

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

Сервис для управления учебными
курсами

Студент

Асянов Илгиз Касимович
(подпись, дата)

Руководитель

Кузнецов А.В.
(подпись, дата)

Нормоконтролёр

Бычков А.С.
(подпись, дата)

Работа допущена к защите (протокол зас. кафедры от 28.05.2025 г. № 16)

Заведующий кафедрой

(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой хорошо (протокол заседания ГЭК от

11 июня 2025 г. № 30)

Секретарь ГЭК

(подпись)

Е.И. Гурин
(фамилия, инициалы)

Пенза 2025

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

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
« 06 » 05 2025 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Асянов Илгиз Касимович гр. 21ВВП1 факультета вычислительной техники
2. Тема работы: Сервис для управления учебными курсами
Тема утверждена приказом ПГУ № 601/о от "06" мая 2025 г.
3. Руководитель бакалаврской работы доцент кафедры ВТ ПГУ Кузнецов А.В.
4. Задание на работу (назначение разработки, исходные данные т.п.)

Разработать «Сервис для управления учебными курсами», предназначенное для автоматизации процессов создания, управления и прохождения курсов в образовательных учреждениях. Система должна обеспечивать взаимодействие пользователей с различными ролями (преподаватель, студент), предоставлять функционал управления курсами, лекциями и задачами, а также сохранять данные в реляционной базе данных. Приложение должно быть спроектировано с применением клиент-серверной архитектуры. Клиентская часть программы должна обладать графическим интерфейсом, реализованным с использованием веб-технологий. Система должна включать функционал аутентификации и авторизации пользователей, а также передачи данных через REST API.

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметных областей: изучить аналоги существующих систем управления образовательными курсами, изучить актуальные способы разработки веб-приложений с клиент-серверной архитектурой, исследовать методы проектирования и реализации REST API, изучить способы обеспечения безопасности веб-приложений, включая аутентификацию и авторизацию, проанализировать функционал фреймворков и технологий: Spring Boot, React, Spring Data JPA, MySQL, исследовать подходы к проектированию реляционных баз данных для хранения образовательного контента.

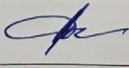
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	5%	06.05.2025-08.05.2025	
2.	Анализ требований	10%	09.05.2025-13.05.2025	
3.	Выбор языков программирования и технологий веб-разработки	10%	14.05.2025-16.05.2025	
4.	Создание архитектуры веб-сервиса для управления учебными курсами	30%	15.05.2025-20.05.2025	
5.	Разработка и интеграция всех компонентов сервиса	15%	21.05.2025-24.05.2025	
6.	Тестирование и отладка системы	10%	25.05.2025-26.05.2025	
7.	Оформление пояснительной записки	20%	26.05.2025-30.05.2025	

Дата выдачи задания "06" мая 2025 г.

Руководитель бакалаврской работы  06.05.2025 Кузнецов А.В.
(подпись, дата)

Задание к исполнению принял студент  06.05.2025 Асянов И.К.
(подпись, дата)

Работу к защите допустить
Декан факультета  11.06.2025 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 114 листов, 16 рисунков, 6 таблиц, 10 источников и 3 приложения.

УПРАВЛЕНИЕ КУРСАМИ, КЛИЕНТ-СЕРВЕРНАЯ АРХИТЕКТУРА, ВЕБ-ПРИЛОЖЕНИЕ, SPRING BOOT, REACT, MYSQL, REST API, АУТЕНТИФИКАЦИЯ, РОЛИ ПОЛЬЗОВАТЕЛЕЙ

Цель работы – разработать веб-систему для управления образовательными курсами, обеспечивающую взаимодействие преподавателей и студентов через клиент-серверную архитектуру.

Объектами разработки являются:

- серверное приложение на основе Spring Boot для управления данными о курсах, лекциях, задачах, пользователях, а также обеспечения REST API и аутентификации;
- клиентское веб-приложение на основе React для взаимодействия пользователей с системой, настройки параметров и отображения данных о курсах и задачах.

В результате проделанной работы разработана система, полностью соответствующая поставленной задаче, а именно: реализована серверная часть приложения на Spring Boot, которая управляет данными о курсах, лекциях, задачах и пользователях, сохраняет их в реляционной базе данных MySQL. Разработано API для обмена данными между сервером и клиентом, обеспечивающее настройку параметров и отображение результатов. Создана клиентская часть на React с графическим интерфейсом, позволяющая преподавателям создавать и редактировать курсы, лекции и задачи, а студентам – записываться на курсы и оценивать их, выполнять задачи.

					ПГУ 1.090301.04.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата	Сервис для управления учебными курсами Пояснительная записка	Лит.	Лист	Листов	
Разраб.		Асянов И.К.		10.06.25					
Провер.		Кузнецов А.В.		10.06.25			2	114	
Н. Контр		Бычков А.С.		10.06.25					
Утверд.		Митрохин М.А.		11.06.25					
						ФВТ, каф. ВТ гр. 21ВВП1			

Содержание

Перечень сокращений и обозначений.....	
Введение.....	
1 Сервис для управления учебными курсами.....	
1.1 Анализ предметной области.....	
1.1.1 Анализ потребностей пользователей.....	
1.1.2 Анализ существующих аналогов.....	
1.1.4 Постановка задачи.....	
1.1.5 Требования к аппаратно-программному обеспечению.....	
1.2 Выбор языков программирования и технологий веб-разработки.....	
1.2.1 Язык гипертекстовой разметки HTML.....	
1.2.2 Язык стилей CSS.....	
1.2.3 Язык программирования JS.....	
1.2.4 Язык программирования Java.....	
1.2.5 Система управления базами данных MySQL.....	
1.3 Выбор инструментов разработки.....	
1.3.1 Выбор среды разработки.....	
1.3.2 Выбор вспомогательных библиотек.....	
1.4 Архитектура приложения.....	
1.4.1 Паттерн MVC.....	
1.4.2 Обзор веб-сервера Tomcat.....	
1.5 Разработка приложения.....	
1.5.1 Разработка фронтенда.....	
1.5.2 Разработка компонентов интерфейса.....	
1.5.3 Разработка стилей интерфейса.....	
1.5.4 Разработка серверной части.....	
1.5.4.1 Разработка базы данных.....	
1.5.4.2 Реализация серверной логики.....	

1.6 Описание веб-сервиса

1.6.1 Руководство пользователя

2 Безопасность жизнедеятельности

2.1 Требования к организации рабочего места

2.2 Требования к освещению

2.3 Требования к режиму отдыха

Заключение

Список использованных источников

Приложение А – UML-диаграммы

Приложение Б – Листинг программы

Приложение В – Презентация

					ПГУ 1.090301.04.001 ПЗ	Лист
						4
Изм.	Лист	№ докум.	Подпись	Дата		

Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения

БД – база данных

ПК – персональный компьютер

ПО – программное обеспечение

СУБД – система управления базами данных

API (Application Programming Interface) – набор готовых правил и определений

CSS (Cascading Style Sheets) – каскадные таблицы стилей

JPA (Java Persistence API) – интерфейс для работы с базами данных в Java

JS (JavaScript) – язык программирования для веб-разработки

JSON (JavaScript Object Notation) – формат обмена данными

GUI (Graphical User Interface) – графический интерфейс пользователя

HTML (HyperText Markup Language) – язык разметки

IDE (Integrated Development Environment) – интегрированная среда разработки

MVC (Model View Controller) – шаблон проектирования архитектуры приложения

UML (Unified Modeling Language) – унифицированный язык моделирования

XML (eXtensible Markup Language) – расширяемый язык разметки

Введение

Сервис для управления учебными курсами представляет собой платформу, предназначенную для организации и упрощения процесса обучения в онлайн-формате. Данный сервис объединяет преподавателей и студентов, предоставляя инструменты для эффективного взаимодействия: преподаватели могут создавать курсы, загружать учебные материалы, задавать задания и оценивать работы, а студенты – записываться на курсы, получать доступ к материалам, сдавать задания и отслеживать свои результаты.

Целью работы является разработка сервиса для управления учебными курсами, который позволит легко и быстро настраивать учебный процесс, следуя современным тенденциям образования.

В рамках работы необходимо создать веб-сервис, включающий бэкенд на основе REST API, фронтенд для взаимодействия пользователей с системой и базу данных для хранения информации. Веб-приложение должно предоставлять интуитивно понятный интерфейс. Входными данными является база данных с информацией о пользователях, курсах и учебных материалах, из которой сервис извлекает необходимые сведения. Результатом работы сервиса будет предоставление пользователям функционала для управления курсами и учебным процессом, что обеспечит удобное и эффективное взаимодействие между преподавателями и студентами.

1 Сервис для управления учебными курсами

1.1 Анализ предметной области

В настоящее время онлайн-обучение приобретает большую популярность, но без удобного сервиса держать всё под контролем становится сложнее. Преподаватели тратят часы на отправку материалов и проверку заданий по почте, а студенты путаются во множестве чатов и ссылок. Сервис для управления учебными курсами способен упростить этот процесс, собрав всё в одном месте: курсы, задания, оценки.

Сервисы для организации обучения через интернет – важный инструмент в современном образовании. Платформы, вроде Moodle или Google Classroom давно популярны, но часто они либо слишком сложные для небольших курсов, либо не дают нужной гибкости. Данный сервис позволит преподавателям создавать курсы, выкладывать материалы и следить за прогрессом студентов, а студентам – легко находить задания и видеть свои результаты.

Одно из главных преимуществ проекта – это удобство для преподавателей. Вместо того чтобы разбираться с несколькими файлами или писать каждому студенту, они могут загрузить лекции в виде URL-ссылки и поставить оценки прямо в системе. Это сокращает время на рутину и позволяет сосредоточиться на самом обучении. Студенты в свою очередь получают доступ ко всем материалам и заданиям в одном месте. Это особенно полезно для тех, кто совмещает учёбу с работой или другими делами.

Ещё одно важное преимущество – прозрачность учебного процесса. Студенты могут сразу видеть свои оценки и комментарии преподавателей, а преподаватели отслеживать, как каждый справляется с курсом.

Таким образом, сервис для управления учебными курсами – это полезный инструмент для всех, кто занимается онлайн-обучением. Он упрощает организацию учебного процесса, помогает следить за прогрессом и создаёт комфортные условия для учёбы.

На рисунке 1 показан график популярности запроса «онлайн курсы для детей» с 2020 по 2024 год.

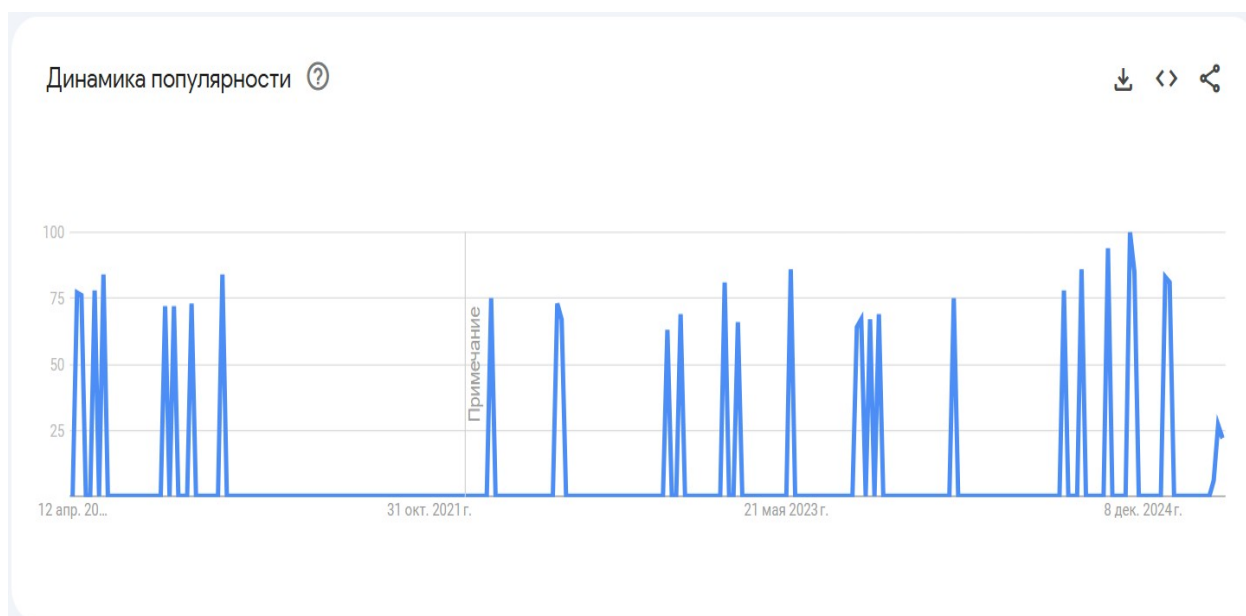


Рисунок 1 – График популярности запросов

1.1.1 Анализ потребностей пользователей

Исходя из темы, можно выделить следующие возможности потребности пользователей:

- простой и понятный интерфейс. Пользователям нужен такой, в котором легко разобраться, чтобы быстро найти нужный курс, скачать материалы или сдать задание;
- удобный поиск и фильтрация курсов. Студентам необходима система, которая позволит находить курсы по названию, тематике, а преподавателям видеть списки своих курсов с возможностью сортировки;
- инструменты для взаимодействия. Пользователям важно иметь возможность общаться внутри сервиса, например, студентам через комментарии к заданиям;
- безопасность и защита данных. Пользователям нужна надёжная платформа, где их личные данные (email, пароли) будут защищены, а доступ к курсам и оценкам только у авторизованных пользователей;

- система оценок и обратной связи. Студентам важно видеть свои оценки и комментарии преподавателей по заданиям, а преподавателям – удобный способ ставить баллы и видеть комментарии студентов, чтобы отслеживать прогресс;
- доступ с мобильных устройств. Пользователям нужен интерфейс, который хорошо работает на телефонах и планшетах, чтобы студенты могли отправить решение или скачать материалы на ходу, а преподаватели управлять курсами без привязки к компьютеру.

Этот сервис будет полезен в первую очередь преподавателям, которые ведут небольшие онлайн-курсы или работают частно. Они часто сталкиваются с проблемой: материалы разбросаны по почте, задания теряются, а следить за каждым студентом вручную становится трудно. С помощью сервиса, построенного на Java Spring Boot с REST API, React для интерфейса и MySQL для базы данных, они смогут всё организовать в одном месте: выложить лекции, задать задания и сразу видеть, кто что сдал.

Студентам сервис нужен, чтобы не путаться в куче ссылок и чатов. Особенно это выручит тех, кто учится между работой или другими делами. Вместо поисков файлов в Telegram или Google Drive они смогут зайти в сервис и найти всё нужное: от расписания курсов до своих оценок. Частным образовательным платформам или небольшим учебным центрам сервис тоже пригодится – он проще, чем громоздкие системы вроде Moodle, и заточен под конкретные задачи.

1.1.2 Анализ существующих аналогов

Среди аналогов сервисов по управлению курсами можно выделить Moodle и Google Classroom.

Moodle – это сервис для организации онлайн-обучения, который используется школами, университетами и частными преподавателями. Запущен в 2002 году в Австралии Мартином Дугимасом. По данным на 2023 год, у Moodle более 400 млн пользователей в 240 странах, а в России – около 10% всех образовательных учреждений используют эту платформу. Свыше 2 млн визитов в месяц на официальный сайт.

Плюсы: бесплатный open-source, гибкая настройка, поддержка множества форматов контента (PDF, видео, тесты). Минусы: сложный интерфейс для новичков, требует настройки сервера.

На рисунке 2 показана главная страница данного сервиса.

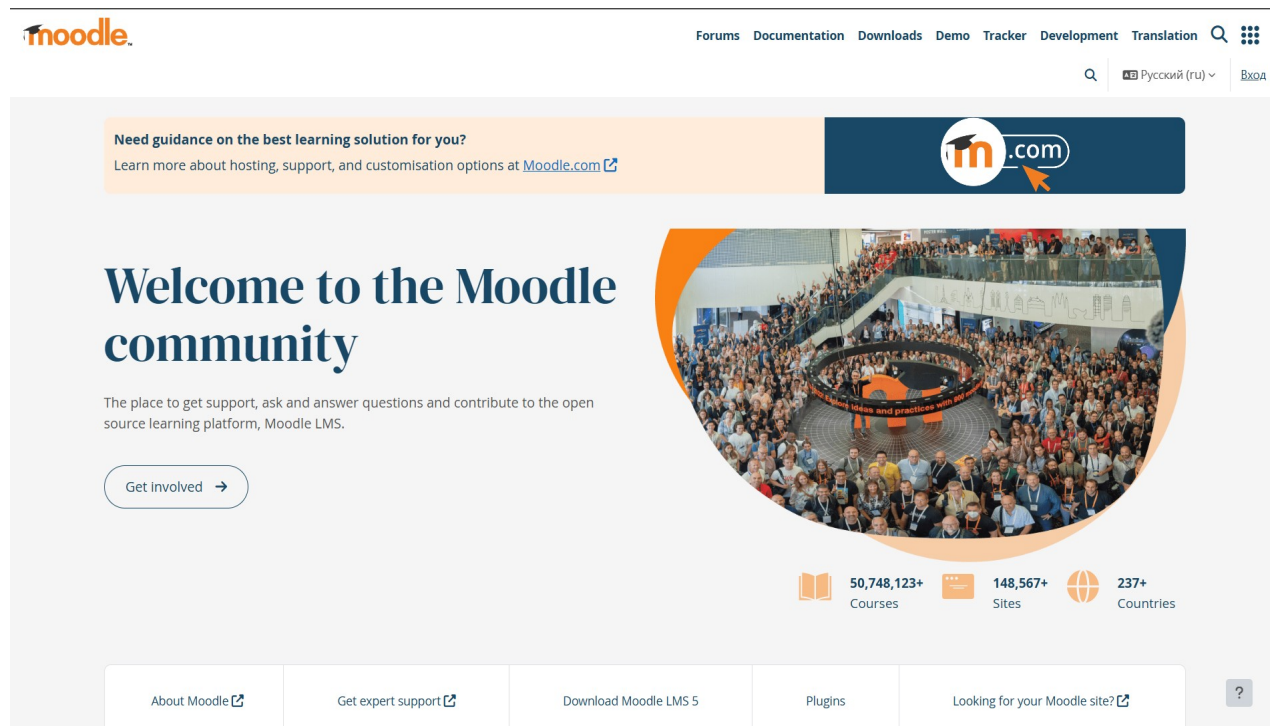


Рисунок 2 – Сервис управления курсами Moodle

Присутствуют следующие разделы:

- главная: содержит поле для входа в систему, информацию о возможностях платформы (курсы, тесты, аналитика) и ссылки на популярные курсы или сообщества пользователей. Также есть раздел с новостями и рекомендациями по использованию;
- о платформе: информация о Moodle, документация, сообщество разработчиков и контакты для поддержки;
- авторизация и регистрация: ссылки для входа или создания аккаунта для преподавателей и студентов;
- ресурсы: раздел с плагинами, шаблонами и обучающими материалами для настройки курсов.

Google Classroom – сервис для управления учебными курсами, интегрированный с экосистемой Google. Запущен в 2014 году компанией Google в США. По данным на 2023 год, у сервиса более 150 млн пользователей по всему миру, из них около 5 млн в России. Свыше 1 млн посещений в месяц. Плюсы: простой и понятный интерфейс, интеграция с Google Drive и Meet, бесплатный доступ. Минусы: ограниченный функционал для сложных курсов, отсутствие поддержки SCORM. На рисунке 3 показана главная страница данного сервиса.

