, в котором навигация осуществляется без перезагрузки страниц.
- 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.
- FSD (Feature-Sliced Design) — архитектурный подход к построению фронтенд-приложений, основанный на разделении кода по смысловым «срезам» (feature, widget, entity и др.).
- API (Application Programming Interface) — программный интерфейс, обеспечивающий взаимодействие между компонентами программного обеспечения.
- UI (User Interface) — пользовательский интерфейс.
- DOM (Document Object Model) — объектная модель документа, представляющая HTML-структуру страницы в виде дерева объектов.
- Virtual DOM — виртуальная копия DOM, используемая React для эффективного выявления и обновления изменившихся элементов.

					ОПРЕДЕЛЕНИЯ, СОКРАЩЕНИЯ И ОБОЗНАЧЕНИЯ								
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.										1	93
Консул.									БГТУ им. В.Г. Шухова, ПВ-212				
Н. контр.		Н.Кнтр. И.О.											
Зав. Каф.		Поляков В.М.											

- JSX — синтаксис, объединяющий JavaScript и XML-подобную разметку, применяемый в React.
- React — библиотека JavaScript для построения пользовательских интерфейсов на основе компонентного подхода.
- Next.js — фреймворк на базе React с поддержкой CSR, SSR и SSG.
- TypeScript — надмножество JavaScript, добавляющее статическую типизацию и расширяющее возможности языка.
- Node.js — серверная среда выполнения JavaScript, используемая для построения бэкенд-сервисов и выполнения скриптов на сервере.
- Socket.IO — библиотека для организации двусторонних WebSocket-соединений между клиентом и сервером с автоматическим фоллбеком.
- WebSocket — протокол для установления постоянного соединения между клиентом и сервером и обмена данными в режиме реального времени.
- OAuth — протокол авторизации, позволяющий сторонним приложениям получать доступ к ресурсам через выданные токены.
- 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.
- Auth.js — библиотека для реализации аутентификации и авторизации в приложениях на Next.js (ранее известная как NextAuth.js).

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

- Redux — библиотека для централизованного управления состоянием приложений, основанная на едином хранилище и концепции «actions» и «reducers».
- Redux Toolkit — официальная утилита для упрощённой работы с Redux, предоставляющая набор шаблонных функций и удобных API для создания «слайсов» и сторов.
- Zustand — лёгкая альтернатива Redux для управления состоянием в React-приложениях с минимальным «боли моле».
- HTTP (Hypertext Transfer Protocol) — протокол передачи гипертекстовых данных между клиентом и сервером; используется для REST-запросов.
- 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.
- OAuth2.0 — современная версия протокола OAuth с расширенными возможностями выдачи и отзыва токенов (часто используется в связке с Auth.js для входа через сторонние сервисы).
- MVC (Model-View-Controller) — архитектурный паттерн разделения

					ОПРЕДЕЛЕНИЯ, СОКРАЩЕНИЯ И ОБОЗНАЧЕНИЯ	Лист
						3
Изм.	Лист	№ докум.	Подп.	Дата		

приложения на модель, представление и контроллер; многие упомянутые фреймворки опираются на похожие принципы.

- 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.
- OAuth2 — использован для интеграции входа через аккаунты сторонних сервисов, таких как Google, Facebook и др.
- HTTP-only cookie — тип cookie, недоступный через JavaScript, используется для безопасного хранения JWT и защиты от XSS.
- DeepSeek — облачный AI-сервис для анализа кода лабораторных работ и выдачи рекомендаций по исправлению ошибок.
- FSD-среэ (feature slice) — логически обособленный фрагмент архитектуры Feature-Sliced Design, который содержит все уровни: UI, бизнес-логику и взаимодействие с API по одной функции.
- Widget — компонент верхнего уровня в архитектуре FSD, объединяющий в себе несколько «фич» или отдельных элементов пользовательского интерфейса.
- Feature — отдельный модуль бизнес-логики в архитектуре FSD, отвечающий за конкретную функциональность приложения.
- Entity — модель предметной области, слой взаимодействия с API и определения типов данных в архитектуре FSD.
- Shared — общий слой в архитектуре FSD, содержащий утилиты, константы, типы и переиспользуемые компоненты.
- CRUD-операции — базовые действия (создание, чтение, обновление, удаление) с ресурсо-ориентированными данными на клиенте и сервере.

					ОПРЕДЕЛЕНИЯ, СОКРАЩЕНИЯ И ОБОЗНАЧЕНИЯ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		4

- UUID (Universally Unique Identifier) — универсальный уникальный идентификатор, используемый для генерации временных или локальных идентификаторов (например, при оптимистичной отправке сообщений).
- Optimistic UI — техника обновления пользовательского интерфейса до получения подтверждения от сервера, чтобы пользователь видел мгновенный отклик.

					ОПРЕДЕЛЕНИЯ, СОКРАЩЕНИЯ И ОБОЗНАЧЕНИЯ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		5

Содержание

ОПРЕДЕЛЕНИЯ, СОКРАЩЕНИЯ И ОБОЗНАЧЕНИЯ	1
Введение	10
1 ОБЗОР И ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	13
1.1 Введение в предметную область	13
1.2 Анализ существующих образовательных платформ	14
1.2.1 Moodle	14
1.2.2 Google Classroom	14
1.2.3 Microsoft Teams for Education	15
1.2.4 Платформы для анализа кода	15
1.3 Проблемы существующих решений	15
1.4 Потребности образовательной среды	16
1.4.1 Доступность материалов и заданий	16
1.4.2 Автоматизация проверок и оценки	16
1.4.3 Удобная система заданий и общения	16
1.4.4 Интеграция всех процессов в одну систему	16
1.4.5 Вывод	17
1.5 Технологии разработки клиентской части приложения	17
1.5.1 TypeScript	17
1.5.2 React	18
1.5.3 Angular	19
1.5.4 Vue.js	20
1.5.5 Сравнительный анализ фреймворков	21
1.6 Требования к функциональности приложения	28
1.6.1 Регистрация и структура университетов	28
1.6.2 Панель преподавателя	28

					ОПРЕДЕЛЕНИЯ, СОКРАЩЕНИЯ И ОБОЗНАЧЕНИЯ			
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.					6	93
Консул.						БГТУ им. В.Г. Шухова, ПВ-212		
Н. контр.		Н.Кнтр. И.О.						
Зав. Каф.		Поляков В.М.						