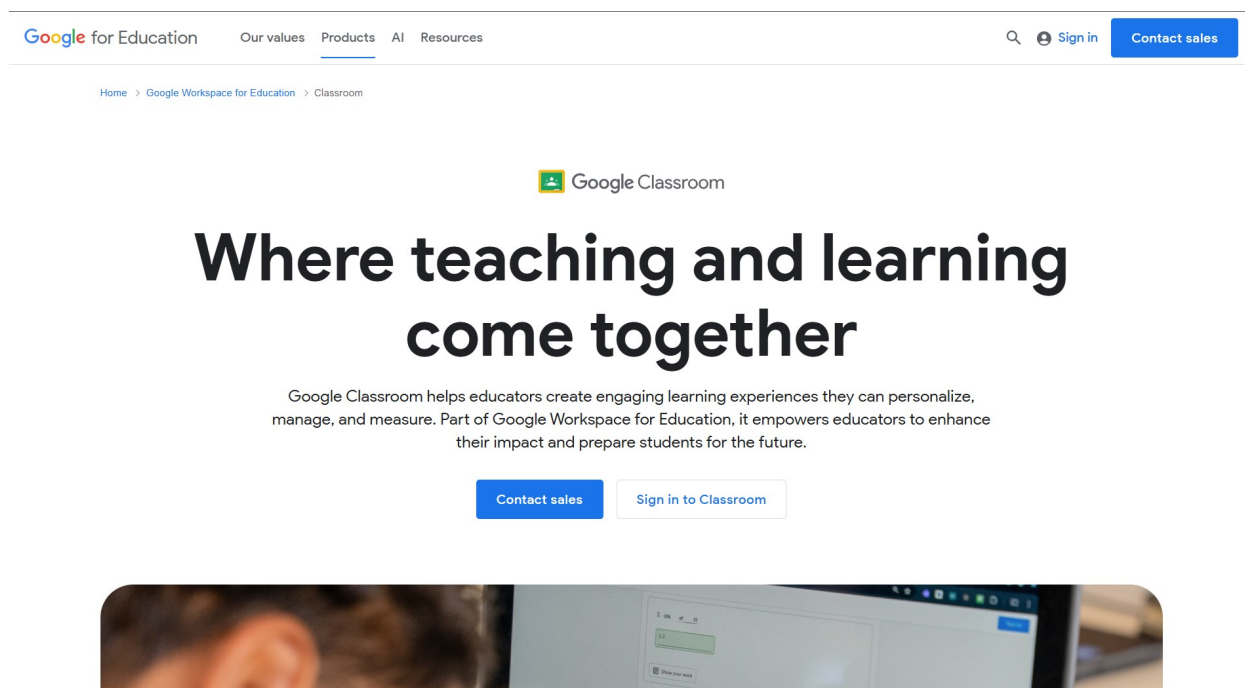


Рисунок 3 – Сервис управления курсами Google Classroom

Присутствуют следующие разделы:

- главная: отображает список курсов для преподавателей и студентов, поле для создания нового курса или задания, а также календарь с дедлайнами;
- о сервисе: информация о Google Classroom, инструкции по использованию и ссылки на поддержку Google;
- авторизация: вход через аккаунт Google, что упрощает регистрацию;
- интеграции: раздел с информацией о подключении Google Drive, Meet и других инструментов для загрузки материалов и проведения видео-встреч.

1.1.4 Постановка задачи

Для того чтобы правильно и объективно сформулировать задачи, необходимо четко представлять себе предметную область разработки.

На рисунке 4 представлен план разработки.

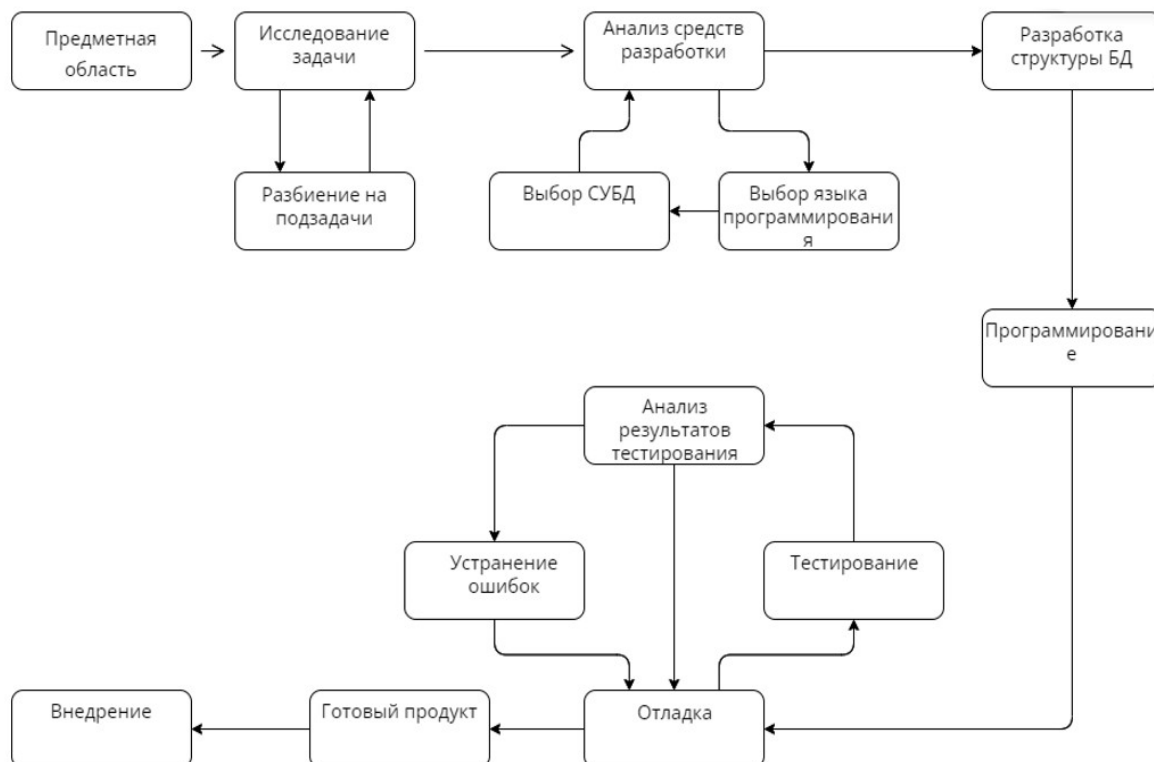


Рисунок 4 – Разработка программного продукта

Необходимо разработать сервис для управления учебными курсами. В данном веб-сервисе будет реализован функционал, обеспечивающий создание и управление курсами, лекциями и заданиями преподавателями, а также просмотр и выполнение учебных материалов студентами.

Возможности сервиса должны предоставлять взаимодействие между преподавателями и студентами, форму создания курсов и заданий, просмотр активных курсов с фильтрацией по категориям (программирование, дизайн, маркетинг, математика).

Функционал рейтинга студентов мотивирует их к активному участию в обучении, повышая вовлечённость и эффективность освоения материала.

Для решения этих задач необходимо разработать простой, интуитивно понятный и удобный пользовательский интерфейс.

Веб-приложение должно быть разработано с использованием модульной архитектуры, включающей отдельные компоненты для управления курсами, лекциями, заданиями и пользовательскими ролями.

Такой подход облегчает добавление новых функций, таких как таблица лидеров или аналитика успеваемости, а также упрощает выявление и устранение ошибок при отладке и тестировании.

1.1.5 Требования к аппаратно-программному обеспечению

В таблице 1 указаны минимальная и рекомендуемая конфигурации аппаратного и программного обеспечения, необходимые и достаточные для корректного функционирования сервиса.

Таблица 1 – Минимальная и рекомендуемая конфигурации аппаратного и программного обеспечения

Параметр оборудования	Минимальная конфигурация	Рекомендуемая конфигурация
Процессор	Intel® Celeron® J4125 (4 ядра, 2.0 ГГц)	Intel® Xeon® E5-1220 v5 (4 ядра, 3.0 ГГц)
Объем оперативной памяти	8 Гб	16 Гб
Объем постоянной памяти	30 Гб	500 Гб
Скорость сетевого интерфейса	100 Мбит/с	1 Гбит/с

1.2 Выбор языков программирования и технологий веб-разработки

В соответствии с поставленной задачей необходимо разработать компоненты сервиса для управления учебными курсами. Современный инструментарий разработчика предоставляет широкий выбор средств для реализации проектов такого рода. Перед началом разработки были проанализированы существующие среды и технологии для создания веб-приложений, обеспечивающих управление курсами, лекциями, заданиями и взаимодействие между преподавателями и студентами.

1.2.1 Язык гипертекстовой разметки HTML

HTML или язык гипертекстовой разметки – это стандартный язык разметки, используемый для создания веб-страниц и приложений. Это основа практически всех сайтов в Интернете. Его основные преимущества и недостатки перечислены в таблице 2.

Таблица 2 – Преимущества и недостатки HTML

Преимущества	Недостатки
Простота в освоении и использовании	Ограниченная интерактивность и функциональность без дополнительных языков сценариев, таких как JavaScript
Поддерживается браузерами	Может потребоваться много времени для создания сложных макетов и дизайнов
Обеспечивает семантику веб-контента, улучшая доступность и SEO.	При неправильном использовании может привести к уязвимостям, например, XSS-атакам.
Легко интегрируется с другими технологиями	Требует строгого соблюдения стандартов для кроссбраузерной совместимости.

1.2.2 Язык стилей CSS

CSS или каскадные таблицы стилей – это язык стилей, используемый для управления представлением и макетом веб-страниц. Он используется для определения внешнего вида HTML-элементов, включая их размер, цвет, шрифт, интервалы и другие визуальные характеристики.

Таблица 3 – Преимущества и недостатки CSS

Преимущества	Недостатки
Разделение задач: CSS позволяет четко разделить содержимое и представление, упрощая поддержку и обновление веб-сайтов	Совместимость с браузерами: разные браузеры могут интерпретировать CSS немного по-разному, что приводит к несоответствиям во внешнем виде
Согласованность: определяя стили в одном месте, CSS может гарантировать, что все страницы на веб-сайте будут выглядеть одинаково	Кривая обучения: CSS может быть трудным для изучения, особенно для новичков, которые могут испытывать трудности с синтаксисом и макетом
Гибкость: CSS предоставляет широкий спектр вариантов стилей, что позволяет создавать сложные макеты и дизайны	Специфика: CSS может быть сложным и требовать глубокого понимания специфики селекторов и правил наследования
Эффективность: CSS может уменьшить количество кода, необходимого для оформления веб-сайта, что может ускорить загрузку страниц	Обслуживание: CSS со временем может усложниться, особенно, если на веб-сайте много страниц, или над проектом работает несколько разработчиков

1.2.3 Язык программирования JS

JavaScript (JS) – это высокоуровневый интерпретируемый язык программирования, который обычно используется для создания интерактивных веб-страниц и динамических пользовательских интерфейсов. Это позволяет разработчикам добавлять функциональные возможности на свои веб-сайты и веб-приложения, включая анимацию, проверку форм и другие динамические элементы. В courseplatform JavaScript, в связке с React, обеспечивает динамическое обновление UI (например, фильтрация курсов по категориям) и взаимодействие с API (запросы к CourseController). JavaScript часто называют клиентским языком, что означает, что он работает на компьютере пользователя, а не на сервере. Он также используется в серверных средах с помощью таких технологий, как Node.js. JS – очень популярный язык, который поддерживается всеми современными веб-браузерами. Преимущества и недостатки представлены в таблице 3

Таблица 4 – Преимущества и недостатки JS

Преимущества	Недостатки
Он лёгок в изучении и реализации	Некорректная реализация может привести к ошибкам в функциональности JavaScript.
Он поддерживается всеми основными веб-браузерами	Он может быть уязвим для угроз безопасности, если не защищен должным образом
Он универсален и может использоваться для самых разных целей	Он может вызвать проблемы с производительностью, если не оптимизировано должным образом
Он постоянно развивается и улучшается с новыми обновлениями и функциями	Это не серверный язык программирования, поэтому он не может самостоятельно выполнять сложные серверные операции

1.2.4 Язык программирования Java

Java – высокоуровневый, объектно-ориентированный язык программирования, широко применяемый для разработки серверных приложений благодаря своей надёжности, масштабируемости и независимости платформы.

В данном проекте Java используется для реализации серверной части, включая бизнес-логику, REST API (например, CourseController для управления курсами, лекциями и заданиями) и взаимодействие с реляционной базой данных MySQL через фреймворк Hibernate. Фреймворк Spring Boot обеспечивает упрощённую конфигурацию серверного приложения, поддержку зависимостей и встроенный сервер (Tomcat), что позволяет быстро разворачивать API для обработки запросов от клиентской части (React, Courses.js). Java обрабатывает такие операции, как создание курсов, управление лекциями, заданиями, а также аутентификацию пользователей. Платформонезависимость Java, обеспечиваемая виртуальной машиной JVM, делает её идеальным для данного проекта.

Таблица 5 – Преимущества и недостатки Java

Преимущества	Недостатки
Java, благодаря JVM и оптимизации JIT-компиляции, обеспечивает эффективную обработку запросов в высоконагруженных системах.	По сравнению с легковесными языками (например, PHP или Node.js), Java требует больше памяти и процессорных ресурсов, что может увеличить затраты на серверное оборудование.
Статическая типизация и строгие проверки компилятора снижают вероятность ошибок.	Переход на новые версии может потребовать доработки из-за изменений в API или зависимостях.

Spring Boot упрощает разработку REST API, а Hibernate автоматизирует взаимодействие с MySQL	Требуется больше времени на настройку и написание кода по сравнению с интерпретируемыми языками (например, Python), особенно для небольших проектов
Обширная поддержка сообщества и документация (например, для Spring, Hibernate) ускоряют решение проблем	Для высоконагруженных платформ требуется тонкая настройка JVM (например, управление сборкой мусора) и оптимизация кода для минимизации задержек
Платформонезависимость: код, написанный на Java, работает на Windows Server или других ОС без изменений, что упрощает развертывание и тестирование	Сложный синтаксис и концепции (например, инверсия управления в Spring) могут быть барьером для начинающих разработчиков, увеличивая время адаптации

1.2.5 Система управления базами данных MySQL

MySQL, написанная на языках C и C++, является одной из наиболее популярных реляционных СУБД для веб-приложений благодаря своей кроссплатформенной совместимости и масштабируемости. Она использует SQL (язык структурированных запросов) для управления данными и отличается высокой эффективностью при обработке больших объёмов информации. MySQL обеспечивает надёжную безопасность и поддерживает функции, такие как управление пользователями, резервное копирование, восстановление и репликацию данных для повышения отказоустойчивости. Кроме того, MySQL интегрируется с популярными фреймворками, такими как Spring Boot, что упрощает разработку современных веб-приложений. Её открытый исходный код позволяет сообществу активно развивать и адаптировать систему под различные нужды.

MySQL поддерживает широкий набор инструментов для оптимизации запросов и индексации, что ускоряет выполнение операций с данными в образовательной платформе. Её интеграция с Java через JDBC и Hibernate упрощает разработку серверной части courseplatform, обеспечивая надёжное управление таблицами. Преимущества и недостатки MySQL указаны в таблице 6.

Таблица 6 – Преимущества и недостатки MySQL

Преимущества	Недостатки
Открытый исходный код: MySQL бесплатна для использования и широко поддерживается сообществом разработчиков.	По сравнению с другими базами данных MySQL имеет ограниченные возможности.
MySQL может обрабатывать большие объёмы данных и известен высокой скоростью обработки.	MySQL имеет репутацию менее безопасной по сравнению с другими базами данных.
MySQL имеет удобный интерфейс, в котором легко ориентироваться и использовать.	MySQL имеет ограниченную систему поддержки по сравнению с другими коммерческими базами данных.
MySQL можно легко масштабировать для удовлетворения потребностей	MySQL может быть менее эффективной для сложных транзакций с множественными зависимостями, что может повлиять на управление заданиями и их сдачей

1.3 Выбор инструментов разработки

Для разработки сервиса «Управление учебными курсами» были выбраны инструменты, обеспечивающие эффективную разработку, тестирование и поддержку проекта. Выбор основывался на совместимости с используемыми технологиями (Java, Spring Boot, React, MySQL), простоте интеграции и поддержке модульной архитектуры.

1.3.1 Выбор среды разработки

В качестве сред разработки выбраны Visual Studio Code (VS Code) для фронтенда и IntelliJ IDEA для бэкенда, что оптимально соответствует задачам разработки платформы «Управление учебными курсами». VS Code используется для работы с React-компонентами, а IntelliJ IDEA – для серверной части на Java и Spring Boot. Комбинация этих сред обеспечивает гибкость и высокую производительность.

Visual Studio Code – редактор с открытым исходным кодом, разработанный Microsoft, идеально подходит для фронтенд-разработки благодаря поддержке JavaScript, HTML и CSS.

Основные преимущества VS Code:

- простота использования: интуитивный интерфейс и быстрая установка упрощают работу с React-компонентами;
- расширяемость: поддержка расширений позволяет настроить среду для форматирования кода и ускорения разработки UI;
- навигация по коду: мощные функции поиска и IntelliSense обеспечивают быструю навигацию по фронтенд-коду, включая фильтрацию категорий;
- отладка: встроенные инструменты отладки для JavaScript помогают устранять ошибки в клиентской части, например, при взаимодействии с API;
- совместная работа: интеграция с Git и GitHub упрощает управление версиями и командную работу над фронтендом;

- многоязычная поддержка: поддержка JavaScript, HTML, CSS и других языков, необходимых для фронтенда платформы;
- межплатформенная поддержка: работает на Windows (основная ОС проекта), macOS и Linux, что удобно для разработки и тестирования.

IntelliJ IDEA – мощная интегрированная среда разработки от JetBrains, выбранная для бэкенд-разработки на Java и Spring Boot. Она обеспечивает глубокую интеграцию с фреймворками, что критично для создания REST API и взаимодействия с MySQL. Основные преимущества IntelliJ IDEA:

- интеллектуальная подсказка кода: автодополнение, анализ кода и рефакторинг ускоряют разработку CourseController и обработку данных;
- интеграция с Spring Boot: поддержка аннотаций, конфигураций и автосборки упрощает создание и отладку API для управления курсами и заданиями;
- отладка: расширенные инструменты отладки для Java позволяют быстро и без лишних операций находить и исправлять ошибки, например, в логике обработки запросов;
- встроенные инструменты: поддержка Maven, Gradle и JUnit облегчает управление зависимостями и тестирование серверной части;
- работа с базами данных: встроенный клиент для MySQL упрощает выполнение запросов и настройку схемы БД;
- производительность: оптимизирована для крупных проектов с модульной архитектурой, включающей курсы, лекции и задания.

Комбинация Visual Studio Code и IntelliJ IDEA обеспечивает эффективную и гибкую разработку фронтенда и бэкенда полностью покрывая потребности проекта. VS Code позволяет быстро создавать и отлаживать React-компоненты, обеспечивая интерактивный интерфейс и фильтрацию категорий. IntelliJ IDEA, в свою очередь, оптимизирует работу с Java и Spring Boot, упрощая разработку REST API в CourseController и взаимодействие с MySQL.

Вместе эти среды поддерживают модульную архитектуру, упрощают управление кодом и интеграцию с Git, а также обеспечивают надёжную отладку и тестирование.

1.3.2 Выбор вспомогательных библиотек

В качестве вспомогательной библиотеки для фронтенда выбран React – популярная JavaScript-библиотека с открытым исходным кодом, разработанная инженером Facebook. React используется для создания динамического и интерактивного пользовательского интерфейса платформы, включая компоненты для отображения курсов, лекций и заданий (например, Courses.js для фильтрации по категориям). Он предоставляет мощный инструментарий для управления компонентами и состояниями, что упрощает разработку сложных веб-приложений. React обладает следующими преимуществами:

- компонентная архитектура: разделение интерфейса на независимые компоненты (например, списки курсов, формы заданий) упрощает разработку, тестирование и поддержку кода;
- виртуальный DOM: обеспечивает высокую производительность при обновлении UI, что критично для динамической фильтрации категорий;
- кроссбраузерная совместимость: гарантирует единообразное отображение интерфейса в современных браузерах, улучшая пользовательский опыт;
- богатая экосистема: поддержка библиотек, таких как React Router для навигации и Axios для отправки запросов к API CourseController, ускоряет разработку;
- активное сообщество: обширная документация и поддержка сообщества упрощают решение проблем, связанных с реализацией UI.

Благодаря гибкости, производительности и поддержке современных стандартов веб-разработки React стал ключевым инструментом для создания интерактивного интерфейса платформы «Управление учебными курсами». Он позволяет эффективно реализовать динамические функции, такие как фильтрация

курсов и отображение лекций, что делает его оптимальным выбором для данного проекта.

Spring Boot, дополненное модулями Spring Data JPA, Hibernate и Spring Security, формирует основу бэкенда, обеспечивая управление бизнес-логикой, взаимодействие с базой данных MySQL и безопасность. Основные преимущества этих библиотек:

- Spring Boot: упрощает настройку и развертывание серверного приложения благодаря встроенному серверу (Tomcat) и автоматической конфигурации, что ускоряет разработку CourseController;

- Spring Data JPA: предоставляет удобный API для работы с MySQL (таблицы courses, lectures, tasks), минимизируя написание SQL-запросов и упрощая CRUD-операции;

- Hibernate: автоматизирует объектно-реляционное отображение (ORM), позволяя эффективно работать с данными, такими как поля content и description;

- Spring Security: обеспечивает аутентификацию и авторизацию пользователей, защищая API от несанкционированного доступа;

- экосистема Spring: интеграция с Maven и обширная документация упрощают управление зависимостями и решение проблем, таких как настройка Hibernate;

- модульность и расширяемость: Spring Boot и его модули позволяют легко интегрировать дополнительные библиотеки (например, Spring Web для REST API или Lombok для упрощения кода), что обеспечивает гибкость при добавлении нового функционала, такого как управление категориями.

Использование React для фронтенда и Spring Boot с модулями для бэкенда обеспечивает высокую производительность, масштабируемость и безопасность платформы, упрощая дальнейшее развитие проекта. Эти библиотеки позволяют эффективно реализовать функционал управления курсами, лекциями и заданиями, делая их оптимальным выбором для проекта и гарантируя надёжное управление данными и их целостность.

1.4 Архитектура приложения

1.4.1 Паттерн MVC

Паттерн – повторяемая архитектурная конструкция, представляющая собой решение типичной проблемы проектирования в определённом контексте. Всего существует три группы паттернов: структурные, порождающие и поведенческие, иерархия которых показана на рисунке 5.

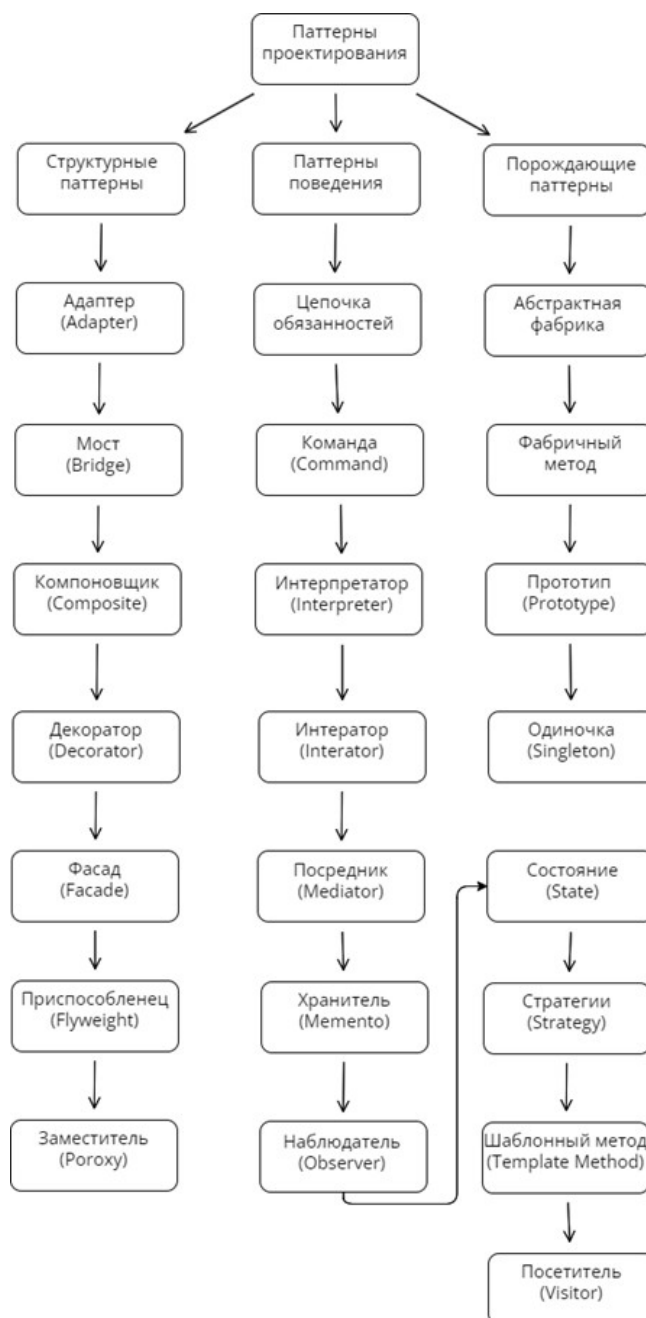


Рисунок 5 – Группы паттернов

Структурные паттерны связаны с организацией кода таким образом, чтобы его было легко читать, понимать и модифицировать. Они предоставляют решения для общих проблем с композицией объектов и структурами классов. Например, паттерн Adapter позволяет несовместимым классам работать вместе, а паттерн Decorator добавляет функциональность к объектам без изменения их структуры.

Порождающие паттерны используются для создания объектов и классов, обеспечивая их гибкость, расширяемость и повторное использование. Например, паттерн Factory отделяет создание объектов от остального кода, а паттерн Singleton гарантирует существование единственного экземпляра класса.

Поведенческие паттерны регулируют взаимодействие и коммуникацию между объектами. Например, паттерн Observer позволяет объектам получать уведомления об изменениях в других объектах, а паттерн Command инкапсулирует запросы в виде объектов для последующего выполнения.

Для разработки платформы «Управление учебными курсами» выбран паттерн Model-View-Controller (MVC), который оптимально подходит для веб-приложений с разделением бизнес-логики, интерфейса и управления. MVC разделяет приложение на три компонента: модель, представление и контроллер, обеспечивая их взаимодействие для создания удобного и масштабируемого интерфейса. Структура паттерна MVC представлена на рисунке 6.

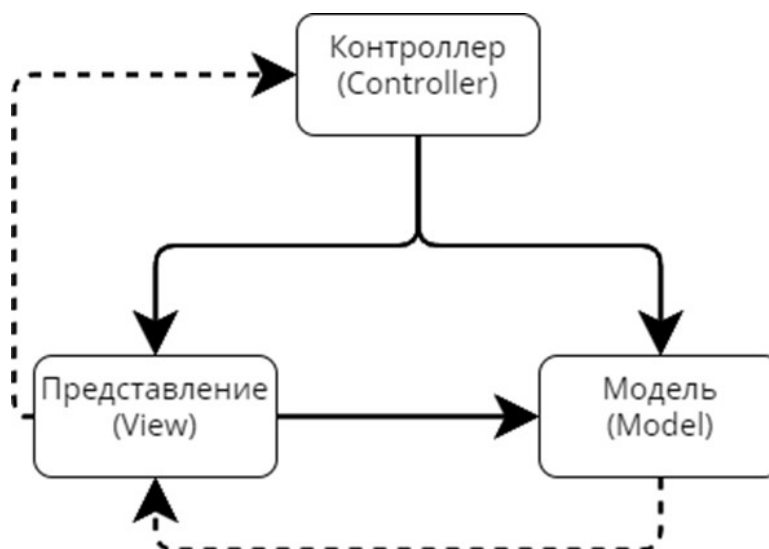


Рисунок 6 – Структура паттерна проектирования MVC

Модель отвечает за управление данными и бизнес-логикой. В проекте модель реализована с использованием Spring Data JPA и Hibernate, которые взаимодействуют с MySQL (таблицы courses, lectures, tasks). Модель обрабатывает данные, такие как content и description (тип TEXT), и обновляет представление при изменениях, например, при добавлении нового курса.

Представление отвечает за отображение данных в пользовательском интерфейсе. В платформе представление реализовано с помощью React (компонент Courses.js), который отображает списки курсов, лекций и заданий, а также поддерживает фильтрацию по категориям. Представление принимает ввод от пользователя и передаёт его контроллеру.

Контроллер управляет потоком данных между моделью и представлением. В проекте контроллер реализован через Spring Boot (CourseController), который обрабатывает HTTP-запросы от фронтенда, обновляет модель (например, создаёт записи в таблице tasks) и возвращает данные для обновления представления.

На рисунке 7 показана диаграмма взаимодействия MVC, демонстрирующая, как пользовательский запрос (например, фильтрация курсов) проходит через контроллер, обновляет модель и отражается в представлении.

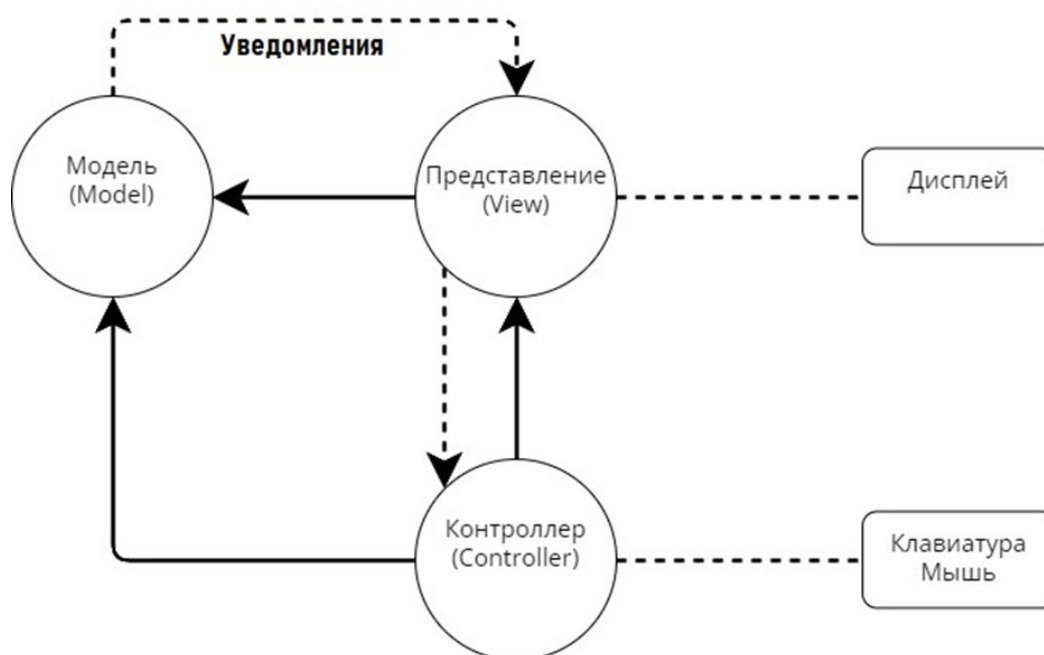


Рисунок 7 – Диаграмма взаимодействия MVC

Преимущества паттерна MVC включают чёткое разделение ответственности, что упрощает тестирование и поддержку приложения. Модель (courses, lectures, tasks) может тестироваться независимо от представления (Courses.js) и контроллера (CourseController), что позволяет вносить изменения в бизнес-логику без влияния на интерфейс. Представление и контроллер также могут модифицироваться без изменений в модели, что облегчает добавление новых функций, таких как управление категориями или заданиями. MVC обеспечивает масштабируемость и гибкость, что делает его оптимальным выбором для платформы «Управление учебными курсами».

1.4.2 Обзор веб-сервера Tomcat

Веб-сервер является ключевым компонентом инфраструктуры веб-приложений, обеспечивая обработку HTTP-запросов и доставку контента пользователям. Основным протоколом для веб-приложений является HTTP (Hypertext Transfer Protocol), который работает по принципу «запрос-ответ». Клиент отправляет запрос, содержащий тип запроса, заголовки и тело, а сервер возвращает код состояния, заголовки ответа и данные.

Для платформы «Управление учебными курсами» в качестве веб-сервера используется Apache Tomcat, встроенный в Spring Boot. Tomcat – это сервер приложений с открытым исходным кодом, разработанный Apache Software Foundation, предназначенный для выполнения Java-приложений, основанных на сервлетах и JSP. В отличие от традиционных веб-серверов, таких как Apache HTTP Server, Tomcat оптимизирован для работы с Java, что делает его идеальным для Spring Boot. Основные преимущества Tomcat:

- интеграция с Spring Boot: встроенный сервер Tomcat в Spring Boot устраняет необходимость в отдельной установке и конфигурации, упрощая деплой CourseController;
- поддержка Java-технологий: Tomcat эффективно обрабатывает сервлеты и REST API, обеспечивая высокую производительность при обработке запросов, например, для фильтрации курсов или управления заданиями;

- кроссплатформенность: работает на Windows, Linux и macOS, что позволяет тестировать приложение в разных окружениях;
- модульная структура поддерживает настройку через конфигурационные файлы, что упрощает управление сессиями, безопасностью и логами;
- открытый исходный код: бесплатное распространение и активное сообщество позволяют быстро устранять ошибки и добавлять новый функционал;
- безопасность: поддерживает SSL/TLS и интеграцию со Spring Security, обеспечивая защиту данных пользователей.

Недостатком Tomcat является его ориентация исключительно на Java-приложения, что ограничивает его использование для других типов серверов (например, PHP). Кроме того, для высоконагруженных систем может потребоваться дополнительная настройка производительности. Однако для дипломного проекта эти ограничения несущественны, так как проект использует Java и Spring Boot. Tomcat поддерживает HTTP/1.1 и HTTP/2, управление сессиями, логирование, настройку без перезапуска и интеграцию с Maven. Это делает Tomcat предпочтительным выбором для Spring Boot-приложений, обеспечивая надёжность и производительность.

1.5 Разработка приложения

1.5.1 Разработка фронтенда

Визуальная часть сайта включает в себя все элементы, которые видит пользователь при посещении сайта: дизайн, цветовую схему, шрифты, изображения, видео, анимацию, кнопки, формы, меню навигации и т.д. Она важна для создания привлекательного и удобного интерфейса, который позволяет пользователям быстро и легко находить нужную информацию и выполнять действия на сайте.

Графический интерфейс должен предоставлять возможности для взаимодействия с сервисом в соответствии с UML диаграммой вариантов использования, представленной на рисунке 8.

преподавателе. Для пользователей с ролью преподавателя предоставляется форма для создания новых курсов и кнопки для редактирования или удаления существующих, если они являются владельцами курса. Компонент использует хук `useState` для хранения списка курсов и формы нового курса, а хук `useEffect` выполняет GET-запрос к API (`GET /api/courses`) при загрузке страницы. Преподаватели могут редактировать курсы через PUT-запрос (`PUT /api/courses/{id}`) или удалять их через DELETE-запрос (`DELETE /api/courses/{id}`). Компонент также поддерживает переход к деталям курса по клику на заголовок, используя маршрутизацию (`/courses/{id}`);

`Leaderboard.js`: компонент отображает таблицу лидеров, показывающую студентов с их баллами, за выполненные задания (`GET /api/users/leaderboard`). Использует `useState` для хранения списка пользователей и `useEffect` для загрузки данных. Таблица формируется в виде сетки карточек;

`CourseDetails.js`: компонент отображает подробную информацию о курсе, включая заголовок, описание, преподавателя и список лекций. Для преподавателей, владеющих курсом, доступна форма для создания новых лекций (`POST /api/courses/{id}/lectures`), а также кнопки для редактирования (`PUT /api/courses/{id}/lectures/{lectureId}`) и удаления лекций (`DELETE /api/courses/{id}/lectures/{lectureId}`). Студенты с ролью могут записаться на курс через кнопку «Записаться» (`POST /api/courses/{id}/enroll`). Компонент использует `useParams` для получения ID курса из URL и `useState` для управления состоянием курса, лекций и формы редактирования. Лекции отображаются в сетке, каждая с возможностью загрузки презентации через `fileUrl`;

`Login.js` и `Register.js`: компоненты реализуют авторизацию и регистрацию пользователей. `Login.js` отправляет запрос `POST /api/users/me` с учетными данными в заголовке `Authorization: Basic {btoa(email:password)}`, сохраняя их в `localStorage` при успешном входе. `Register.js` позволяет выбрать роль и отправляет `POST /api/auth/register`. Оба компонента используют `useState` для управления полями формы и отображения ошибок;

Tasks.js: компонент управляет заданиями, предоставляя преподавателям возможность создавать задания (POST /api/tasks), просматривать и оценивать решения студентов (POST /api/tasks/submissions/{id}/grade). Студенты могут отправлять решения в виде текста или ссылки на файл (POST /api/tasks/{id}/submit). Компонент использует useState для хранения списка заданий, курсов и данных решений, а useEffect загружает задания (GET /api/tasks) и решения (GET /api/tasks/{id}/submissions). Преподаватели могут выставять баллы за решения, которые влияют на рейтинг студентов в таблице лидеров.

JSX (JavaScript XML) используется для описания структуры интерфейса, объединяя HTML-подобный синтаксис с JavaScript. JSX-теги определяют элементы интерфейса, такие как заголовки (<h1>, <h2>), списки (<div className="grid">), формы (<form>), кнопки (<button>), поля ввода (<input>, <textarea>), и ссылки (<a>). JSX позволяет встраивать JavaScript-выражения.

JSX обеспечивает декларативный подход, упрощая создание сложных интерфейсов и их обновление при изменении данных.

События: Обработчики событий (onClick, onSubmit, onChange) обеспечивают интерактивность.

В CourseDetails.js кнопка «Создать лекцию» вызывает handleCreateLecture, отправляющий POST-запрос с данными лекции.

В Tasks.js поле для решения задания обновляет состояние через onChange, а кнопка «Отправить решение» вызывает handleSubmitTask.

Состояние. Хуки useState и useEffect управляют локальным состоянием и асинхронными операциями. Первый хранит данные, такие как список курсов, форма новой лекции или текст решения задания. Второй выполняет запросы к API при монтировании компонента или изменении зависимостей. Хуки обеспечивают реактивность интерфейса, автоматически обновляя компоненты при изменении данных. Маршрутизация. Библиотека react-router-dom обеспечивает навигацию между страницами. Основные маршруты:

- /: список курсов (Courses.js);
- /courses/{id}: детали курса (CourseDetails.js);
- /tasks: список заданий (Tasks.js);
- /leaderboard: таблица лидеров (Leaderboard.js);
- /login и /register: Авторизация и регистрация. Компонент ProtectedRoute проверяет наличие учетных данных в localStorage, перенаправляя неавторизованных пользователей на /login.