1.6.3	Панель студента	28
1.7	Диаграмма вариантов использования	29
1.8	Коммуникация и взаимодействие	29
1.9	Безопасность данных	30
1.9.1	Авторизация с использованием JWT и Auth.js	30
1.9.2	Роль Auth.js	31
1.9.3	Меры защиты	31
1.10	Система создания заданий с AI-анализом кода	32
1.10.1	Общее описание модуля	32
1.10.2	Функциональные возможности	32
1.10.3	AI-анализ кода	33
1.11	ИИ-анализ кода на основе DeepSeek	33
1.11.1	Общее описание модуля DeepSeek	33
1.11.2	Функциональные возможности DeepSeek	34
1.11.3	Принцип работы DeepSeek	34
1.12	Тестирование с использованием Jest	35
1.13	Выводы	36
2	ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОЕКТА	37
2.1	Архитектура клиентской части системы	37
2.1.1	Общая структура архитектуры	37
2.1.2	Архитектурная диаграмма	38
2.1.3	Слои архитектуры Feature-Sliced Design	39
2.1.4	Концепция срезов (slices)	39
2.1.5	Горизонтальное деление на сегменты (segments)	40
2.2	Проектирование интерфейсных подсистем и экранов	41
2.2.1	Выделение ключевых интерфейсных подсистем	42
2.2.2	Страницы и их структура	45
2.2.3	Компоненты и принципы их структурирования	48
2.2.4	Распределение логики по слоям архитектуры	49
2.2.5	UX-решения и пользовательские сценарии	50
2.2.6	Вывод	51
2.3	Проектирование взаимодействия с сервером и WebSocket	51

2.3.1	Аутентификация и управление токенами	51
2.3.2	Унифицированная функция отправки запросов	52
2.3.3	Взаимодействие через WebSocket	53
2.3.4	Вывод	54
2.4	Интеграция ИИ-модуля DeepSeek в архитектуру проекта	55
2.4.1	Контейнеризация DeepSeek	55
2.4.2	Взаимодействие клиентской части с DeepSeek	55
2.4.3	Преимущества контейнеризированного подхода	56
2.4.4	Вывод	56
2.5	Обеспечение безопасности клиентской части	56
2.5.1	Механизм разграничения доступа с использованием Middleware	57
2.5.2	Роль и защита при работе с форматируемым текстом . .	58
2.5.3	Вывод	58
2.6	Покрывтие бизнес-логики юнит-тестами	59
3	ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	61
3.1	Архитектура по Feature-Sliced Design	61
3.1.1	Слой shared	61
3.1.2	Слой entities	62
3.1.3	Слои features, widgets, pages	62
3.1.4	Пример: RegistrationUniversity	63
3.1.5	Соглашения по неймингу и структуре	64
3.2	Собственная UI-библиотека и генерация форм	65
3.2.1	Общая идея и мотивация	65
3.2.2	Компонент FormBuilder : концепция и API	65
3.2.3	Типы элементов схемы и их поведение	67
3.2.4	Преимущества и выводы	69
3.2.5	JWT-callback и обновление токенов	70
3.3	Модуль «Регистрация и вход»	73
3.3.1	Модуль формирования и отправки приглашений	76
3.3.2	Последовательная диаграмма работы модуля Classrooms	78
3.3.3	Последовательная диаграмма работы модуля Chats . . .	81

ЗАКЛЮЧЕНИЕ	85
СПИСОК ЛИТЕРАТУРЫ	88
ПРИЛОЖЕНИЕ А	90
ПРИЛОЖЕНИЕ Б	93

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 приложения, которое объединяет функции чатов, проведения занятий и автоматического анализа решений, используя возможности ИИ. На данный момент отсутствует единое решение, которое бы эффективно сочетало в себе эти ключевые аспекты: возможность общения через чаты, создание заданий и автоматизированную 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 объединило бы эти элементы, позволяет улучшить качество образовательного процесса, повысив продуктивность студентов и преподавателей, а также упростив взаимодействие и автоматизировав многие рутинные задачи.

Целью данной работы является улучшение и упрощение опыта работы преподавателей и процесса обучения студентов посредством разработки клиентской части интегрированного образовательного приложения, обеспечиваю-

					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.					10	93
Руковод.		Мельников А.Б.				БГТУ им. В.Г. Шухова, ПВ-212		
Консул.								
Н. контр.		Н.Кнтр. И.О.						
Зав. Каф.		Поляков В.М.						

щего эффективное взаимодействие, автоматизированную проверку решений и удобные средства коммуникации.

Для достижения поставленной цели необходимо решить следующие задачи:

- а) Проанализировать предметную область и существующие системы, выявив их сильные и слабые стороны.
- б) Определить архитектурные и технологические решения, подходящие для реализации клиентской части приложения.
- в) Спроектировать пользовательский интерфейс, обеспечивающий интуитивное и удобное взаимодействие для преподавателей и студентов.
- г) Разработать компоненты для управления учебными структурами (институт, кафедра, группа), заданиями и чатами.
- д) Интегрировать средства для автоматизированной проверки решений студентов с применением ИИ.
- е) Реализовать тестирование бизнес-логики приложения для обеспечения её корректности и эффективности.

Структура пояснительной записки включает следующие разделы:

- В первом разделе рассматриваются особенности предметной области, проводится анализ существующих решений и обоснование выбора технологий и методов проектирования. Приводится обзор существующих образовательных приложений и их недостатков, а также объясняется необходимость разработки интегрированного решения.
- Во втором разделе описывается архитектура клиентской части приложения, структура пользовательского интерфейса, проектирование компонентов и их взаимодействие. Рассматриваются решения для реализации системы чатов, создания и проверки заданий, а также интеграции ИИ-анализа.

					Введение	Лист
						11
Изм.	Лист	№ докум.	Подп.	Дата		

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

					Введение	Лист
						12
Изм.	Лист	№ докум.	Подп.	Дата		

1 ОБЗОР И ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Введение в предметную область

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

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

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

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

					ОБЗОР И ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ			
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.					13	93
Руковод.		Мельников А.Б.				БГТУ им. В.Г. Шухова, ПВ-212		
Консул.								
Н. контр.		Н.Кнтр. И.О.						
Зав. Каф.		Поляков В.М.						

Интеграция всех этих функций в одну платформу позволит значительно упростить организацию учебного процесса, улучшить взаимодействие между преподавателями и студентами, а также повысить качество обучения за счет автоматизации рутинных задач.

1.2 Анализ существующих образовательных платформ

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

1.2.1 Moodle

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

1.2.2 Google Classroom

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

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

1.2.3 Microsoft Teams for Education

Microsoft Teams, в отличие от Moodle и Google Classroom, активно используется для организации видеоконференций и групповых чатов. Он позволяет преподавателям и студентам взаимодействовать в реальном времени, а также интегрирует различные сервисы Microsoft 365. Однако, как и в случае с другими платформами, Microsoft Teams не предоставляет функционала для интегрированного анализа кода студентов с использованием искусственного интеллекта, что ограничивает его возможности в обучении программированию.

1.2.4 Платформы для анализа кода

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

1.3 Проблемы существующих решений

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

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

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

1.4 Потребности образовательной среды

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

1.4.1 Доступность материалов и заданий

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

1.4.2 Автоматизация проверок и оценки

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

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, объединя-

ющая 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 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 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 типа-

ми 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.

1.5.2 React

React — библиотека для построения пользовательских интерфейсов, разработанная Facebook. 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.

Ключевые особенности:

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

Преимущества для образовательных платформ:

- Быстрая разработка за счёт декларативности и компонентного подхода
- Большое сообщество и развитая экосистема (Next.js, Redux, React Query и др.)

- Поддержка SSR и SSG при использовании Next.js
- Простая интеграция с библиотеками и сторонними сервисами

Ограничения:

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

1.5.3 Angular

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

Ключевые особенности:

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

Преимущества для образовательных платформ:

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

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

Ограничения:

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

1.5.4 Vue.js

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

Основные характеристики:

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

Сильные стороны для учебных проектов:

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

Ограничения:

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

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

Таблица 1 – Сравнение характеристик фреймворков

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

1.5.5 Сравнительный анализ фреймворков

Ключевые выводы:

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

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

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

Next.js

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

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

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

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

Next.js также упрощает настройку маршрутизации и управление данными. Встроенная система маршрутизации автоматически генерирует страницы на основе файловой структуры, что делает создание новых страниц и маршрутов простым и интуитивно понятным. Кроме того, Next.js предостав-

ляет инструменты для работы с API, что позволяет без труда интегрировать серверную логику в приложение.