Обработчики событий обеспечивают интерактивность. Навигация реализуется через хук useNavigate и компонент Link для ссылок в меню.

Аутентификация основана на HTTP Basic Authentication. Учетные данные кодируются в Base64 и сохраняются в localStorage как credentials. Каждый запрос к API включает заголовок Authorization: Basic {credentials}. Компоненты проверяют роль пользователя через GET /api/users/me, устанавливая флаги.

1.5.3 Разработка стилей интерфейса

Для оформления пользовательского интерфейса платформы «Управление учебными курсами» использован язык CSS, обеспечивающий визуальную привлекательность, адаптивность и единообразие дизайна. Стили применялись к элементам интерфейса, созданным с использованием React-компонентов (Courses.js, CourseDetails.js, Tasks.js, Leaderboard.js, Login.js, Register.js), и определены в файле index.css. Основные элементы оформления включают списки курсов, карточки лекций и заданий, формы для создания контента, навигационное меню, таблицу рейтинга, а также страницы входа и регистрации. Селекторы в CSS используются для выбора элементов интерфейса и применения к ним стилей. В проекте применялись селектор элементов, селектор классов и селектор ID.

Селектор элементов применяется для стилизации всех экземпляров определённого HTML-элемента. Например, для задания фона и шрифта всей страницы использовался селектор элемента <body>.

Селектор классов используется для стилизации элементов с определённым атрибутом class. Классы широко применялись в React-компонентах через атрибут className. Например, для оформления карточек курсов, лекций и заданий использовался класс .card.

Селектор ID применяется для уникальных элементов, но использовался редко, так как React ориентирован на классы. Например, для корневого контейнера страницы мог использоваться селектор #root.

Классы задавались в JSX-тегах, например: <div className="grid">, где grid это название класса, определённое в index.css.

Блочная модель CSS определяет макет элементов интерфейса, включая их размеры, отступы, границы и поля. В проекте использовались следующие свойства блочной модели:

- ширина и высота устанавливают размеры элементов. Например, для контейнера страницы использовалось свойство max-width;
- границы формируют рамки вокруг элементов. Для карточек курсов применялись скруглённые границы;
- отступы создают пространство между содержимым и границей элемента. Например, для форм задавался внутренний отступ.

Эти свойства обеспечили чёткую структуру интерфейса, разделяя элементы и создавая визуальную иерархию.

Текстовые стили управляют внешним видом текста в интерфейсе. В проекте применялись следующие свойства:

- семейство шрифтов устанавливает шрифт для текста. Для всего интерфейса использовался шрифт Arial;
- размер шрифта определяет размер текста. Например, для заголовков страниц использовался увеличенный размер;
- выравнивание текста управляет расположением текста. Заголовки центрировались.

Свойства макета определяют расположение элементов на странице. В проекте использовались следующие подходы:

- `display` управляет отображением элементов, а для создания адаптивной сетки курсов использовался `grid`;
- `flexbox` применялся для горизонтального расположения элементов, например, в навигационном меню;
- `position` использовался редко, но мог применяться для эффектов, таких как позиционирование анимированных элементов.

Эти свойства обеспечили адаптивный и гибкий макет, подходящий для разных экранов.

Текстовые свойства используются для управления внешним видом текста на веб-сайте. В CSS применялись такие свойства, как `font-family`, `font-size`, `color` и `text-align`, для стилизации заголовков курсов и текстов лекций. Для повышения интерактивности интерфейса применялись CSS-анимации эффекта мерцания (`glitch`). При наведении на заголовки (например, название курса) создаётся анимация мерцания. А также эффект пульсации (`pulse`): при наведении на кнопки применяется масштабирование.

Анимации добавляют динамики, не перегружают интерфейс и соответствуют современным стандартам дизайна. Анимации добавляют динамики, не перегружают интерфейс и соответствуют современным стандартам дизайна. Для оптимизации анимации использовались свойства `will-change` и `transform`, что повысило производительность интерфейса. Эффекты были интегрированы в React-компоненты, обеспечивая плавное взаимодействие с пользователем.

На рисунке 9 показан итог верстки главной страницы веб-сервиса.

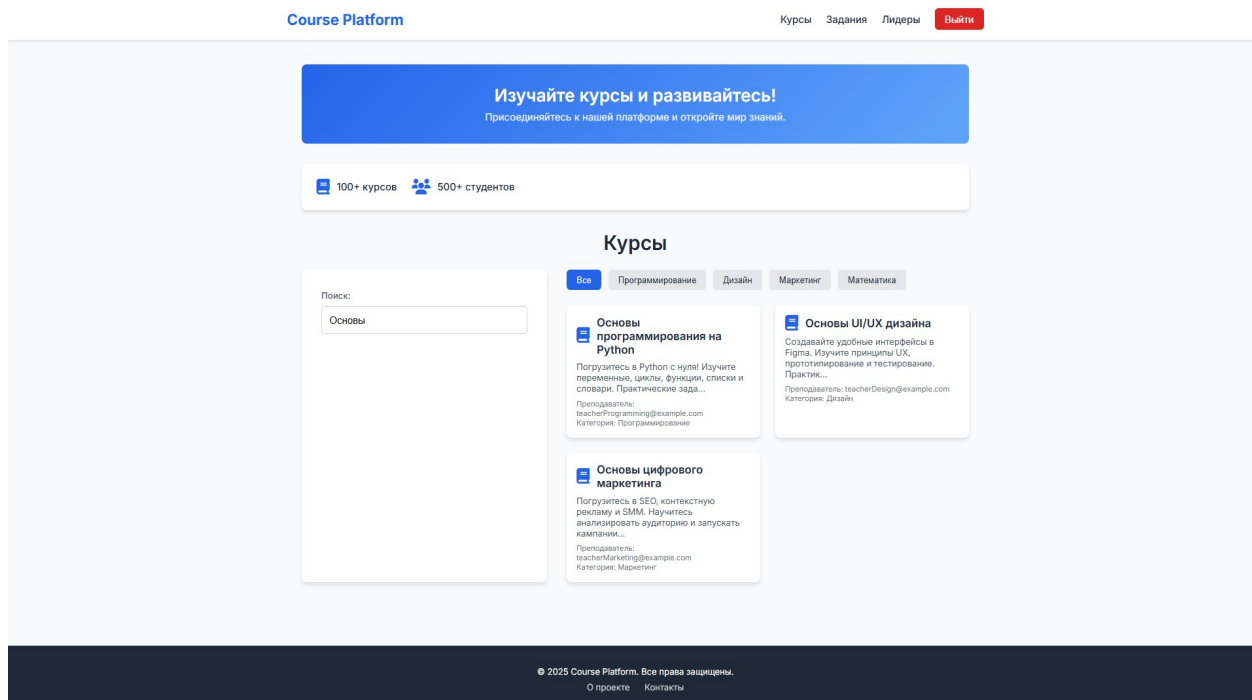


Рисунок 9 – Итог верстки главной страницы

1.5.4 Разработка серверной части

1.5.4.1 Разработка базы данных

Для создания базы данных используется MySQL, управление которой осуществляется через командную строку (MySQL CLI):

Запускаем MySQL CLI, выполнив команду в терминале: `mysql -u root -p`. После ввода пароля (по умолчанию пароль может быть пустым, если не изменён) получаем доступ к серверу MySQL;

Убеждаемся, что сервер MySQL работает, выполнив команду `SHOW DATABASES`, которая отображает список доступных баз данных;

Для создания новой базы данных выполняем команду `CREATE DATABASE course_platform CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci`. Это создаёт базу данных с именем `course_platform` и кодировкой `utf8mb4`, обеспечивающей поддержку широкого набора символов.

Переключаемся на созданную базу данных командой `use course_platform`;

После создания базы данных можно приступать к созданию таблиц, выполняя соответствующие SQL-команды.

Структура созданной базы данных состоит из семи таблиц: users, courses, course_reviews, enrollments, lectures, tasks и task_submissions. Таблица users хранит информацию о пользователях платформы (студентах и преподавателях). Таблица courses содержит данные о курсах, включая их название, описание и преподавателя. Таблица course_reviews предназначена для хранения отзывов пользователей о курсах. Таблица enrollments фиксирует записи студентов на курсы. Таблица lectures хранит информацию о лекциях внутри курсов. Таблица tasks содержит задания, связанные с курсами. Таблица task_submissions хранит ответы студентов на задания.

Схема созданной базы данных представлена на рисунке 10.

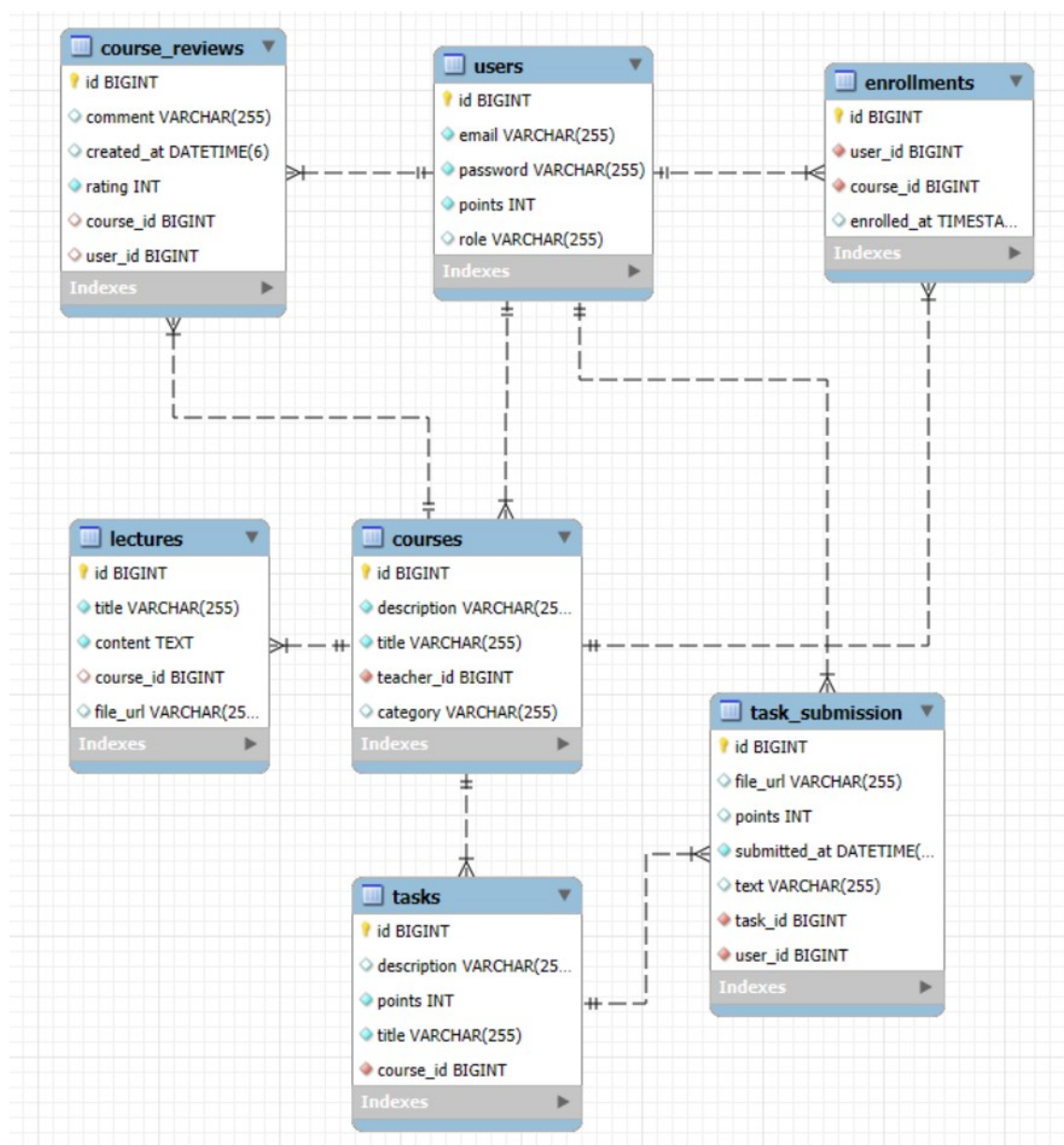


Рисунок 10 – Схема базы данных

База данных спроектирована для обеспечения целостности данных за счёт использования внешних ключей, связывающих таблицы между собой. Каждая таблица имеет первичный ключ, обеспечивающий уникальность записей и оптимизацию запросов. Кодировка utf8mb4 выбрана для поддержки многоязычности, включая кириллицу и специальные символы. Структура таблиц позволяет эффективно управлять образовательным процессом, от записи на курсы до оценки заданий.

1.5.4.2 Реализация серверной логики

Реализация серверной логики платформы курсов представляет собой REST API, разработанный с использованием фреймворка Spring Boot. Код бэкенда организован в виде контроллеров, сервисов и репозиторий, обеспечивающих взаимодействие с базой данных MySQL через JPA. Основной функционал включает регистрацию пользователей, управление курсами, обработку отзывов, запись на курсы, управление лекциями, заданиями и их сдачей, а также формирование таблицы лидеров.

Для обработки запросов на регистрацию используется AuthController, который принимает данные пользователя (email, пароль, роль) через POST-запрос на /api/auth/register. Внутри метода выполняется валидация входных данных: проверяется корректность email с помощью регулярного выражения, длина пароля и допустимость роли (ROLE_STUDENT или ROLE_TEACHER). Если email уже зарегистрирован, возвращается ошибка. Пароль хешируется с использованием BCryptPasswordEncoder, после чего пользователь сохраняется в таблице users.

Управление курсами реализовано в CourseController. При создании курса (POST-запрос на /api/courses) проверяется, что пользователь имеет роль ROLE_TEACHER, а также валидируются поля title и description. Курс привязывается к преподавателю через поле teacher_id и сохраняется в таблице courses. Аналогично, методы для обновления и удаления курсов проверяют, что запрос выполняется владельцем курса. Для добавления лекций (POST-запрос на /api/courses/{id}/lectures) валидируются поля title, content и опциональная ссылка

					ПГУ 1.090301.04.001 ПЗ	Лист
						37
Изм.	Лист	№ докум.	Подпись	Дата		

Запись на курсы осуществляется через POST-запрос на `/api/courses/{id}/enroll`, где проверяется роль `ROLE_STUDENT` и отсутствие предыдущей записи через `EnrollmentRepository`. Данные сохраняются в таблице `enrollments` с уникальным ограничением на пару `user_id` и `course_id`.

Обработка отзывов реализована в `CourseReviewController`. При добавлении отзыва (POST-запрос на `/api/reviews/course/{courseId}`) проверяется аутентификация пользователя, а также валидируется рейтинг (от 1 до 5). Метод возвращает сохранённый отзыв, а GET-запросы позволяют получить список отзывов и средний рейтинг курса.

Управление заданиями реализовано в `TaskController`. Преподаватели создают задания (POST-запрос на `/api/tasks`), указывая курс и проверяя свою принадлежность к нему. Студенты отправляют решения (POST-запрос на `/api/tasks/{id}/submit`), предоставляя текст или ссылку на файл. Проверка уникальности решения обеспечивается ограничением `UNIQUE (user_id, task_id)` в таблице `task_submission`. Преподаватели могут оценивать решения (POST-запрос на `/api/submissions/{submissionId}/grade`), обновляя баллы студента в таблице `users`.

Формирование таблицы лидеров реализовано в `LeaderboardController`. GET-запрос на `/api/users/leaderboard` возвращает список студентов, отсортированных по убыванию баллов, полученных за задания. Данные извлекаются из таблицы `users` с использованием метода `findByRoleOrderByPointsDesc`.

На рисунке 11 представлена UML-диаграмма классов.

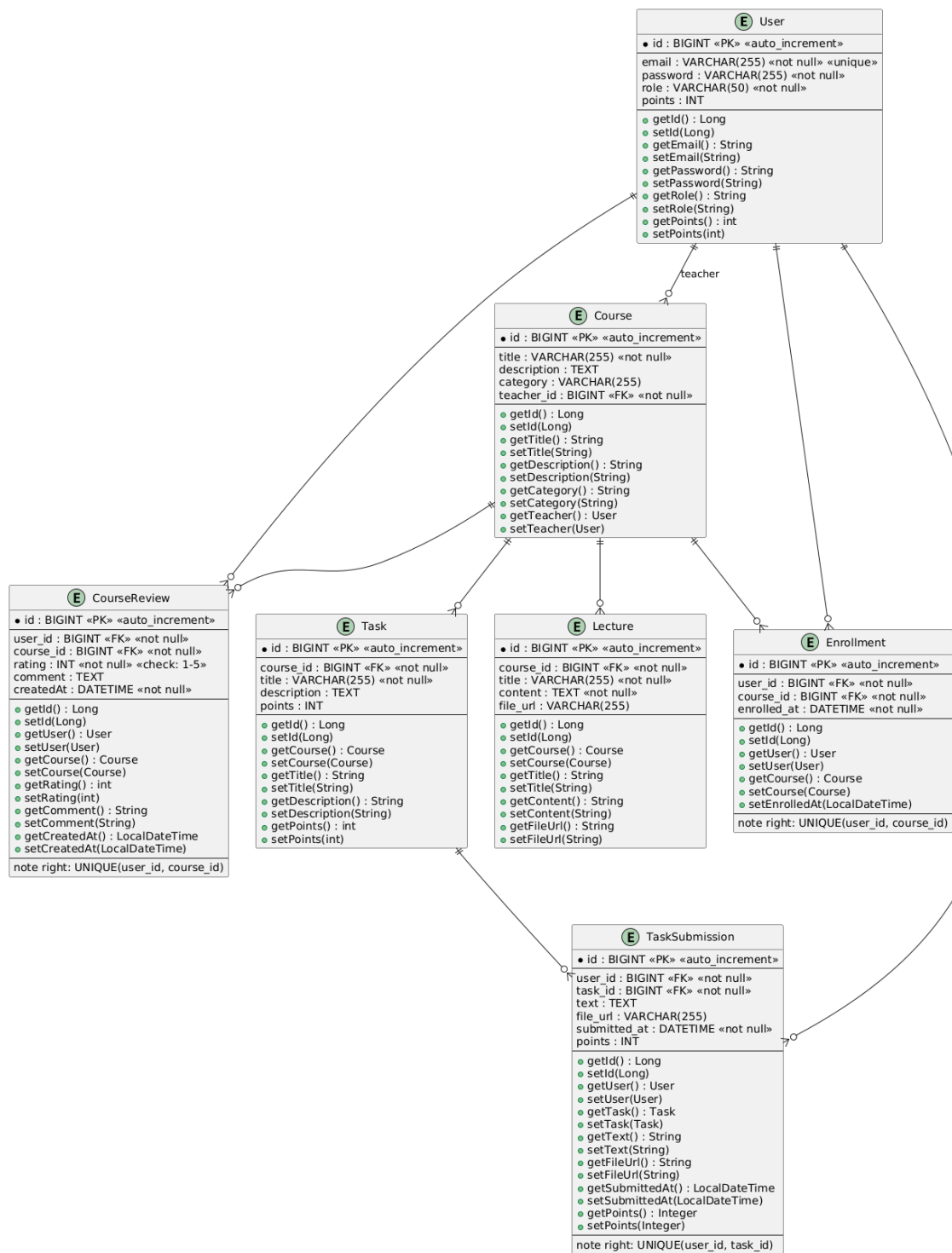


Рисунок 11 – UML-диаграмма классов

Таким образом, серверная логика обеспечивает полный цикл работы образовательной платформы: от регистрации и создания курсов до управления заданиями и формирования рейтинга студентов. Все операции валидируются, а целостность данных поддерживается ограничениями базы данных.

1.6 Описание веб-сервиса

1.6.1 Руководство пользователя

Сервис предлагает наличие таких функций, как:

- регистрация и авторизация;
- создание/просмотр курсов и лекций;
- запись на курсы;
- отзывы на курсы;
- создание/сдача заданий;
- просмотр таблицы лидеров.

На странице с курсами, если пользователь является преподавателем, то при нажатии на кнопку «Войти» и успешной авторизации он переходит в личный кабинет, где может нажать кнопку «Создать курс». В форме преподаватель заполняет поля: «Название курса», «Описание» и «Категория». Также преподаватель может указать дополнительные материалы, загрузив ссылку на файл. Кроме этого, для преподавателя доступно редактирование, удаление созданных курсов и их фильтрация. Функция поиска под формой создания курса позволяет найти нужный среди всех существующих.

На рисунке 12 представлена главная страница веб-сервиса.

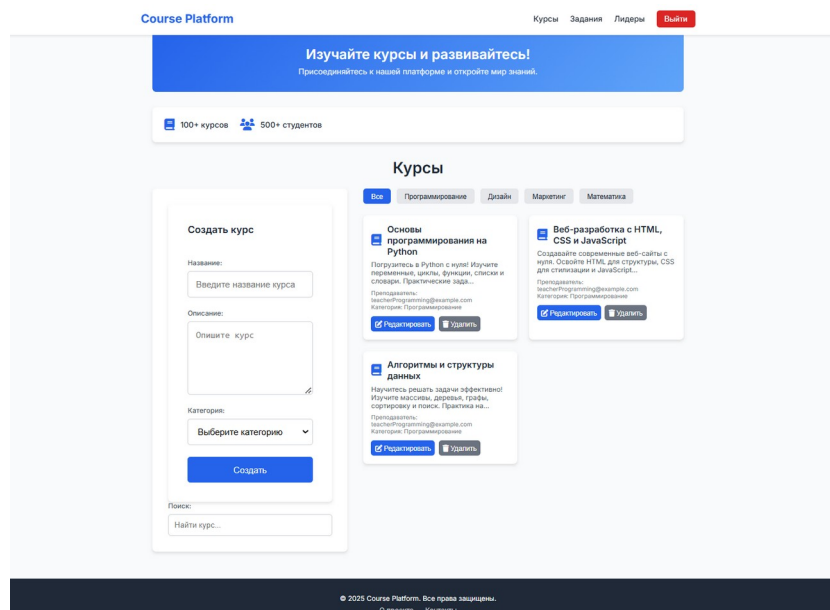


Рисунок 12 – Главная страница

Изм.	Лист	№ докум.	Подпись	Дата

Если пользователь является студентом, он может воспользоваться фильтром на странице курсов, чтобы найти курс по названию или категории. После выбора курса студент нажимает кнопку «Записаться», чтобы присоединиться к курсу, или переходит к деталям курса, где может оставить отзыв, указав рейтинг (от 1 до 5) и комментарий.

На странице заданий отображаются две кнопки: «Создать задание» (для преподавателей) и «Мои задания» (для студентов).

Если пользователь является преподавателем, то при нажатии на кнопку «Создать задание» открывается модальное окно с формой создания задания. В форме преподаватель заполняет поля: «Название задания», «Описание» и «Максимальные баллы». Также можно указать курс, к которому относится задание, выбрав его из выпадающего списка.

После полного заполнения формы преподаватель нажимает кнопку «Отправить». Созданное задание отображается на странице заданий курса, доступной всем записанным студентам и преподавателю. На рисунке 13 представлена страница со списком заданий.

Course Platform Курсы Задания Лидеры Выйти

Задания

Поиск по названию... Все курсы + Создать задание

Название ↑	Курс	Очки	Действия
Написать функцию подсчёта слов	Основы программирования на Python	4	Решения
Реализовать стек на Python	Алгоритмы и структуры данных	6	Решения
Создать интерактивный to-do list	Веб-разработка с HTML, CSS и JavaScript	5	Решения

Решения: Написать функцию подсчёта слов

Студент: studentTest@example.com
 Здравствуйте, прикрепляю ссылку на решение. В первом архиве проект, а во втором листинг.
[Открыть ссылку](#)
 Баллы: 4
 Оценка (баллы):
 Введите баллы
[Оценить](#)

Рисунок 13 – Страница со списком заданий преподавателя

Если пользователь является студентом, он может воспользоваться фильтром на странице заданий, чтобы найти задание по названию или курсу. После выбора задания студент нажимает кнопку «Отправить решение», открывая форму, где можно ввести текст для ответа или указать ссылку на файл, как показано на рисунке 14.

Course Platform

КурсыЗаданияЛидерыВыйти

Задания

✓ Ваш прогресс

Вы выполнили 0 из 2 заданий

Поиск по названию...

Все курсы

Все статусы

Название ↑	Курс	Очки	Статус	Действия
Дизайн рекламного баннера	Графический дизайн в Adobe Photoshop	5	Не отправлено	Решить
Настроить рекламную кампанию	Основы цифрового маркетинга	4	Не отправлено	Решить

Дизайн рекламного баннера

Решение:

Введите решение

Ссылка:

Введите ссылку (например, Google Drive)

Отправить

Настроить рекламную кампанию

Решение:

Введите решение

Ссылка:

Введите ссылку (например, Google Drive)

Отправить

Рекомендуемые задания

Введение в Python

Изучите основы программирования на Python.

Курс: Программирование

UI/UX Дизайн

Создайте прототип интерфейса.

Курс: Дизайн

© 2025 Course Platform. Все права защищены.

О проектеКонтакты

Рисунок 14 – Страница с заданиями студента

Страница таблицы лидеров представляет собой список студентов, где для каждого указаны email, количество баллов и место в рейтинге. Список формируется автоматически на основе данных из таблицы users, с сортировкой по полю points в порядке убывания.

На рисунке 15 показана страница со списком лидеров.

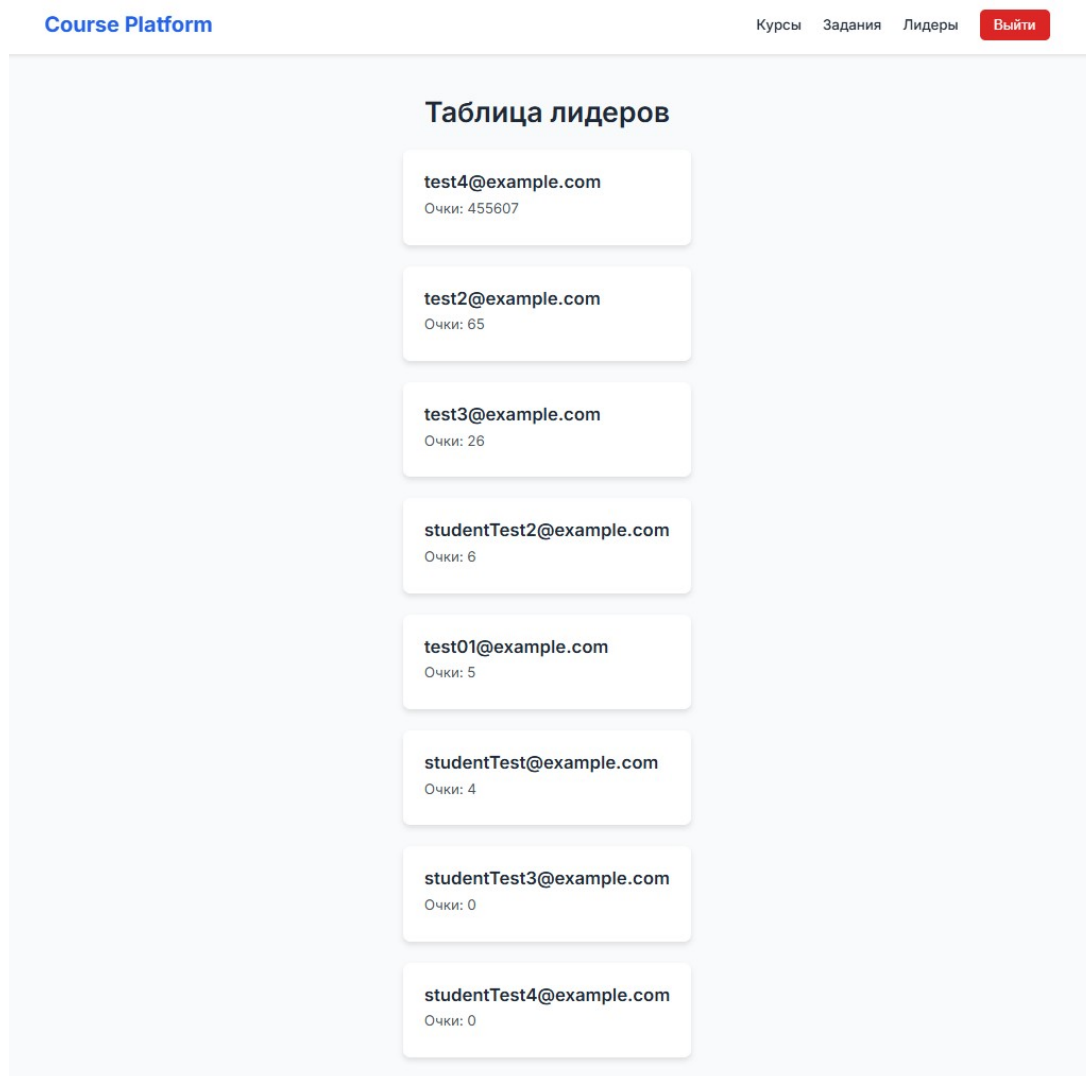


Рисунок 15 – Таблица лидеров

Созданный проект продемонстрировал стабильную работу всех функциональных компонентов. Пользователи успешно регистрируются и авторизуются, курсы и лекции создаются преподавателями, а студенты записываются на курсы выполняют задания и получают оценки. Отзывы на курсы добавляются корректно, таблица лидеров отображает рейтинг студентов.

2 Безопасность жизнедеятельности

В наше время работа за компьютером стала неотъемлемой частью жизни многих людей, что приводит к появлению различных проблем со здоровьем, таких как напряжение глаз, неправильная осанка, туннельный синдром и других заболеваний. В особенности это касается людей чья профессиональная деятельность связана с долгим нахождением перед экраном монитора. Для того, чтобы обезопасить себя от проблем продолжительной работы за компьютером, необходимо соблюдать ряд правил. В этой главе мы рассмотрим необходимые меры, при соблюдении которых, можно предотвратить все возможные проблемы, связанные с долгой работой за компьютером.

2.1 Требования к организации рабочего места

При организации рабочего места необходимо соблюдать ряд рекомендаций, которые помогут минимизировать риски для здоровья. Рассмотрим список предлагаемых рекомендаций:

- для комфортной работы за компьютером важно приобрести хорошее и качественное кресло, которое обладает возможностью регулировки высоты и угла наклона, и необходимой формой для спины и комплекции человека;
- рабочий стол должен быть на такой высоте, чтобы руки находились под углом 90 градусов при работе с клавиатурой и мышью;
- чтобы предотвратить проблемы со зрением, монитор должен находиться на комфортном расстоянии от глаз. Рекомендуемым значением является 50-70 см от монитора. Положение монитора должно быть такое чтобы, верхняя часть экрана была на уровне глаз.

Пример соблюдения всех правил можно увидеть на рисунке 16.



Рисунок 16 – Правильное положение при работе за компьютером

2.2 Требования к освещению

Одна из главных проблем, которая возникает при долгой работе за компьютером – это вред для глаз. Неправильное освещение помещения и настроек яркости монитора, может привести к усталости глаз, сухости, покраснению, размытости зрения и головным болям. Чтобы предотвратить это, необходимо соблюдать требования к освещению в помещении.

Рекомендуется, по возможности, использовать естественный свет, при этом избегать попадания прямых солнечных лучей на монитор. Естественный свет снижает напряжение глаз и улучшает общее самочувствие.

Для работы в вечернее время рекомендуется использовать искусственное освещение, такие как настольная лампа, комнатное освещение и т.д.

Предпочтительно комбинировать освещения для более лучшего эффекта. Освещение должно быть ярким, чтобы избежать напряжения глаз, но не слишком

интенсивным, чтобы исключить появления бликом на экране монитора.

Также нужно учитывать правильное расположение источников света. Локальный источник, например настольная лампа, должен быть установлен сбоку от монитора, чтобы не создавать прямых бликов на экране.

Настройка монитора. Правильная настройка яркости и контрастности монитора, также может уменьшить напряжение глаз при длительной работе за ПК. Рекомендуется в вечернее время использовать ночной режим, который делает цвета экрана более теплыми. Это поможет уменьшить нагрузка на глаза, а также улучшить качество сна.

2.3 Требования к режиму отдыха

Чтобы избежать проблем со здоровьем при длительной работе за компьютером, необходимо ежечасно делать небольшой перерыв, для выполнения разминки, чтобы избежать мышечного напряжения и улучшить кровообращение в суставах и мышцах. Не стоит забывать и про глаза, которые так же подвержены воздействию, поэтому будет полезно сделать зарядку, которая включает в себя круговые движения открытыми глазами, смена точки фокусировки взгляда с близкой до дальней. Важно также соблюдать строгий график работы и отдыха, ведь постоянный труд может привести к негативным последствиям. Необходимо составить четкое расписание и расписать план работы, для оптимизации своего времени. Не следует работать в дни отдыха, переработки могут плохо сказаться на вашем физическом и ментальном здоровье. При соблюдение данных рекомендаций можно обеспечить комфортную работу с минимальным риска для своего здоровья.

Заключение

В ходе выполнения выпускной квалификационной работы была разработана образовательная платформа в формате веб-сервиса для управления курсами и обучения студентов.

Разработанное веб-приложение предоставляет удобный интерфейс для регистрации и авторизации пользователей, создания и управления курсами, добавления лекций, записи на курсы, написания отзывов, выполнения заданий и просмотра таблицы лидеров студентов по баллам. Пользователи с ролью преподавателя могут создавать курсы и задания, а студенты — записываться на курсы, сдавать задания и оставлять отзывы.

Была разработана структура базы данных для хранения информации о пользователях, курсах, лекциях, отзывах, записях, заданиях и их решениях. Данная структура реализована в формате реляционной СУБД.

Результатом работы веб-сервиса является предоставление пользователю функционала для полного цикла обучения: от создания курса преподавателем до оценки заданий и формирования рейтинга студентов. Одним из достоинств разработанного приложения является наличие интуитивно понятного интерфейса.

Выполненные отладка и тестирование REST API, включая запросы на регистрацию, управление курсами, отзывами и заданиями, доказали работоспособность и корректность работы приложения.

В дальнейшем можно усовершенствовать разработанное веб-приложение, добавив функцию уведомлений о новых заданиях и оценках, чат для общения преподавателя с группой студентов. Улучшение графического интерфейса может включать отображение статистики прохождения курсов, включая процент завершения и средний рейтинг.

Список использованных источников

1. Официальная документация MySQL: [Электронный ресурс]. URL: <https://dev.mysql.com/doc/> (дата обращения: 30.04.2025).
2. Васильев, А.Н. Java. Объектно-ориентированное программирование: Учебное пособие: для магистров и бакалавров. Базовый курс по объектно-ориентированному программированию / А.Н. Васильев. – СПб.: Питер, 2013. – 400 с.
3. Арнольд К., Гослинг Д. Язык программирования Java. -СПб.: Питер, 1997. – 304с
4. Давыдов, С. IntelliJ IDEA. Профессиональное программирование на Java / С. Давыдов. – СПб.: BHV, 2005. – 800 с
5. Гончаров А.Ю. Web-дизайн: HTML, JavaScript и CSS. Карманный справочник – М.: КУДИЦ-ПРЕСС, 2007. – 320с.
6. Документация к фреймворку Spring Framework. [Электронный ресурс] / Режим доступа: <https://docs.spring.io/spring-framework/docs/4.2.x/spring-frameworkreference/html>, свободный. (дата обращения 04.05.2025).
7. Исси Коэн, Л. Полный справочник по HTML, CSS и JavaScript. – М.: ЭКОМ Паблишерз – 2011. – 1168с.
8. Дакетт Джон HTML и CSS. Разработка и дизайн веб-сайтов (CDROM); Эксмо – Москва, 2013. – 480 с.
9. Осипов Д. Л. Технологии проектирования баз данных. – М.: «ДМК Пресс», 2019. – С. 135-215.
10. Ефремова О. С. Требования охраны труда при работе на персональных электронно-вычислительных машинах. – М.: Альфа-Пресс, 2008.