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

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

Socket.IO

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

Преимущества использования Socket.IO:

- **Гибкость:** Разработчики имеют полный контроль над архитектурой соединения, включая маршрутизацию сообщений, обработку событий, систему комнат (rooms) и пространств имён (namespaces).
- **Масштабируемость:** Поддержка кластеризации и горизонтального масштабирования при помощи Redis-адаптеров.
- **Совместимость с Node.js:** Socket.IO органично интегрируется в

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

стек на основе Node.js, что упрощает реализацию единой инфраструктуры.

- **Производительность:** Низкая задержка при передаче сообщений благодаря постоянному соединению между клиентом и сервером.
- **Интеграция с Python:** Для серверной части на Python существует аналогичная библиотека *python-socketio*, которая позволяет использовать те же возможности для реализации чатов и двустороннего общения между клиентом и сервером. Это делает возможным использование Socket.IO как на front-end (с помощью Node.js), так и на back-end (с помощью Python), обеспечивая совместимость и синхронизацию данных между клиентом и сервером.

Недостатки:

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

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

Pusher

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

Преимущества использования Pusher:

- **Упрощённая интеграция:** SDK и готовые клиентские библиотеки позволяют быстро настроить соединение и передавать события.

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

- **Масштабируемость:** Обеспечивается на уровне платформы без участия разработчика.
- **Надёжность:** Pusher использует устойчивую облачную инфраструктуру с балансировкой нагрузки.
- **Аналитика и мониторинг:** Панель управления предоставляет данные о соединениях, событиях и каналах.

Ограничения:

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

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

WebSocket в JavaScript

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

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

- переподключение сокета при его падении,

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

- буферизацию сообщений в случае разрыва соединения,
- обработку таймаутов и ошибок,
- поддержку различных сред (например, Node.js и браузер),
- масштабирование на сервере.

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

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

Сравнение и выбор технологии для чатов

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

Таблица 2 – Сравнение технологий для реализации чатов в реальном времени

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

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

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

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

Auth.js

Auth.js — это библиотека для реализации аутентификации и авторизации в веб-приложениях. Она является официальным решением, рекомендуемым и поддерживаемым фреймворком Next.js, что гарантирует хорошую интеграцию и поддержку всех необходимых функций. Библиотека позволяет легко подключать сторонние провайдеры аутентификации, такие как Google, Facebook и другие, а также реализовывать собственную аутентификацию с использованием базы данных. Auth.js обеспечивает 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.

Redux

Redux — это библиотека для управления состоянием в приложениях, основанных на React. Она используется для централизованного хранения состояния приложения, что облегчает обмен данными между компонентами и упрощает их взаимодействие. Redux помогает избежать ”проблемы пропс-дерева” в больших приложениях, когда передача данных через множество вложенных компонентов становится сложной. Хотя Redux часто используется в более сложных приложениях, в данном проекте его роль заключается в том, чтобы сделать взаимодействие между компонентами более организованным и улучшить предсказуемость состояния приложения.

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

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

1.6.3 Панель студента

Студенты, присоединенные к определенным группам, имеют доступ к:

- групповым чатам своей учебной группы;
- личной переписке с преподавателями;
- материалам, отправленным преподавателями;

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

- 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 AI;
- **Студент** — участие в групповых и личных чатах, выполнение заданий и просмотр результатов;
- **Администратор** — управление институтами, кафедрами, группами, студентами и преподавателями, а также создание приглашений в систему.

1.8 Коммуникация и взаимодействие

Ключевым элементом платформы является система обмена сообщениями, включающая:

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

- Передача файлов тех же форматов, что и в групповых чатах

Обе системы поддерживают базовые функции:

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

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

1.9 Безопасность данных

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:

а) Аутентификация пользователя:

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

б) Работа с токеном:

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

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

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

в) Доступ к защищённым ресурсам:

- При обращении к API, front-end автоматически прикрепляет токен к запросу.
- При недействительном или истёкшем токене, Auth.js может автоматически обновить его через refresh-токен, если он присутствует.

1.9.2 Роль Auth.js

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

а) Инкапсуляция логики:

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

б) Поддержка современных стандартов:

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

1.9.3 Меры защиты

Для повышения безопасности механизма авторизации реализованы дополнительные меры:

а) JWT:

					ОБЗОР И ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	Лист
						31
Изм.	Лист	№ докум.	Подп.	Дата		

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

б) Auth.js:

- Валидирует параметры входа, включая redirect URI.

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

1.10 Система создания заданий с AI-анализом кода

1.10.1 Общее описание модуля

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

Основные функции:

- Создание виртуального класса преподавателем
- Назначение заданий с параметрами оценки
- Загрузка решений студентами
- Получение отчётов об автоматическом AI-анализе кода
- Просмотр статистики и аналитики преподавателем

1.10.2 Функциональные возможности

Для преподавателей:

- Создание и управление классами
- Назначение заданий
- Просмотр AI-отчётов по каждому студенту
- Сводная статистика по группе

					ОБЗОР И ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		32

Для студентов:

- Просмотр активных заданий и дедлайнов
- Загрузка решения

1.10.3 AI-анализ кода

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

На основе заданных преподавателем критериев формируется интерактивный отчёт, включающий:

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

1.11 ИИ-анализ кода на основе DeepSeek

1.11.1 Общее описание модуля DeepSeek

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

Платформа DeepSeek способна:

- Выявлять самые разнообразные синтаксические и логические ошибки, обнаруживать неточности в реализации алгоритмов;
- Оценивать соответствие структурных блоков кода требованиям конкретного задания, обращая внимание на правильность использования функций и корректность их связи;
- Фиксировать стилевые отклонения и «code smells», которые могут усложнить поддержку и дальнейшее развитие проекта;

					ОБЗОР И ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		33

- Анализировать степень оригинальности решения с помощью семантического сравнения embedding-представлений, что позволяет не только обнаружить плагиат, но и оценить творческий подход студента.

Доступ к вызовам DeepSeek строго ограничен: только преподаватель, работая в защищённом интерфейсе фронтенда, может инициировать анализ кода. Студенты видят лишь результат — отчёт с пояснениями и рекомендациями — без какой-либо информации о внутреннем устройстве сервиса.

1.11.2 Функциональные возможности DeepSeek

Для преподавателей:

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

Для студентов:

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

1.11.3 Принцип работы DeepSeek

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

					ОБЗОР И ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		34

- б) **Семантический анализ:** алгоритмы трансформеров сопоставляют логику решения с огромной базой примеров, выявляя нетривиальные отклонения от оптимальной структуры.
- в) **Расчет метрик:** рассчитываются показатели цикломатической сложности, глубины вложенности, соответствия coding standard и других параметров качества.
- г) **Антиплагиатный компонент:** сравнение embedding-векторов загруженного решения с репозиторием эталонных и ранее проверенных работ для определения степени оригинальности.
- д) **Формирование отчёта:** на выходе генерируется детальный JSON-документ, содержащий список найденных замечаний, метрик и чётких рекомендаций по улучшению.