Приложение А

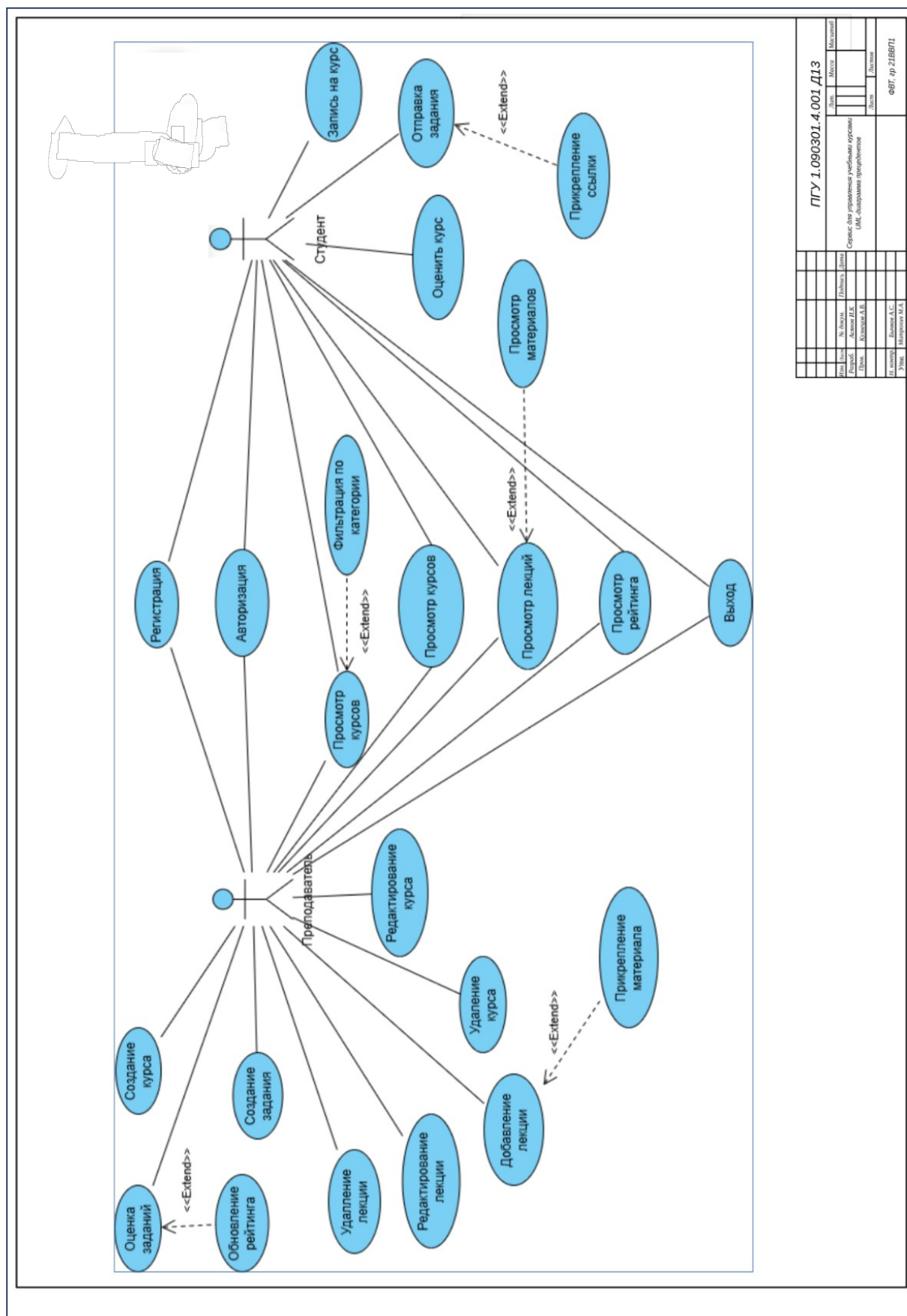
(обязательное)

Сервис для управления учебными курсами

UML-диаграммы

					ПГУ 1.090301.04.001 ПЗ	Лист
						48
Изм.	Лист	№ докум.	Подпись	Дата		

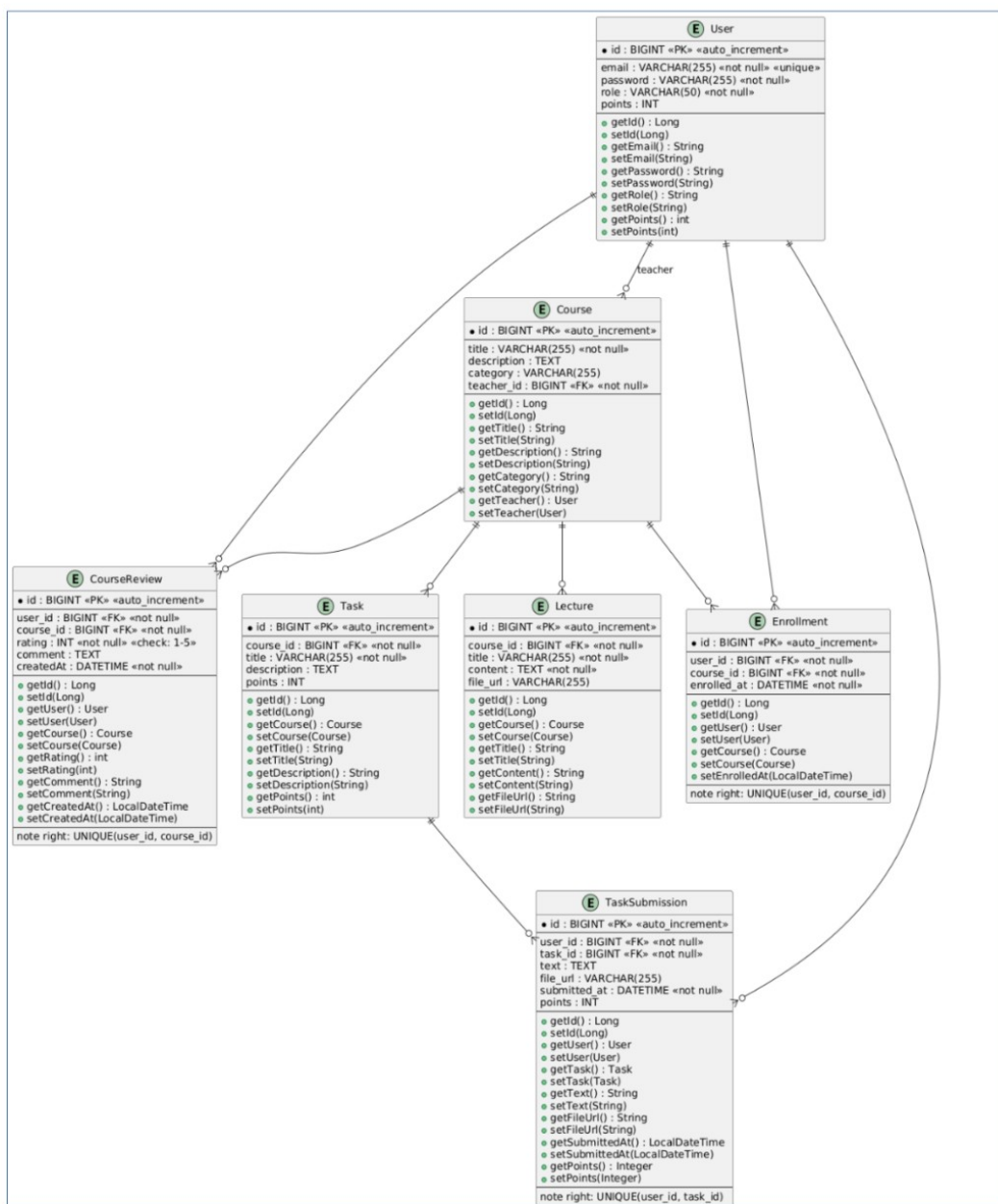
Приложение А.1 – Диаграмма прецедентов



--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Изм.	Лист	№ докум.	Подпись	Дата

Приложение А.2 – Диаграмма классов



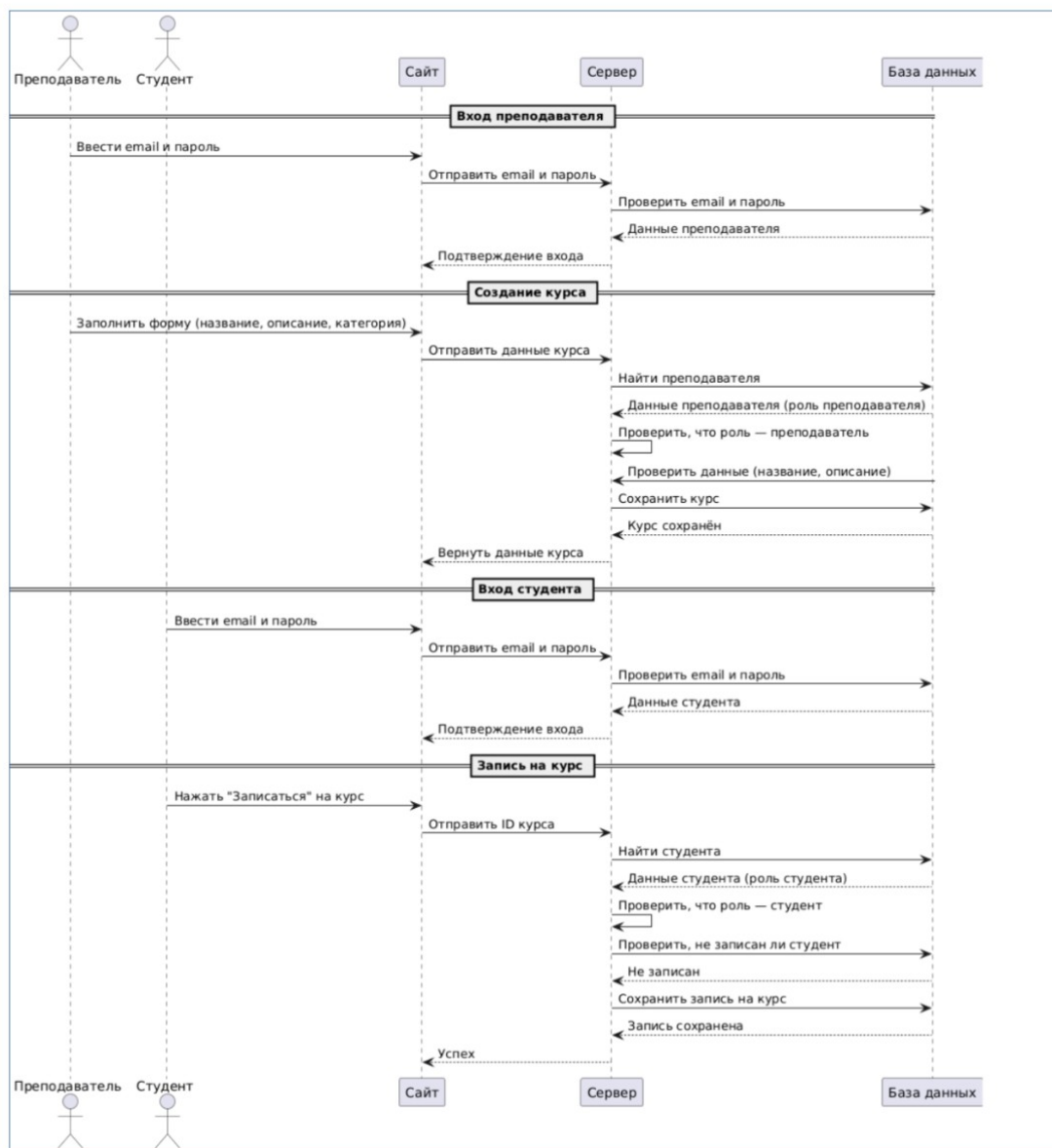
ПГУ 1.090301.4.002 Д11				
Имя	Фамилия	Инициалы	Подпись	Дата
Решен	Дополн. Р.Р.	Прим.	Визирован	В.В.
И.И.И.	К.К.К.	Л.Л.Л.	М.М.М.	Н.Н.Н.
В.В.В.	Г.Г.Г.	Д.Д.Д.	З.З.З.	И.И.И.

Сервис для управления учебными курсами
УМ-Диагностика классов

Лист 1 Листов

ФВТ, стр 2188/11

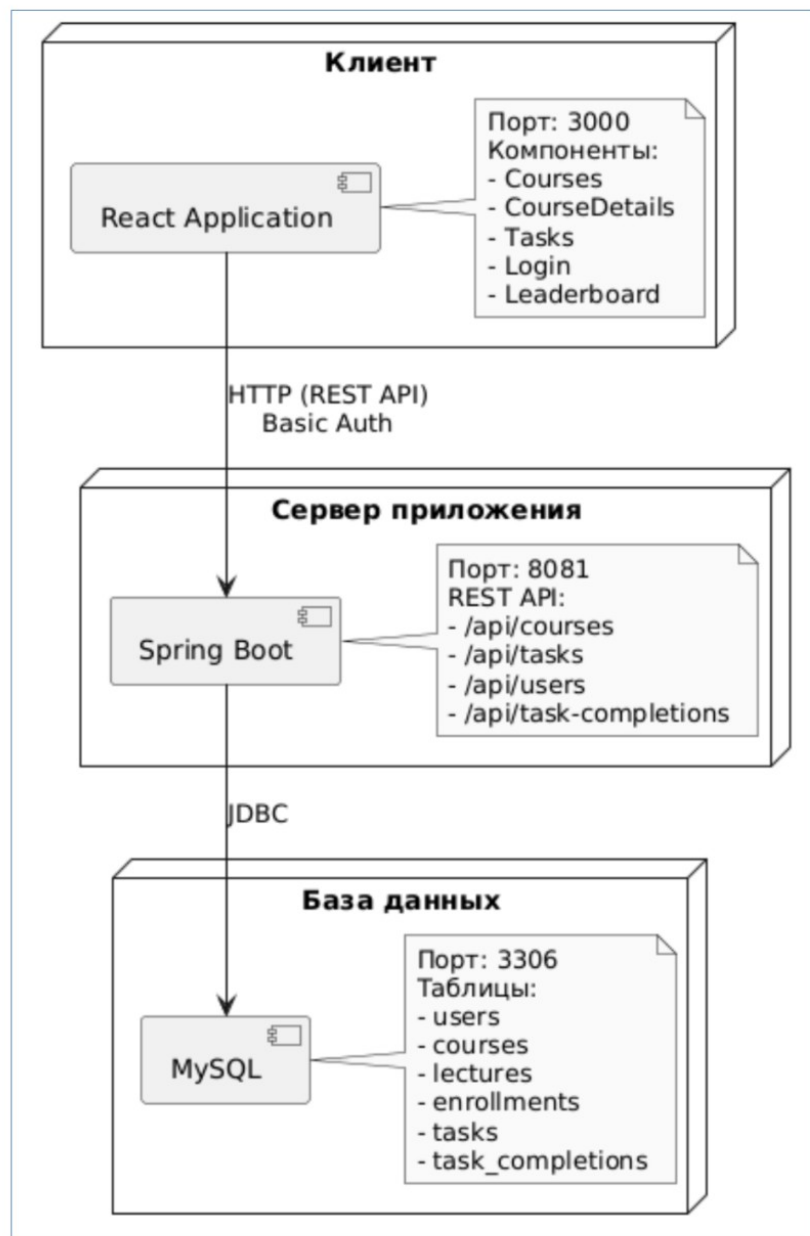
Приложение А.3 – Диаграмма последовательностей



					ПГУ 1.090301.4.003 Д14		
Имя	Фамилия	№ документа	Подпись	Дата	Сервис для управления учебными курсами (UML-диаграмма последовательности)		
Рязань	Александр В.Б.						
Почта	Евдокимов А.В.				Дата	Место	Масштаб
И. номер	Булочков А.С.				ФВТ, зр 21ВВ/П1		
Уч. номер	Морозов М.А.						

Изм.	Лист	№ докум.	Подпись	Дата

Приложение А.4 – Диаграмма последовательностей



ПГУ 1.090301.4.004 Д19									
Имя	Фамилия	Имя	Фамилия	Подпись	Дата	Сервис для управления учебными курсами			
Лектор	Александр П.К.	Имя				Имя			
Преод.	Евгений А.В.	Лектор				Лектор			
М. преподав.	Васильев А.С.	М. преподав.				М. преподав.			
Учен.	Михайлов М.А.	Учен.				Учен.			

ФБТ, стр 216В/П1

Изм.	Лист	№ докум.	Подпись	Дата

Приложение Б

(обязательное)

Сервис для управления учебными курсами

Листинг программы

					ПГУ 1.090301.04.001 ПЗ	Лист
						53
Изм.	Лист	№ докум.	Подпись	Дата		

SecurityConfig.java

```
package com.example.courseplatform.config;

import com.example.courseplatform.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.web.builders.HttpSecu
rity;
import
org.springframework.security.config.annotation.web.configuration.Ena
bleWebSecurity;
import
org.springframework.security.core.userdetails.UserDetailsService;
import
org.springframework.security.core.userdetails.UsernameNotFoundExcept
ion;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.CorsConfigurationSource;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
import org.springframework.http.HttpMethod;
import java.util.Arrays;

@Configuration
@EnableWebSecurity
public class SecurityConfig {
    @Autowired
    private UserRepository userRepository;
```

					ПГУ 1.090301.04.001 ПЗ	Лист
						54
Изм.	Лист	№ докум.	Подпись	Дата		

@Bean

```
public SecurityFilterChain securityFilterChain(HttpSecurity http)
throws Exception {
    http
        .cors(cors ->
cors.configurationSource(corsConfigurationSource()))
        .csrf(csrf -> csrf.disable())
        .authorizeHttpRequests(auth -> auth
            .requestMatchers(HttpMethod.OPTIONS,
"/**").permitAll()
            .requestMatchers(HttpMethod.POST,
"/api/auth/register").permitAll()
            .requestMatchers(HttpMethod.POST,
"/api/courses").hasRole("TEACHER")
            .requestMatchers(HttpMethod.PUT,
"/api/courses/**").hasRole("TEACHER")
            .requestMatchers(HttpMethod.DELETE,
"/api/courses/**").hasRole("TEACHER")
            .requestMatchers(HttpMethod.POST,
"/api/courses/*/lectures").hasRole("TEACHER")
            .requestMatchers(HttpMethod.PUT,
"/api/courses/*/lectures/*").hasRole("TEACHER")
            .requestMatchers(HttpMethod.DELETE,
"/api/courses/*/lectures/*").hasRole("TEACHER")
            .requestMatchers(HttpMethod.POST,
"/api/courses/*/enroll").hasRole("STUDENT")
            .requestMatchers(HttpMethod.GET, "/api/courses",
"/api/courses/**").hasAnyRole("TEACHER", "STUDENT")
            .requestMatchers(HttpMethod.POST,
"/api/tasks").hasRole("TEACHER").requestMatchers(HttpMethod.GET,
"/api/tasks").hasAnyRole("TEACHER",
"STUDENT").requestMatchers(HttpMethod.POST,
"/api/tasks/complete").hasRole("STUDENT")
```

					ПГУ 1.090301.04.001 ПЗ	Лист
						55
Изм.	Лист	№ докум.	Подпись	Дата		

```

.requestMatchers(HttpMethod.GET,
"/api/users/leaderboard").hasAnyRole("TEACHER", "STUDENT")
        .requestMatchers(HttpMethod.GET,
"/api/users/me").hasAnyRole("TEACHER", "STUDENT")
        .anyRequest().authenticated()
    )
    .httpBasic(httpBasic -> httpBasic
        .authenticationEntryPoint((request, response,
authException) -> {
            response.setStatus(401);
            response.setContentType("application/json");
            response.getWriter().write("{\"error\":
\"Unauthorized\"}");
        })
    );
return http.build();
@Bean
public CorsConfigurationSource corsConfigurationSource() {
    CorsConfiguration configuration = new CorsConfiguration();

configuration.setAllowedOrigins(Arrays.asList("http://localhost:3000
"));
    configuration.setAllowedMethods(Arrays.asList("GET", "POST",
"PUT", "DELETE", "OPTIONS"));
    configuration.setAllowedHeaders(Arrays.asList("Authorization",
"Content-Type"));
    configuration.setAllowCredentials(true);
    UrlBasedCorsConfigurationSource source = new
    UrlBasedCorsConfigurationSource();
    source.registerCorsConfiguration("/**", configuration);
    return source;
}
@Bean

```

```

public UserDetailsService userDetailsService() {
    return username -> userRepository.findByEmail(username)
        .map(user ->
org.springframework.security.core.userdetails.User.builder()
            .username(user.getEmail())
            .password(user.getPassword())
            .roles(user.getRole().startsWith("ROLE_") ?
user.getRole().substring(5) : user.getRole())
            .build())
        .orElseThrow(() -> new
UsernameNotFoundException("Пользователь не найден: " + username));
    }
@Bean(name = "passwordEncoder")
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}

```

AuthController.java

```

package com.example.courseplatform.controller;
import com.example.courseplatform.model.User;
import com.example.courseplatform.repository.UserRepository;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.server.ResponseStatusException
import java.util.Map;
@RestController
@RequestMapping("/api/auth")
public class AuthController {

```

```

        private static final Logger logger =
LoggerFactory.getLogger(AuthController.class);

@Autowired
private UserRepository userRepository;

@Autowired
private PasswordEncoder passwordEncoder;

@PostMapping("/register")
public void register(@RequestBody Map<String, String> request) {
    logger.debug("Запрос на регистрацию: {}", request.get("email"));
    try {
        String email = request.get("email");
        String password = request.get("password");
        String role = request.get("role");

        // Валидация входных данных
        if (email == null || email.trim().isEmpty() || !
email.matches("^[\\w-\\.]+@([\\w-]+\\.){2,4}$")) {
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST, "Введите корректный
email");
        }
        if (password == null || password.length() < 6) {
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST, "Пароль должен быть
не короче 6 символов");
        }
        if (role == null || (!role.equals("ROLE_STUDENT") && !
role.equals("ROLE_TEACHER"))) {
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST, "Роль должна быть
ROLE_STUDENT или ROLE_TEACHER");
        }
    }
}