1.12 Тестирование с использованием Jest

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

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

- **Snapshot-тесты.** Возможность сохранять «снимки» выходных данных функций или компонентов и автоматически отслеживать их изменения с течением времени.

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

1.13 Выводы

В результате проведённого анализа можно сделать вывод о наличии устойчивого запроса на интегрированное образовательное приложение, способное решать сразу несколько ключевых задач. Современные платформы зачастую фокусируются либо на предоставлении учебных материалов, либо на коммуникации, либо на автоматизации проверки знаний, при этом разрозненность этих функций создаёт неудобства для всех участников образовательного процесса.

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

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

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

2 ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОЕКТА

2.1 Архитектура клиентской части системы

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

2.1.1 Общая структура архитектуры

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

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

					ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОЕКТА			
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.					37	93
Консул.						БГТУ им. В.Г. Шухова, ПВ-212		
Н. контр.		Н.Кнтр. И.О.						
Зав. Каф.		Поляков В.М.						

2.1.2 Архитектурная диаграмма

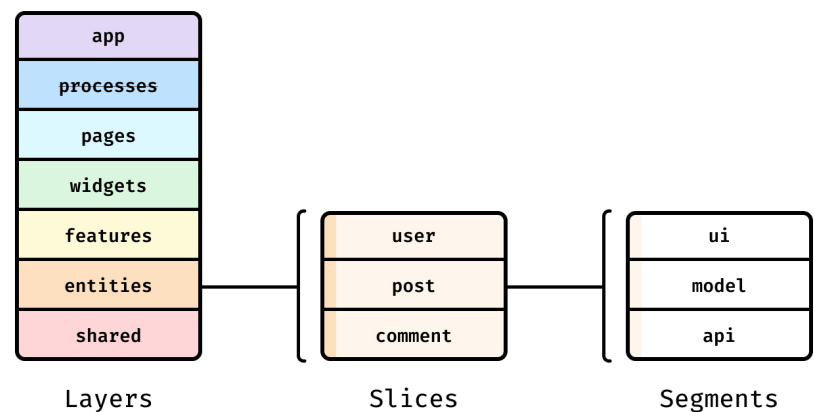


Рисунок 2.1 – Схема архитектуры клиентской части (FSD + Next.js)

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

2.1.3 Слои архитектуры Feature-Sliced Design

Таблица 3 – Слои архитектуры Feature-Sliced Design

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

2.1.4 Концепция срезов (slices)

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

Срезы группируются по смысловому признаку и могут располагаться в рамках различных уровней архитектуры: *features*, *entities*, *widgets*, *pages*. Такое структурирование обеспечивает лучшую декомпозицию кода,

повышает его читабельность и облегчает модульное тестирование.

Например:

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

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

2.1.5 Горизонтальное деление на сегменты (segments)

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

Наиболее распространённые типы сегментов включают:

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

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

- **config** — конфигурационные файлы и переключатели функциональности (feature flags).

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

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.

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

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**.

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.

а) Подсистема авторизации и регистрации

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

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

- Первый сценарий реализован для новых организаций (институтов) и сопровождается созданием административной учётной записи. На этом этапе формируется корневая структура управления учреждением.
- Второй и третий сценарии предназначены для регистрации преподавателей и студентов соответственно. Оба сценария

доступны исключительно по индивидуальным приглашениям, что обеспечивает контроль над составом участников образовательного процесса и предотвращает несанкционированный доступ.

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

б) Подсистема управления университетом

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

Основными функциями данной подсистемы являются:

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

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

в) Подсистема работы с заданиями и отправкой решений

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

Основной интерфейс включает:

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

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

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

г) Подсистема обмена сообщениями (чаты)

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

Основной функционал включает:

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

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

д) Подсистема AI-анализа решений

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

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

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

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

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

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

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

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

компонентами пользовательского интерфейса, бизнес-логикой и хранилищем состояния.

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

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

– **Страница авторизации**

Отвечает за вход пользователя в систему. Содержит форму для ввода учётных данных, а также реализует логику валидации, передачи данных на сервер, обработки ошибок и сохранения сессионного токена. После успешной авторизации пользователь перенаправляется на главную страницу, соответствующую его роли.

– **Страница заданий**

Представляет собой ключевой интерфейс для организации и выполнения учебной деятельности. Интерфейс страницы включает:

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

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

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

– **Административная панель института**

Данная страница является основным рабочим инструментом пользователя с ролью администратора. Интерфейс включает:

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

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

– **Страница чатов**

Реализует коммуникационную составляющую платформы. Пользователь получает доступ к:

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

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

Все функциональные страницы приложения, за исключением экранов регистрации и входа, используют единый шаблон компоновки **AppLayout**,

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

обеспечивающий целостность визуального восприятия и унификацию пользовательского опыта. Данный шаблон включает в себя общие элементы интерфейса — верхнюю панель навигации, боковое меню и основной контейнер для отображения содержимого, который динамически наполняется в зависимости от текущего маршрута.

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

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

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 при-

кладной 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 **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, включающий боковую и верхнюю панели, интерфейс остаётся единообразным и интуитивно понятным вне зависимости от текущего маршрута.
- **Toast-уведомления** — реализация мгновенной обратной связи при выполнении действий: успешная отправка формы, ошибка сети, получение новых сообщений.
- **Обработка пустых состояний и ошибок** — предусмотрены интерфейсы для ситуаций отсутствия данных, ошибок загрузки или недоступности сервера.

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

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

2.2.6 Вывод

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

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

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

2.3 Проектирование взаимодействия с сервером и WebSocket

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

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

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

хранения токенов.

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

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

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

2.3.2 Унифицированная функция отправки запросов

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

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

					ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОЕКТА	Лист
Изм.	Лист	№ докум.	Подп.	Дата		52

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

2.3.3 Взаимодействие через WebSocket

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

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

- Поддержка автоматического переподключения при обрыве соединения;
- Передача структурированных событий с именами и аргументами;
- Интеграция с механизмами авторизации и middleware;
- Гибкость при работе с пространствами имён и комнатами;
- Совместимость с fallback-транспортирами (в случае недоступности WebSocket).

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

- Инициализацию соединения с сервером по заданному адресу;
- Отправку и приём событий с типизированной структурой данных;
- Автоматическую подписку и отписку от необходимых каналов;
- Управление жизненным циклом подключения;

– Обработку ошибок и логирование сетевых событий.

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

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

- а) Иницируется обновление токена через заранее определённый механизм;
- б) Закрывается текущее соединение;
- в) После получения нового токена создаётся новое подключение с обновлёнными параметрами.

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

2.3.4 Вывод

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

					ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОЕКТА	Лист
Изм.	Лист	№ докум.	Подп.	Дата		54

2.4 Интеграция ИИ-модуля DeepSeek в архитектуру проекта

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.

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

2.4.2 Взаимодействие клиентской части с DeepSeek

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

- Преподаватель выбирает работу студента и нажатием кнопки инициирует анализ.
- Фронтенд направляет корректно сформированный запрос к API, где указаны идентификатор работы и необходимые параметры оценки.
- По готовности отчёта фронтенд периодически проверяет статус анализа и загружает результаты в привычном формате.

					ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОЕКТА	Лист
Изм.	Лист	№ докум.	Подп.	Дата		55

- Все шаги интегрированы в единый пользовательский поток: преподаватель не видит инфраструктуры контейнеров, а получает только готовый отчёт.

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

2.4.3 Преимущества контейнеризированного подхода

Контейнеризация DeepSeek даёт следующие ключевые плюсы:

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

2.4.4 Вывод

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

2.5 Обеспечение безопасности клиентской части

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

					ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОЕКТА	Лист
Изм.	Лист	№ докум.	Подп.	Дата		56

2.5.1 Механизм разграничения доступа с использованием Middleware

Одним из ключевых компонентов обеспечения безопасности клиентской части является система контроля доступа на основе промежуточного слоя — `middleware`. В рамках архитектуры `Next.js`, `middleware` представляет собой функцию, исполняемую при каждом запросе к защищённым маршрутам. Она позволяет перехватывать обращения к страницам до их рендеринга и на этой стадии выполнять необходимые проверки: наличие токена, его валидность, а также права пользователя.

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

На основе этой информации `middleware` выполняет следующие действия:

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

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

2.5.2 Роль и защита при работе с форматировемым текстом

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

Для решения данной задачи в проекте используется специализированная библиотека **tiptap** — расширяемый редактор форматированного текста, основанный на **ProseMirror**. Одним из ключевых преимуществ **tiptap** является контроль над тем, какие HTML-теги и атрибуты допускаются к интерпретации и отображению. Таким образом, даже если пользователь попытается вставить опасный HTML-код (например, `<script>` или инъекцию с обработчиком событий), редактор проигнорирует или удалит такие элементы на этапе парсинга.

Технически это реализуется следующим образом:

- При вводе текстов редактор не сохраняет «сырые» HTML-строки, а формирует безопасное представление контента на основе строго описанных схем;
- При рендеринге текста из базы или состояния редактор отображает только те элементы, которые были описаны как допустимые;
- Расширения (extensions), добавляемые к **tiptap**, позволяют точно контролировать список разрешённых действий и структур (например, разрешить только ``, ``, `` и `<code>`).

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

2.5.3 Вывод

Комплекс реализованных решений позволяет эффективно защитить клиентскую часть приложения как от внешнего вмешательства, так и от оши-

					ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОЕКТА	Лист
Изм.	Лист	№ докум.	Подп.	Дата		58

бочного доступа со стороны пользователей. Механизм `middleware` обеспечивает надёжную проверку подлинности сессии и прав доступа до загрузки страниц, а редактор `tiptap` гарантирует безопасность при работе с форматированным текстом. Такое сочетание архитектурных и прикладных решений способствует созданию устойчивой и безопасной пользовательской среды.

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

					ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОЕКТА	Лист
Изм.	Лист	№ докум.	Подп.	Дата		59

даёт уверенность в корректности отображения без сложных технических средств.

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

3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

3.1 Архитектура по Feature-Sliced Design

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

3.1.1 Слой **shared**

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

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

					ПРОГРАММНАЯ РЕАЛИЗАЦИЯ			
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.					61	93
Консул.						БГТУ им. В.Г. Шухова, ПВ-212		
Н. контр.		Н.Кнтр. И.О.						
Зав. Каф.		Поляков В.М.						

- **SVG-иконки и графика:** импорт через SVGR для единообразного подключения и управления атрибутами SVG;
- **Компоненты навигации и управления состоянием:** `PaginationComponent` для управления пагинацией через URL-параметры, `Breadcrumbs`, `Tabs` и т. д.

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

3.1.2 Слой **entities**

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

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

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

3.1.3 Слои **features, widgets, pages**

Главные рабочие слои приложения, отвечающие за реализацию конкретной функциональности:

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

- а) **pages**: маршрутизация и верхний уровень страницы. Здесь описываются пути, разрешения доступа (guards), асинхронная загрузка данных на уровне страницы и выбор необходимого набора виджетов;
- б) **widgets**: презентационные и «умные» компоненты, разделённые по Smart/Presentational-паттерну:
- *Presentational*: исключительно UI и пропсы, без прямой работы с API и глобальным состоянием;
 - *Smart*: обёртки, получающие данные через хуки из **features**, передающие результат в презентационные компоненты;
- в) **features**: бизнес-логика и состояние, реализованное в виде кастомных хуков, функций-редьюсеров, слайсов Redux или Zustand:
- **hooks.ts**: главный хук-фабрика (например, `useRegistrationUniversity`);
 - **schema.ts**: схемы валидации через Yup или Zod;
 - **utils.ts**: вспомогательные функции и вычисляемые селекторы;
 - **store.ts**: подключение к Redux/Redux Toolkit или настройка Zustand;

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

3.1.4 Пример: RegistrationUniversity

Листинг 1: RegistrationUniversityWidget

```
1 // Presentationкомпонент-
```

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

```

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

```

Листинг 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 неймингу и структуре

Для поддержания высокого уровня единого стиля и предсказуемости структуры проекта:

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

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

3.2 Собственная UI-библиотека и генерация форм

3.2.1 Общая идея и мотивация

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

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

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

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:

- Пользователь задаёт схему параметров формы, представляющую собой массив объектов с полем `type` и соответствующим набором `props`.
- `FormBuilder` инициализирует внутреннее состояние формы и передаёт каждому полю текущие значения и функции обработки изменений.

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

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

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

- **type**: идентификатор типа элемента (например, **input_field**, **array_fields**);
- **props**: набор свойств, необходимых для рендеринга и обработки (имя поля, текст метки, дополнительные параметры).

Листинг 3: Пример описания схемы формы

```
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 }
```

Листинг 4: Использование **FormBuilder**

```
1 <FormBuilder schema={inviteTeacherScheme()} 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.

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

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

Листинг 5: Пример INPUT_FIELD_SCHEMA

```
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 };
```

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

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

Листинг 6: Пример ARRAY_FIELDS_SCHEMA

```
1 const schema: ARRAY_FIELDS_SCHEMA = [
```

					ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	Лист
						67
Изм.	Лист	№ докум.	Подп.	Дата		

```

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 ];

```

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

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

Листинг 7: Пример FORM_WRAPPER_SCHEMA

```

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 }];

```

BLOCK_WRAPPER_SCHEMA Тип `block_wrapper` позволяет визуаль-но группировать элементы без изменения структуры данных: поля в нём обра-батываются как часть текущего массива или объекта.

Листинг 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 ];
```

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

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

- Консистентность API при описании разных типов полей;

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

3.2.5 JWT-callback и обновление токенов

Для передачи и обновления JWT в сессии используется callback `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 неизменённая структура. Пример реализации:

Листинг 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, по-

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

лучение 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.

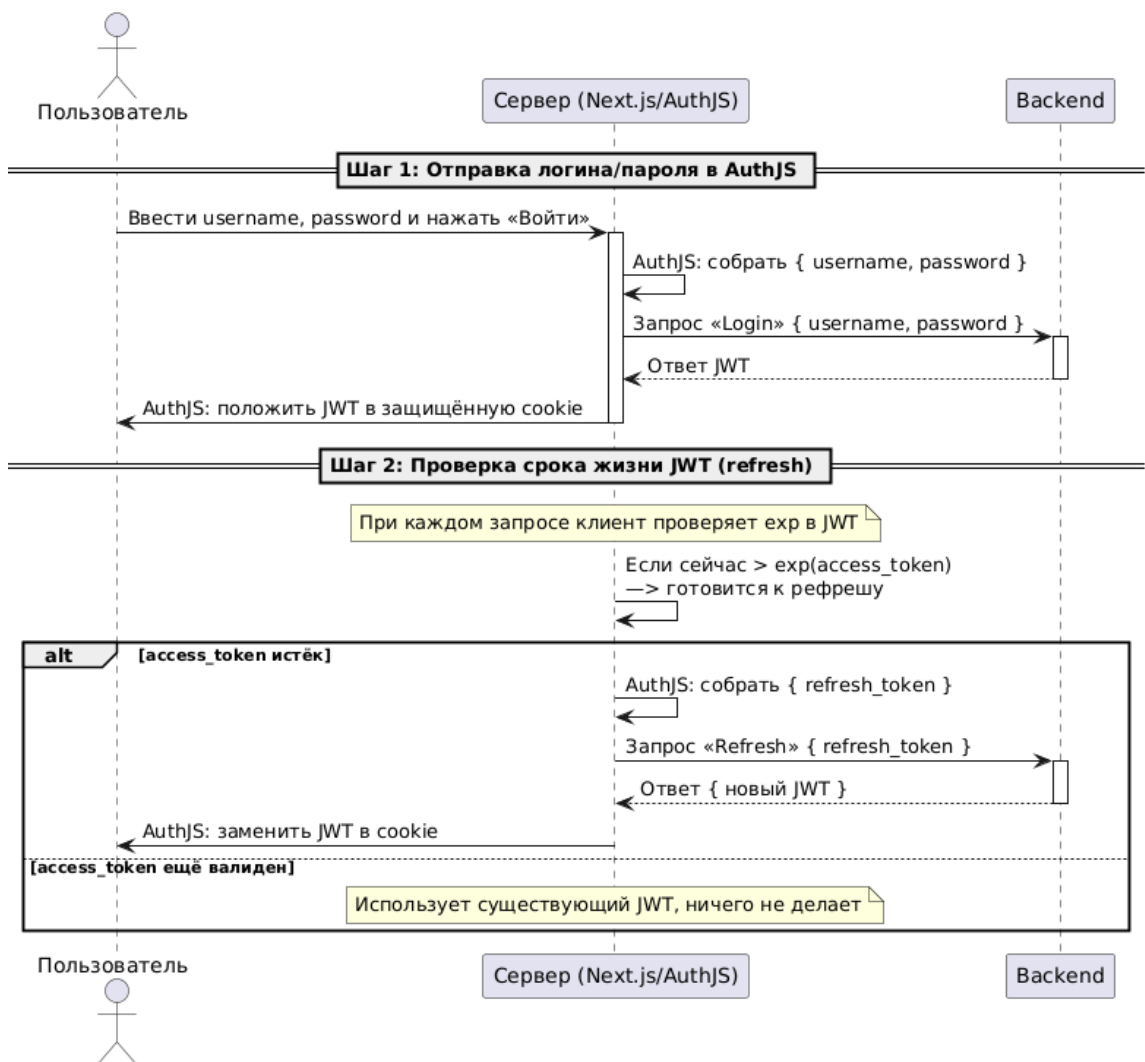


Рисунок 3.1 – Схема процесса проверки и обновления JWT через refresh-токен

На рисунке 3.1 можно выделить два основных этапа:

- а) **Первичный вход (signIn):**
- Пользователь вводит `username` и `password` в приложение.
 - Auth.js формирует объект с учётными данными и отправляет запрос на бэкенд (метод `login`).

- Бэкенд возвращает пару токенов: `access_token` и `refresh_token`.
- Auth.js сохраняет полученный JWT в защищённую cookie (или в хранилище клиента).

б) Проверка срока жизни и refresh:

- При каждом запросе клиент проверяет поле `exp` (срок жизни) в `access_token`.
- Если текущий момент времени превысил `exp(access_token)`, начинается подготовка к рефрешу.
- 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.
- 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.

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

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 ниже:

Листинг 11: Механизм предотвращения race condition при рефреше токена

```

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

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

Листинг 12: Выбор виджета регистрации

```

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

```

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

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.

Листинг 13: RegistrationTeacherWidget

					ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		74