```

					ПГУ 1.090301.04.001 ПЗ	Лист
						58
Изм.	Лист	№ докум.	Подпись	Дата		

```

        // Проверка уникальности email
        if (userRepository.findByEmail(email).isPresent()) {
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST, "Email уже
зарегистрирован");
        }

        // Создание пользователя
        User user = new User();
        user.setEmail(email);
        user.setPassword(passwordEncoder.encode(password));
        user.setRole(role);
        user.setPoints(0);

        userRepository.save(user);
        logger.info("Пользователь зарегистрирован: {}", email);
    } catch (Exception e) {
        logger.error("Ошибка при регистрации: {}",
e.getMessage(), e);
        throw new
ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "Ошибка
при регистрации: " + e.getMessage(), e);
    }
}
}

```

CourseController.java

```

package com.example.courseplatform.controller;

import com.example.courseplatform.model.Course;
import com.example.courseplatform.model.Lecture;
import com.example.courseplatform.model.User;
import com.example.courseplatform.model.Enrollment;

```

```

import com.example.courseplatform.repository.CourseRepository;
import com.example.courseplatform.repository.LectureRepository;
import com.example.courseplatform.repository.UserRepository;
import com.example.courseplatform.repository.EnrollmentRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import
org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.bind.annotation.*;

import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

@RestController
@RequestMapping("/api/courses")
public class CourseController {

    @Autowired
    private CourseRepository courseRepository;

    @Autowired
    private LectureRepository lectureRepository;

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private EnrollmentRepository enrollmentRepository;
    @GetMapping

```

					ПГУ 1.090301.04.001 ПЗ	Лист
						60
Изм.	Лист	№ докум.	Подпись	Дата		

```

public List<Map<String, Object>> getAllCourses() {
    System.out.println("Fetching all courses");
    String currentUserEmail =
SecurityContextHolder.getContext().getAuthentication().getName();
    User user = userRepository.findByEmail(currentUserEmail)
        .orElseThrow(() -> new RuntimeException("Пользователь не
найден: " + currentUserEmail));
    List<Course> courses;

    if ("ROLE_TEACHER".equals(user.getRole())) {
        System.out.println("Filtering courses for teacher: " +
currentUserEmail);
        courses = courseRepository.findByTeacherId(user.getId());
    } else {
        System.out.println("Returning all courses for user: " +
currentUserEmail);
        courses = courseRepository.findAll();
    }
    List<Map<String, Object>> response = new ArrayList<>();
    for (Course course : courses) {
        Map<String, Object> courseMap = new HashMap<>();
        courseMap.put("id", course.getId());
        courseMap.put("title", course.getTitle());
        courseMap.put("description", course.getDescription());
        courseMap.put("category", course.getCategory()); //
Добавлено
        Map<String, Object> teacherMap = new HashMap<>();
        teacherMap.put("email", course.getTeacher().getEmail());
        courseMap.put("teacher", teacherMap);
        response.add(courseMap);
    }
    System.out.println("Returning courses: " + response);
    return response;
}

```

```

@PostMapping
@PreAuthorize("hasRole('TEACHER')")
public ResponseEntity<Course> createCourse(@RequestBody Map<String,
String> request) {
    try {
        System.out.println("Creating course: " +
request.get("title"));
        String title = request.get("title");
        String description = request.get("description");
        String category = request.get("category"); // Добавлено

        if (title == null || title.length() < 3) {
            throw new IllegalArgumentException("Title must be at
least 3 characters long");
        }
        if (description == null || description.length() < 10) {
            throw new IllegalArgumentException("Description must be
at least 10 characters long");
        }

        String username =
SecurityContextHolder.getContext().getAuthentication().getName();
        User teacher = userRepository.findByEmail(username)
            .orElseThrow(() -> new RuntimeException("Teacher not
found"));
        Course course = new Course();
        course.setTitle(title);
        course.setDescription(description);
        course.setCategory(category); // Добавлено
        course.setTeacher(teacher);
        System.out.println("Saving course to database");
        Course savedCourse = courseRepository.save(course);
        System.out.println("Course saved with ID: " + savedCourse.getId());
    }
}

```

```

return ResponseEntity.ok(savedCourse);
    } catch (IllegalArgumentException e) {
        System.err.println("Validation error: " + e.getMessage());
        throw new RuntimeException("Validation failed: " +
e.getMessage());
    } catch (Exception e) {
        System.err.println("Course creation error: " +
e.getMessage());
        e.printStackTrace();
        throw new RuntimeException("Failed to create course: " +
e.getMessage());
    }
}

@GetMapping("/{id}")
public Map<String, Object> getCourseById(@PathVariable Long id) {
    System.out.println("Fetching course ID: " + id);
    Course course = courseRepository.findById(id)
        .orElseThrow(() -> new RuntimeException("Course not
found"));
    String currentUserEmail =
SecurityContextHolder.getContext().getAuthentication().getName();
    User user =
userRepository.findById(currentUserEmail).orElse(null);
    List<Lecture> lectures = new ArrayList<>();
    boolean isEnrolled = false;
    boolean isCourseOwner = user != null &&
course.getTeacher().getEmail().equals(currentUserEmail);
    if (user != null) {
        isEnrolled =
enrollmentRepository.existsByUserIdAndCourseId(user.getId(), id);
        if (isEnrolled || isCourseOwner) {
            lectures = lectureRepository.findById(id);
        }
    }
}

```

```

List<Map<String, Object>> lectureData = new ArrayList<>();
for (Lecture lecture : lectures) {
    Map<String, Object> lectureMap = new HashMap<>();
    lectureMap.put("id", lecture.getId());
    lectureMap.put("title", lecture.getTitle());
    lectureMap.put("content", lecture.getContent());
    lectureMap.put("fileUrl", lecture.getFileUrl());
    lectureData.add(lectureMap);
}

Map<String, Object> response = new HashMap<>();
response.put("id", course.getId());
response.put("title", course.getTitle());
response.put("description", course.getDescription());
response.put("category", course.getCategory()); // Добавлено
Map<String, Object> teacherMap = new HashMap<>();
teacherMap.put("email", course.getTeacher().getEmail());
response.put("teacher", teacherMap);
response.put("lectures", lectureData);
response.put("isEnrolled", isEnrolled);

return response;
}

@PutMapping("/{id}")
@PreAuthorize("hasRole('TEACHER')")

public ResponseEntity<Course> updateCourse(@PathVariable Long id,
@RequestBody Map<String, String> request) {
    try {
        System.out.println("Updating course ID: " + id);
        Course course = courseRepository.findById(id)
            .orElseThrow(() -> new RuntimeException("Course not found"));
    }
}

```

```

        String currentUserEmail =
SecurityContextHolder.getContext().getAuthentication().getName();
        if (!
course.getTeacher().getEmail().equals(currentUserEmail)) {
            throw new RuntimeException("You are not authorized to
edit this course");
        }

        String title = request.get("title");
        String description = request.get("description");
        String category = request.get("category"); // Добавлено
        if (title == null || title.length() < 3) {
            throw new IllegalArgumentException("Title must be at
least 3 characters long");
        }
        if (description == null || description.length() < 10) {
            throw new IllegalArgumentException("Description must be
at least 10 characters long");
        }

        course.setTitle(title);
        course.setDescription(description);
        course.setCategory(category); // Добавлено
        System.out.println("Saving updated course: " + title);
        Course updatedCourse = courseRepository.save(course);
        System.out.println("Course updated with ID: " +
updatedCourse.getId());
        return ResponseEntity.ok(updatedCourse);
    } catch (IllegalArgumentException e) {
        System.err.println("Validation error: " + e.getMessage());
        throw new RuntimeException("Validation failed: " +
e.getMessage());
    }

```

```

    } catch (Exception e) {
        System.err.println("Course update error: " +
e.getMessage());
        e.printStackTrace();
        throw new RuntimeException("Failed to update course: " +
e.getMessage());
    }
}

public ResponseEntity<Void> deleteCourse(@PathVariable Long id) {
    try {
        System.out.println("Deleting course ID: " + id);
        Course course = courseRepository.findById(id)
            .orElseThrow(() -> new RuntimeException("Course not
found"));
        String currentUserEmail =
SecurityContextHolder.getContext().getAuthentication().getName();
        if (!
course.getTeacher().getEmail().equals(currentUserEmail)) {
            throw new RuntimeException("You are not authorized to
delete this course");
        }
        System.out.println("Deleting course and associated
lectures");
        courseRepository.delete(course);
        System.out.println("Course deleted: " + id);
        return ResponseEntity.noContent().build();
    } catch (Exception e) {
        System.err.println("Course deletion error: " +
e.getMessage());
        e.printStackTrace();
        throw new RuntimeException("Failed to delete course: " +
e.getMessage());
    }
}

```

```

@PostMapping("/{id}/lectures")
@PreAuthorize("hasRole('TEACHER')")
public ResponseEntity<Lecture> createLecture(@PathVariable Long id,
@RequestBody Map<String, String> request) {
    try {
        System.out.println("Creating lecture for course ID: " + id);
        Course course = courseRepository.findById(id)
            .orElseThrow(() -> new RuntimeException("Course not
found"));

        String currentUserEmail =
SecurityContextHolder.getContext().getAuthentication().getName();
        if (!
course.getTeacher().getEmail().equals(currentUserEmail)) {
            throw new RuntimeException("You are not authorized to
add lectures to this course");
        }

        String title = request.get("title");
        String content = request.get("content");
        String fileUrl = request.get("fileUrl");
        if (title == null || title.length() < 3) {
            throw new IllegalArgumentException("Title must be at
least 3 characters long");
        }
        if (content == null || content.length() < 10) {
            throw new IllegalArgumentException("Content must be at
least 10 characters long");
        }
        if (fileUrl != null &&
!fileUrl.matches("^(https?|ftp)://[^\s/$.?\#].[^\\s]*$")) {
            throw new IllegalArgumentException("Invalid URL format
for file");
        }
    }
}

```

```

Lecture lecture = new Lecture();
    lecture.setTitle(title);
    lecture.setContent(content);
    lecture.setCourse(course);
    lecture.setFileUrl(fileUrl);

    System.out.println("Saving lecture to database");
    Lecture savedLecture = lectureRepository.save(lecture);
    System.out.println("Lecture saved with ID: " +
savedLecture.getId());
    return ResponseEntity.ok(savedLecture);
} catch (IllegalArgumentException e) {
    System.err.println("Validation error: " + e.getMessage());
    throw new RuntimeException("Validation failed: " +
e.getMessage());
} catch (Exception e) {
    System.err.println("Lecture creation error: " +
e.getMessage());
    e.printStackTrace();
    throw new RuntimeException("Failed to create lecture: " +
e.getMessage());
}
}
@PutMapping("/{courseId}/lectures/{lectureId}")
@PreAuthorize("hasRole('TEACHER')")
public ResponseEntity<Lecture> updateLecture(
    @PathVariable Long courseId,
    @PathVariable Long lectureId,
    @RequestBody Map<String, String> request) {
    try {
System.out.println("Updating lecture ID: " + lectureId + " for
course ID: " + courseId);
Course course = courseRepository.findById(courseId)

```

```

.orElseThrow(() -> new RuntimeException("Course not found"));

String currentUserEmail =
SecurityContextHolder.getContext().getAuthentication().getName();
if (!course.getTeacher().getEmail().equals(currentUserEmail)) {
    throw new RuntimeException("You are not authorized to edit this
lecture");
}

Lecture lecture = lectureRepository.findById(lectureId)
    .orElseThrow(() -> new RuntimeException("Lecture not
found"));

if (!lecture.getCourse().getId().equals(courseId)) {
    throw new RuntimeException("Lecture does not belong to this
course");
}

String title = request.get("title");
String content = request.get("content");
String fileUrl = request.get("fileUrl");
if (title == null || title.length() < 3) {
    throw new IllegalArgumentException("Title must be at least 3
characters long");
}
if (content == null || content.length() < 10) {
    throw new IllegalArgumentException("Content must be at least 10
characters long");
}
if (fileUrl != null && !fileUrl.matches("^(https?|ftp)://[^\s/$.?\#].
[^\s]*$")) {
    throw new IllegalArgumentException("Invalid URL format for
file");
}

```

					ПГУ 1.090301.04.001 ПЗ	Лист
						69
Изм.	Лист	№ докум.	Подпись	Дата		

```

        lecture.setTitle(title);
        lecture.setContent(content);
        lecture.setFileUrl(fileUrl);

        System.out.println("Saving updated lecture: " + title);
        Lecture updatedLecture = lectureRepository.save(lecture);
        System.out.println("Lecture updated with ID: " +
updatedLecture.getId());
        return ResponseEntity.ok(updatedLecture);
    } catch (IllegalArgumentException e) {
        System.err.println("Validation error: " + e.getMessage());
        throw new RuntimeException("Validation failed: " +
e.getMessage());
    } catch (Exception e) {
        System.err.println("Lecture update error: " +
e.getMessage());
        e.printStackTrace();
        throw new RuntimeException("Failed to update lecture: " +
e.getMessage());
    }
}

@DeleteMapping("/{courseId}/lectures/{lectureId}")
@PreAuthorize("hasRole('TEACHER')")
public ResponseEntity<Void> deleteLecture(@PathVariable Long
courseId, @PathVariable Long lectureId) {
    try {
        System.out.println("Deleting lecture ID: " + lectureId + "
for course ID: " + courseId);
        Course course = courseRepository.findById(courseId)
            .orElseThrow(() -> new RuntimeException("Course not
found"));
        String currentUserEmail =
SecurityContextHolder.getContext().getAuthentication().getName();

```

```

if (!course.getTeacher().getEmail().equals(currentUserEmail)) {
    throw new RuntimeException("You are not authorized to
delete this lecture");
}

Lecture lecture = lectureRepository.findById(lectureId)
    .orElseThrow(() -> new RuntimeException("Lecture not
found"));

if (!lecture.getCourse().getId().equals(courseId)) {
    throw new RuntimeException("Lecture does not belong to
this course");
}

System.out.println("Deleting lecture");
lectureRepository.delete(lecture);
System.out.println("Lecture deleted: " + lectureId);
return ResponseEntity.noContent().build();
} catch (Exception e) {
    System.err.println("Lecture deletion error: " +
e.getMessage());
    e.printStackTrace();
    throw new RuntimeException("Failed to delete lecture: " +
e.getMessage());
}
}

@PostMapping("/{id}/enroll")
@PreAuthorize("hasRole('STUDENT')")
public ResponseEntity<Void> enrollCourse(@PathVariable Long id) {
    try {
        System.out.println("Enrolling user to course ID: " + id);
        Course course = courseRepository.findById(id)

```

```

        .orElseThrow(() -> new RuntimeException("Course not found"));

        String currentUserEmail =
SecurityContextHolder.getContext().getAuthentication().getName();
        User user = userRepository.findByEmail(currentUserEmail)
            .orElseThrow(() -> new RuntimeException("User
not found"));

        if
(enrollmentRepository.existsByUserIdAndCourseId(user.getId(),
course.getId())) {
            throw new RuntimeException("User is already enrolled
in this course");
        }

        Enrollment enrollment = new Enrollment();
        enrollment.setUser(user);
        enrollment.setCourse(course);
        enrollment.setEnrolledAt(LocalDateTime.now());

        System.out.println("Saving enrollment to database");
        enrollmentRepository.save(enrollment);
        System.out.println("User enrolled to course ID: " + id);
        return ResponseEntity.ok().build();
    } catch (Exception e) {
        System.err.println("Enrollment error: " +
e.getMessage());
        e.printStackTrace();
        throw new RuntimeException("Failed to enroll in course:
" + e.getMessage());
    }
}
}
}

```

CourseReviewController

```
package com.example.courseplatform.controller;
import com.example.courseplatform.model.CourseReview;
import com.example.courseplatform.model.User;
import com.example.courseplatform.service.CourseReviewService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import
org.springframework.security.core.annotation.AuthenticationPrincipal
;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/reviews")
public class CourseReviewController {
    @Autowired
    private CourseReviewService reviewService;

    @PostMapping("/course/{courseId}")
    public ResponseEntity<CourseReview> addReview(@PathVariable Long
courseId,
                                                    @RequestBody
ReviewRequest request,

    @AuthenticationPrincipal User user) {
        CourseReview review = reviewService.addReview(courseId,
request.getRating(),
            request.getComment(), user);
        return ResponseEntity.ok(review);
    }
}
```

					ПГУ 1.090301.04.001 ПЗ	Лист
						73
Изм.	Лист	№ докум.	Подпись	Дата		

```

    @GetMapping("/course/{courseId}")
    public ResponseEntity<List<CourseReview>>
getCourseReviews(@PathVariable Long courseId) {
        return
ResponseEntity.ok(reviewService.getReviewsByCourseId(courseId));
    }
    @GetMapping("/course/{courseId}/average-rating")
    public ResponseEntity<Double> getAverageRating(@PathVariable
Long courseId) {
        return
ResponseEntity.ok(reviewService.getAverageRating(courseId));
    }
}
class ReviewRequest {
    private int rating;
    private String comment;

    public int getRating() { return rating; }
    public void setRating(int rating) { this.rating = rating; }
    public String getComment() { return comment; }
    public void setComment(String comment) { this.comment = comment; }
}
}

```

LeaderboardController.java

```

package com.example.courseplatform.controller;
import com.example.courseplatform.model.User;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.RequestMapping;
import java.util.List;

```

```

@RestController
@RequestMapping("/api/users")
public class LeaderboardController {

    private static final Logger logger =
LoggerFactory.getLogger(LeaderboardController.class);

    @Autowired
    private UserRepository userRepository;

    @GetMapping("/leaderboard")
    public List<User> getLeaderboard() {
        logger.debug("Получение лидерборда");
        try {
            List<User> students =
userRepository.findByRoleOrderByPointsDesc("ROLE_STUDENT");
            logger.info("Найдено студентов для лидерборда: {}",
students.size());
            return students;
        } catch (Exception e) {
            logger.error("Ошибка при получении лидерборда: {}",
e.getMessage(), e);
            throw new
ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "Ошибка
при получении лидерборда: " + e.getMessage(), e);
        }
    }
}

```

TaskController.java

```

@RestController
@RequestMapping("/api/tasks")
public class TaskController {

```

					ПГУ 1.090301.04.001 ПЗ	Лист
						75
Изм.	Лист	№ докум.	Подпись	Дата		

```

private static final Logger logger =
LoggerFactory.getLogger(TaskController.class);

@Autowired
private TaskRepository taskRepository;

@Autowired
private TaskSubmissionRepository taskSubmissionRepository;

@Autowired
private UserRepository userRepository;

@Autowired
private EnrollmentRepository enrollmentRepository;

@Autowired
private CourseRepository courseRepository;

@GetMapping
public List<Task> getAllTasks() {
    String currentUserEmail =
SecurityContextHolder.getContext().getAuthentication().getName();
    logger.debug("Получение заданий для пользователя: {}",
currentUserEmail);
    User user = userRepository.findByEmail(currentUserEmail)
        .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND, "Пользователь не
найден: " + currentUserEmail));
    if ("ROLE_STUDENT".equals(user.getRole())) {
        List<Long> enrolledCourseIds =
enrollmentRepository.findByUserId(user.getId())
            .stream()
            .map(enrollment -> {