```

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 }

```

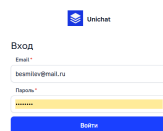


Рисунок 3.3 – Экран входа в систему: поля для ввода Email и пароля, а также кнопка «Войти».

На рисунке 3.3 представлен экран входа в систему. Для авторизации пользователь вводит Email и пароль, после чего нажимает кнопку «Войти». В успешном случае система перенаправляет его в личный кабинет; при некорректных учётных данных отображается сообщение об ошибке.

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

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:

а) 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».

б) 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.

в) 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.

					ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		76

Ниже 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.

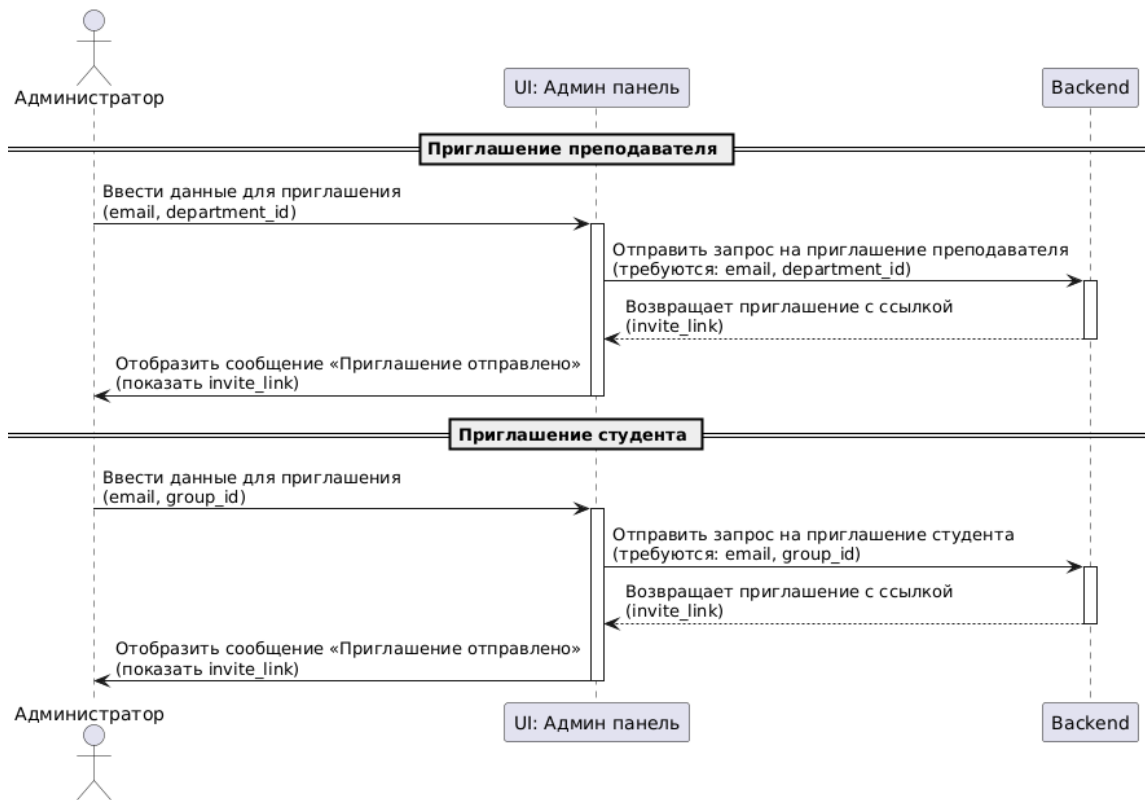


Рисунок 3.4 – Схема процесса формирования и отправки приглашений преподавателям и студентам

На рисунке 3.4 можно выделить следующие ключевые этапы:

– **Приглашение преподавателя:**

- а) Администратор вводит в UI email и department_id.
- б) UI отправляет запрос на бэкенд «Приглашение преподавателя (требуется: email, department_id)».
- в) Бэкенд проверяет данные, создаёт приглашение и возвращает уникальную ссылку (invite_link).
- г) UI отображает сообщение «Приглашение отправлено» и показывает invite_link.

– **Приглашение студента:**

- а) Администратор вводит в UI email и group_id.
- б) UI отправляет запрос на бэкенд «Приглашение студента (требуется: email, group_id)».
- в) Бэкенд проверяет данные, создаёт приглашение и возвращает уникальную ссылку (invite_link).
- г) 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, UI, бэкенда и AI-сервиса 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 через AI и выставление оценки преподавателем.

На рисунке А.3 можно выделить следующие ключевые этапы:

а) 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».

					ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		78

б) Преподаватель создаёт задание с возможностью задать промт для AI-проверки:

- Преподаватель переходит в форму создания задания, указывая вместе с условием текста задания промт для AI-проверки.
- UI отправляет запрос на бэкенд с данными задания и промтом.
- Бэкенд возвращает подтверждение успешного сохранения задания.
- UI отображает преподавателю сообщение «Задание создано».

в) 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.

г) 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».

					ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		79

д) Автопроверка через AI:

- Бэкенд получает файлы решения и ранее заданный промт к заданию.
- Бэкенд отправляет файлы и промт во внешний AI-сервис.
- AI-сервис выполняет анализ кода (например, проверку корректности, стилизовых нарушений и т. д.) и возвращает результат вместе с комментариями.
- Бэкенд сохраняет результат автопроверки и передаёт его UI обоим ролям:
 - UI Студента: отображается результат автопроверки (оценка AI, комментарии).
 - UI Преподавателя: отображается результат автопроверки (для последующей ручной проверки и выставления итоговой оценки).

е) 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.

					ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		80

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:

а) Открытие приложения:

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

б) 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.

в) Отправка нового сообщения:

					ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		81

- 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.

г) 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.
- 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.

д) 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.

					ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		82

- 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.

е) 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»).

ж) 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.
- Компонент проверяет поле `exp(access_token)` локально. При истечении:
 - 1) UI «Чат» отправляет `refresh_token` на AuthJS/Next.js, получая новый `access_token`.
 - 2) После получения нового токена выполняется повторное подключение WebSocket, передавая обновлённый JWT.

					ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		83

3) 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.

					ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		84

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:

- а) Проанализирована предметная область и существующие образовательные системы, выявлены их сильные и слабые стороны.
- б) Определены архитектурные и технологические решения (Next.js, React, TypeScript, Redux, Auth.js, Socket.IO) для реализации гибкой, надёжной и удобной клиентской части приложения.
- в) Спроектирован пользовательский интерфейс, обеспечивающий интуитивное и удобное взаимодействие для преподавателей и студентов (навигация, адаптивная вёрстка, единые UI-компоненты).
- г) Разработаны компоненты для управления учебными структурами (институты, кафедры, группы), заданиями и чатами, включая административную панель, что упростило работу преподавателей при ведении учебного процесса.
- д) Интегрированы средства автоматизированной проверки решений студентов с применением ИИ (DeepSeek) в модуле виртуальных классов, что повысило качество и скорость проверки лабораторных работ.
- е) Реализовано тестирование бизнес-логики клиентского приложения с

					ЗАКЛЮЧЕНИЕ			
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.					85	93
Консул.						БГТУ им. В.Г. Шухова, ПВ-212		
Н. контр.		Н.Кнтр. И.О.						
Зав. Каф.		Поляков В.М.						

использованием Jest, что подтвердило корректность и стабильность основных функций.

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, что упрощает работу административного персонала при настройке доступа;
- 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 возможностей AI-анализа кода с поддержкой дополнительных языков программирования и более глубокой семантической проверки, чтобы преподаватели могли быстрее и точнее оценивать работы студентов;

					ЗАКЛЮЧЕНИЕ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		86

- 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;
- 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			
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.					87	93
Консул.						БГТУ им. В.Г. Шухова, ПВ-212		
Н. контр.		Н.Кнтр. И.О.						
Зав. Каф.		Поляков В.М.						

СПИСОК ЛИТЕРАТУРЫ

tocsection

1. React. Официальная документация. URL: <https://reactjs.org/>.
2. Next.js. Официальная документация. URL: <https://nextjs.org/docs>.
3. TypeScript Handbook. Официальная документация. URL: <https://www.typescriptlang.org/docs/handbook/intro.html>.
4. Auth.js (ранее NextAuth.js). Официальная документация. URL: <https://next-auth.js.org/>.
5. JSON Web Token (JWT). RFC 7519. URL: <https://tools.ietf.org/html/rfc7519>.
6. Socket.IO. Официальная документация. URL: <https://socket.io/docs/>.
7. MDN Web Docs. HTML. URL: <https://developer.mozilla.org/ru/docs/Web/HTML>.
8. MDN Web Docs. CSS. URL: <https://developer.mozilla.org/ru/docs/Web/CSS>.
9. MDN Web Docs. JavaScript. URL: <https://developer.mozilla.org/ru/docs/Web/JavaScript>.
10. Habr. «SPA: преимущества и недостатки одностраничных приложений». URL: <https://habr.com/ru/post/451234/>.
11. Habr. «SSR vs SSG в Next.js: что выбрать?» URL: <https://habr.com/ru/post/559876/>.
12. Habr. «Современные фишки React 18 и их применение». URL: <https://habr.com/ru/post/568123/>.
13. Habr. «Feature-Sliced Design: организация структуры Frontend-проекта». URL: <https://habr.com/ru/post/481567/>.

					СПИСОК ЛИТЕРАТУРЫ	Лист
						88
Изм.	Лист	№ докум.	Подп.	Дата		

14. Habr. «Управление состоянием в React с помощью Redux Toolkit». URL: <https://habr.com/ru/post/512345/>.
15. Habr. «Лёгкие альтернативы Redux: обзор Zustand». URL: <https://habr.com/ru/post/537890/>.
16. Habr. «Введение в WebSocket: что это и как работает». URL: <https://habr.com/ru/post/445678/>.
17. Habr. «Авторизация и аутентификация в Next.js с Auth.js». URL: <https://habr.com/ru/post/583210/>.
18. Habr. «Безопасность веб-приложений: защита JWT и CSRF». URL: <https://habr.com/ru/post/579432/>.
19. Habr. «SCSS-препроцессор: основы и лучшие практики». URL: <https://habr.com/ru/post/520123/>.
20. Habr. «Интеграция искусственного интеллекта в веб-приложение на примере Next.js». URL: <https://habr.com/ru/post/590456/>.

					СПИСОК ЛИТЕРАТУРЫ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		89

ПРИЛОЖЕНИЕ А

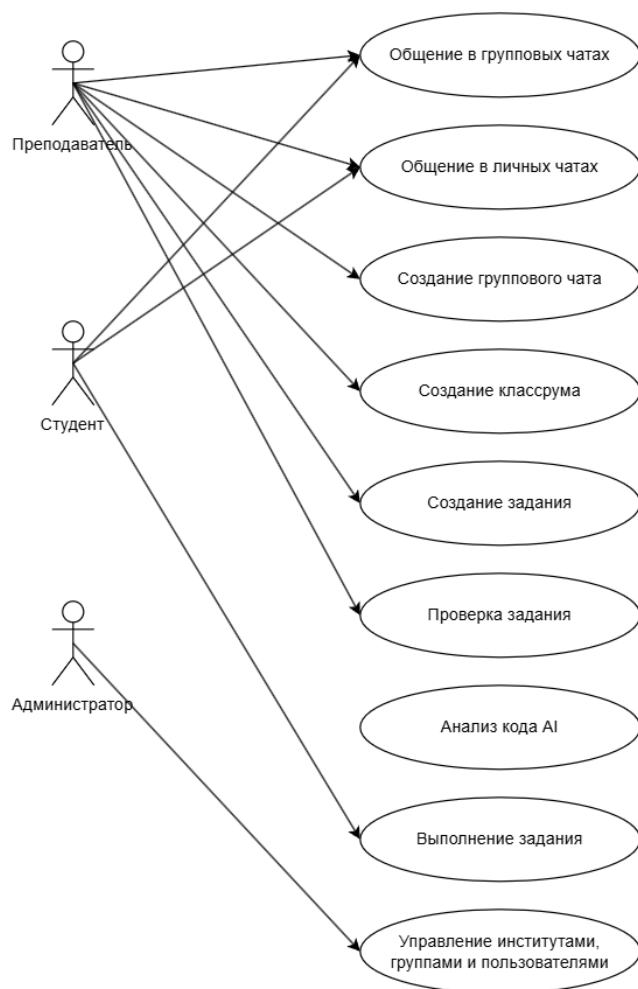


Рисунок А.1 – Диаграмма вариантов использования системы для различных ролей пользователей.

					ПРИЛОЖЕНИЕ А			
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.					90	93
Руковод.		Мельников А.Б.				БГТУ им. В.Г. Шухова, ПВ-212		
Консул.								
Н. контр.		Н.Кнтр. И.О.						
Зав. Каф.		Поляков В.М.						

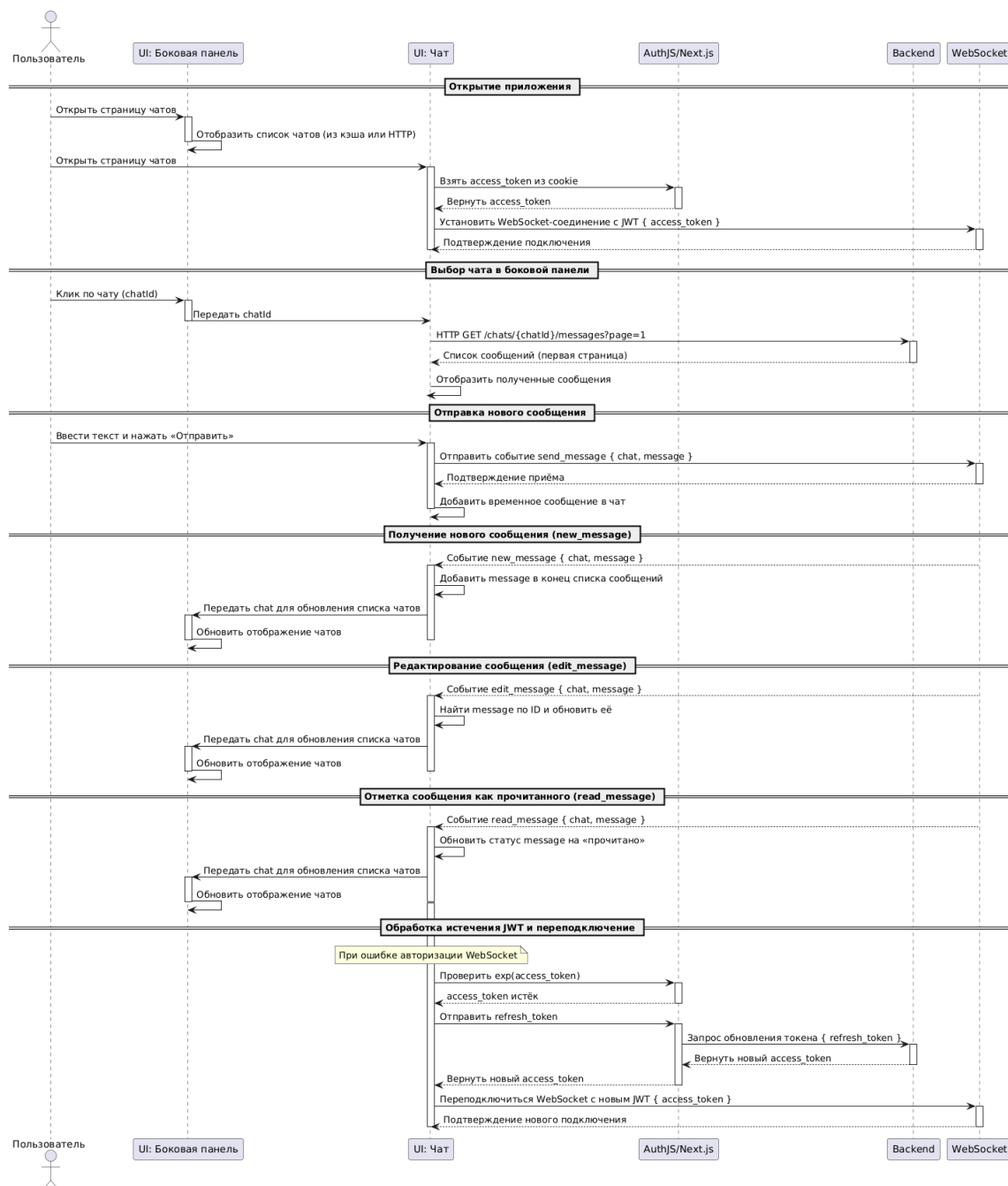


Рисунок А.2 – Схема взаимодействия клиента (UI: Боковая панель и UI: Чат), AuthJS (Next.js), бэкенда и WebSocket при работе модуля «Chats».

ПРИЛОЖЕНИЕ Б

Исходный код фронтенда доступен в GitHub-репозитории:
<https://github.com/BeSmileV/unichat-front>

					ПРИЛОЖЕНИЕ Б	Лист
						93
Изм.	Лист	№ докум.	Подп.	Дата		