```

```

Course course = enrollment.getCourse();
        return course != null ? course.getId() : null;
    })
    .filter(Objects::nonNull)
    .toList();
return taskRepository.findAll().stream()
    .filter(task -> {
        Course course = task.getCourse();
        return course != null &&
enrolledCourseIds.contains(course.getId());
    })
    .toList();
} else if ("ROLE_TEACHER".equals(user.getRole())) {
    List<Long> teacherCourseIds =
courseRepository.findByTeacherId(user.getId())
        .stream()
        .map(Course::getId)
        .toList();
    return taskRepository.findAll().stream()
        .filter(task -> {
            Course course = task.getCourse();
            return course != null &&
teacherCourseIds.contains(course.getId());
        })
        .toList();
}
return List.of();
}

@PostMapping
@PreAuthorize("hasRole('TEACHER')")
public ResponseEntity<Task> createTask(@RequestBody Task task)
{

```

```

String currentUserEmail =
SecurityContextHolder.getContext().getAuthentication().getName();
    logger.debug("Создание задания пользователем: {}",
currentUserEmail);
    User user = userRepository.findByEmail(currentUserEmail)
        .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND, "Пользователь не
найден: " + currentUserEmail));
    if (task == null || task.getCourse() == null ||
task.getCourse().getId() == null) {
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST,
"Курс не указан");
    }
    Course course =
courseRepository.findById(task.getCourse().getId())
        .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND, "Курс не найден: " +
task.getCourse().getId()));
    if (!course.getTeacher().getId().equals(user.getId())) {
        throw new ResponseStatusException(HttpStatus.FORBIDDEN, "Вы
не являетесь преподавателем курса: " + course.getId());
    }
    task.setCourse(course);
    Task savedTask = taskRepository.save(task);
    logger.info("Задание создано: {}", savedTask.getId());
    return ResponseEntity.ok(savedTask);
}
@PostMapping("/{id}/submit")
@PreAuthorize("hasRole('STUDENT')")
public ResponseEntity<TaskSubmission> submitTask(
    @PathVariable Long id,
    @RequestParam(value = "text", required = false) String text,
    @RequestParam(value = "file_url", required = false) String
fileUrl) {

```

```

logger.debug("Отправка решения для задания {} пользователем {}", id,
SecurityContextHolder.getContext().getAuthentication().getName());
try {
    // Проверка задания
    Task task = taskRepository.findById(id)
        .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND, "Задание не найдено: "
+ id));
    logger.debug("Задание найдено: {}", task.getId());

    // Проверка пользователя
    String currentUserEmail =
SecurityContextHolder.getContext().getAuthentication().getName();
    User user = userRepository.findByEmail(currentUserEmail)
        .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND, "Пользователь не
найден: " + currentUserEmail));
    logger.debug("Пользователь найден: {}", user.getId());

    // Проверка курса
    if (task.getCourse() == null) {
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST,
"Задание не связано с курсом: " + id);
    }
    logger.debug("Курс задания: {}", task.getCourse().getId());

    // Проверка записи на курс
    if (!
enrollmentRepository.existsByUserIdAndCourseId(user.getId(),
task.getCourse().getId())) {
        throw new ResponseStatusException(HttpStatus.FORBIDDEN, "Вы
не записаны на курс: " + task.getCourse().getId());
    }
    logger.debug("Пользователь записан на курс");

```

```

if (taskSubmissionRepository.existsByUserIdAndTaskId(user.getId(),
id)) {
    logger.debug("Решение для задания {} пользователем {} уже
существует", id, user.getId());
    throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "Вы
уже отправили решение для этого задания");
}
logger.debug("Решение ещё не отправлено");

// Проверка данных
if ((text == null || text.trim().isEmpty()) && (fileUrl == null ||
fileUrl.trim().isEmpty())) {
    throw new ResponseStatusException(HttpStatus.BAD_REQUEST,
"Необходимо предоставить текст или ссылку");
}
logger.debug("Данные для отправки: text={}, file_url={}", text,
fileUrl);

// Создание записи
TaskSubmission submission = new TaskSubmission();
submission.setUser(user);
submission.setTask(task);
submission.setText(text != null && !text.trim().isEmpty() ? text :
null);
submission.setFileUrl(fileUrl != null && !fileUrl.trim().isEmpty() ?
fileUrl : null);
submission.setSubmittedAt(LocalDateTime.now());
logger.debug("Создана запись TaskSubmission");
// Сохранение решения (без начисления баллов, ждём оценки
преподавателя)
TaskSubmission savedSubmission =
taskSubmissionRepository.save(submission);
logger.info("Решение сохранено: {}", savedSubmission.getId());
return ResponseEntity.ok(savedSubmission);

```

```

    } catch (Exception e) {
        logger.error("Ошибка при отправке решения для задания {}:
{}", id, e.getMessage(), e);
        throw new
ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "Ошибка
при отправке решения: " + e.getMessage(), e);
    }
}

@GetMapping("/{id}/submissions")
@PreAuthorize("hasRole('TEACHER')")
public List<TaskSubmission> getTaskSubmissions(@PathVariable Long
id) {
    logger.debug("Получение решений для задания {}", id);
    try {
        Task task = taskRepository.findById(id)
            .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND, "Задание не найдено: "
+ id));
        logger.debug("Задание найдено: {}", task.getId());
        String currentUserEmail =
SecurityContextHolder.getContext().getAuthentication().getName();
        User user = userRepository.findByEmail(currentUserEmail)
            .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND, "Пользователь не
найден: " + currentUserEmail));
        logger.debug("Пользователь найден: {}", user.getId());
        Course course = task.getCourse();
        if (course == null) {
            logger.error("Задание {} не привязано к курсу", id);
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST, "Задание не
привязано к курсу");
        }
    }
}

```

```

User teacher = course.getTeacher();
    if (teacher == null) {
        logger.error("Курс {} не имеет преподавателя",
course.getId());
        throw new
ResponseStatusException(HttpStatus.BAD_REQUEST, "Курс не имеет
преподавателя");
    }
    if (!teacher.getId().equals(user.getId())) {
        logger.debug("Преподаватель {} не имеет доступа к
решениям задания {}. Курс: {}, Преподаватель курса: {}",
currentUserEmail, id, course.getId(),
teacher.getId());
        throw new ResponseStatusException(HttpStatus.FORBIDDEN,
"Вы не можете просматривать решения этого задания");
    }
    logger.debug("Преподаватель авторизован для курса: {}",
course.getId());

    logger.debug("Поиск решений для задания {}", id);
    List<TaskSubmission> submissions =
taskSubmissionRepository.findById(id);
    logger.info("Найдено решений для задания {}: {}", id,
submissions.size());
    return submissions;
} catch (Exception e) {
    logger.error("Ошибка при получении решений для задания {}:
{}", id, e.getMessage(), e);
    throw new
ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "Ошибка
при получении решений: " + (e.getMessage() != null ?
e.getMessage() : "Неизвестная ошибка"), e);
}
}

```

```

@GetMapping("/submissions/my")
@PreAuthorize("hasRole('STUDENT')")
public List<TaskSubmission> getMySubmissions() {
    logger.debug("Получение отправленных решений для пользователя {}\"",
SecurityContextHolder.getContext().getAuthentication().getName());
    try {
        String currentUserEmail =
SecurityContextHolder.getContext().getAuthentication().getName();
        User user = userRepository.findByEmail(currentUserEmail)
            .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND, "Пользователь не
найден: " + currentUserEmail));
        logger.debug("Пользователь найден: {}", user.getId());

        List<TaskSubmission> submissions =
taskSubmissionRepository.findById(user.getId());
        logger.info("Найдено решений для пользователя {}: {}",
user.getId(), submissions.size());
        return submissions;
    } catch (Exception e) {
        logger.error("Ошибка при получении отправленных решений:
{}", e.getMessage(), e);
        throw new
ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "Ошибка
при получении отправленных решений: " + e.getMessage(), e);
    }
}

@PostMapping("/submissions/{submissionId}/grade")
@PreAuthorize("hasRole('TEACHER')")
public ResponseEntity<TaskSubmission> gradeSubmission(
    @PathVariable Long submissionId,
    @RequestBody Map<String, Integer> request) {

```

```

logger.debug("Выставление баллов для решения {} пользователем {}",
submissionId,
SecurityContextHolder.getContext().getAuthentication().getName());
try {
    // Проверка решения
    TaskSubmission submission =
taskSubmissionRepository.findById(submissionId)
        .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND, "Решение не найдено: "
+ submissionId));
    logger.debug("Решение найдено: {}", submission.getId());

    // Проверка преподавателя
    String currentUserEmail =
SecurityContextHolder.getContext().getAuthentication().getName();
    User teacher = userRepository.findByEmail(currentUserEmail)
        .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND, "Преподаватель не
найден: " + currentUserEmail));
    logger.debug("Преподаватель найден: {}", teacher.getId());

    // Проверка прав преподавателя
    Task task = submission.getTask();
    if (task == null || task.getCourse() == null || !
task.getCourse().getTeacher().getId().equals(teacher.getId())) {
        throw new ResponseStatusException(HttpStatus.FORBIDDEN, "Вы
не можете оценивать это решение");
    }
    logger.debug("Преподаватель авторизован для курса: {}",
task.getCourse().getId());

    // Проверка баллов
    Integer points = request.get("points");

```

```

if (points == null || points < 0) {
    throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "Баллы
должны быть неотрицательными");
}
logger.debug("Указанные баллы: {}", points);
// Обновление баллов пользователя
User student = submission.getUser();
Integer previousPoints = submission.getPoints();
if (previousPoints != null) {
    // Вычитаем старые баллы
    student.setPoints(student.getPoints() - previousPoints);
    logger.debug("Вычтены старые баллы: {}", previousPoints);
}
// Добавляем новые баллы
student.setPoints(student.getPoints() + points);
userRepository.save(student);
logger.debug("Баллы студента обновлены: {}", student.getPoints());
// Обновление решения
submission.setPoints(points);
TaskSubmission updatedSubmission =
taskSubmissionRepository.save(submission);
logger.debug("Решение обновлено с баллами: {}", points);
logger.info("Баллы для решения {} установлены: {}", submissionId,
points);
return ResponseEntity.ok(updatedSubmission);
    } catch (Exception e) {
        logger.error("Ошибка при выставлении баллов для решения {}:
{}", submissionId, e.getMessage(), e);
        throw new
ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "Ошибка
при выставлении баллов: " + e.getMessage(), e);
    }
}

```

					ПГУ 1.090301.04.001 ПЗ	Лист
						85
Изм.	Лист	№ докум.	Подпись	Дата		

```

@DeleteMapping("/{id}")
@PreAuthorize("hasRole('TEACHER')")
@Transactional
public ResponseEntity<Void> deleteTask(@PathVariable Long id) {
    logger.debug("Удаление задания {} пользователем {}", id,
SecurityContextHolder.getContext().getAuthentication().getName());
    try {
        Task task = taskRepository.findById(id)
            .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND, "Задание не найдено: "
+ id));
        logger.debug("Задание найдено: {}", task.getId());

        String currentUserEmail =
SecurityContextHolder.getContext().getAuthentication().getName();
        User user = userRepository.findById(currentUserEmail)
            .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND, "Пользователь не
найден: " + currentUserEmail));
        logger.debug("Пользователь найден: {}", user.getId());

        Course course = task.getCourse();
        if (course == null || !
course.getTeacher().getId().equals(user.getId())) {
            logger.debug("Преподаватель {} не имеет прав на удаление
задания {}", currentUserEmail, id);
            throw new ResponseStatusException(HttpStatus.FORBIDDEN,
"Вы не можете удалять это задание");
        }

        logger.debug("Преподаватель авторизован для курса: {}",
course.getId());
    }
}

```

```

        // Удаляем связанные решения
        taskSubmissionRepository.deleteByTaskId(id);
        logger.debug("Связанные решения для задания {} удалены",
id);

        // Удаляем задание
        taskRepository.delete(task);
        logger.info("Задание {} удалено", id);
        return ResponseEntity.noContent().build();
    } catch (Exception e) {
        logger.error("Ошибка при удалении задания {}: {}", id,
e.getMessage(), e);
        throw new
ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "Ошибка
при удалении задания: " + e.getMessage(), e);
    }
}

```

UserController.java

@RestController

@RequestMapping("/api/users")

public class UserController {

@Autowired

private UserRepository userRepository

@GetMapping("/me")

public ResponseEntity<Map<String, Object>> getCurrentUser() {

String email =

SecurityContextHolder.getContext().getAuthentication().getName();

User user = userRepository.findByEmail(email)

.orElseThrow(() -> new RuntimeException("User not
found"));

Map<String, Object> response = new HashMap<>();

response.put("email", user.getEmail());

response.put("role", user.getRole());

return ResponseEntity.ok(response);

}

					ПГУ 1.090301.04.001 ПЗ	Лист
						87
Изм.	Лист	№ докум.	Подпись	Дата		

Course.java

```
package com.example.courseplatform.model;

import jakarta.persistence.*;
import com.fasterxml.jackson.annotation.JsonProperty;

@Entity
@Table(name = "courses")
public class Course {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    @JsonProperty("title")
    private String title;

    @Column(columnDefinition = "TEXT")
    @JsonProperty("description")
    private String description;

    @Column
    @JsonProperty("category")
    private String category; // Добавлено

    @ManyToOne
    @JoinColumn(name = "teacher_id", nullable = false)
    @JsonProperty("teacher")
    private User teacher;

    public Course() {}

    public Course(String title, String description, String category,
        User teacher) {
        this.title = title;
    }
}
```

					ПГУ 1.090301.04.001 ПЗ	Лист
						88
Изм.	Лист	№ докум.	Подпись	Дата		

```

        this.description = description;
        this.category = category;
        this.teacher = teacher;
    }

    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }
    public String getDescription() { return description; }
    public void setDescription(String description)
{ this.description = description; }
    public String getCategory() { return category; }
    public void setCategory(String category) { this.category =
category; }
    public User getTeacher() { return teacher; }
    public void setTeacher(User teacher) { this.teacher = teacher; }
}

```

CourseReview.java

```

package com.example.courseplatform.model;

import jakarta.persistence.*;
import java.time.LocalDateTime;

@Entity
@Table(name = "course_reviews")
public class CourseReview {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "user_id")
    private User user;
}

```

@ManyToOne

@JoinColumn(name = "course_id")

private Course course;

private int rating; // 1-5

private String comment;

private LocalDateTime createdAt;

// Геттеры и сеттеры

public Long getId() { return id; }

public void setId(Long id) { this.id = id; }

public User getUser() { return user; }

public void setUser(User user) { this.user = user; }

public Course getCourse() { return course; }

public void setCourse(Course course) { this.course = course; }

public int getRating() { return rating; }

public void setRating(int rating) { this.rating = rating; }

public String getComment() { return comment; }

public void setComment(String comment) { this.comment = comment; }

}

public LocalDateTime getCreatedAt() { return createdAt; }

public void setCreatedAt(LocalDateTime createdAt)

{ this.createdAt = createdAt; }

}

Enrollment.java

package com.example.courseplatform.model;

import jakarta.persistence.*;

import java.time.LocalDateTime;

@Entity

@Table(name = "enrollments")

public class Enrollment {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

					ПГУ 1.090301.04.001 ПЗ	Лист
						90
Изм.	Лист	№ докум.	Подпись	Дата		

```

private Long id;
    @ManyToOne
    @JoinColumn(name = "user_id", nullable = false)
    private User user;
    @ManyToOne
    @JoinColumn(name = "course_id", nullable = false)
    private Course course;
    public Enrollment() {}
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public User getUser() {
        return user;
    }
    public void setUser(User user) {
        this.user = user;
    }
    public Course getCourse() {
        return course;
    }
    public void setCourse(Course course) {
        this.course = course;
    }
    public void setEnrolledAt(LocalDateTime now) {
    }
}

```

Lecture.java

```

package com.example.courseplatform.model;
import jakarta.persistence.*;

```

```

import java.time.LocalDateTime;
@Entity
@Table(name = "enrollments")
public class Enrollment {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @ManyToOne
    @JoinColumn(name = "user_id", nullable = false)
    private User user;
    @ManyToOne
    @JoinColumn(name = "course_id", nullable = false)
    private Course course;
    public Enrollment() {}
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public User getUser() {
        return user;
    }
    public void setUser(User user) {
        this.user = user;
    }
    public Course getCourse() {
        return course;}
    public void setCourse(Course course) {
        this.course = course;}
    public void setEnrolledAt(LocalDateTime now) {}
}

```

Task.java

```
package com.example.courseplatform.model;
import jakarta.persistence.*;
import java.time.LocalDateTime;
@Entity
@Table(name = "enrollments")
public class Enrollment {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @ManyToOne
    @JoinColumn(name = "user_id", nullable = false)
    private User user;
    @ManyToOne
    @JoinColumn(name = "course_id", nullable = false)
    private Course course;
    public Enrollment() {}
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public User getUser() {
        return user;
    }
    public void setUser(User user) {
        this.user = user;
    }
    public Course getCourse() {
        return course;
    }
    public void setCourse(Course course) {
        this.course = course;
    }
    public void setEnrolledAt(LocalDateTime now) {}
}
```

					ПГУ 1.090301.04.001 ПЗ	Лист
						93
Изм.	Лист	№ докум.	Подпись	Дата		

TaskSubmission.java

```
package com.example.courseplatform.model;
import jakarta.persistence.*;
import java.time.LocalDateTime;
@Entity
@Table(name = "enrollments")
public class Enrollment {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @ManyToOne
    @JoinColumn(name = "user_id", nullable = false)
    private User user;
    @ManyToOne
    @JoinColumn(name = "course_id", nullable = false)
    private Course course;
    public Enrollment() {}
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public User getUser() {
        return user;
    }
    public void setUser(User user) {
        this.user = user;
    }
    public Course getCourse() {
        return course;
    }
    public void setCourse(Course course) {
        this.course = course;
    }
    public void setEnrolledAt(LocalDateTime now) {
    }
}
```

User.java

```
package com.example.courseplatform.model;
import jakarta.persistence.*;
@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String email;
    private String password;
    private String role;
    private int points;
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
    public String getPassword() { return password; }
    public void setPassword(String password) { this.password =
password; }
    public String getRole() { return role; }
    public void setRole(String role) { this.role = role; }
    public int getPoints() { return points; }
    public void setPoints(int points) { this.points = points; }
}
```

CourseRepository.java

```
import com.example.courseplatform.model.Course;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.List;
public interface CourseRepository extends JpaRepository<Course,
Long> {
    List<Course> findByTeacherId(Long teacherId);}
```

CourseReviewRepository.java

```
package com.example.courseplatform.repository;
import com.example.courseplatform.model.CourseReview;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

import java.util.List;

public interface CourseReviewRepository extends
JpaRepository<CourseReview, Long> {
    List<CourseReview> findByCourseId(Long courseId);
    @Query("SELECT AVG(r.rating) FROM CourseReview r WHERE
r.course.id = :courseId")
    Double getAverageRatingByCourseId(@Param("courseId") Long
courseId);
}
```

EnrollmentRepository.java

```
package com.example.courseplatform.repository;
import com.example.courseplatform.model.CourseReview;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import java.util.List;

public interface CourseReviewRepository extends
JpaRepository<CourseReview, Long> {
    List<CourseReview> findByCourseId(Long courseId);
    @Query("SELECT AVG(r.rating) FROM CourseReview r WHERE
r.course.id = :courseId")
    Double getAverageRatingByCourseId(@Param("courseId") Long
courseId);
}
```

LectureRepository.java

```
package com.example.courseplatform.repository;

import com.example.courseplatform.model.Lecture;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.List;

public interface LectureRepository extends JpaRepository<Lecture, Long> {

    List<Lecture> findByCourseId(Long courseId);

}
```

TaskRepository

```
package com.example.courseplatform.repository;

import com.example.courseplatform.model.Task;
import org.springframework.data.jpa.repository.JpaRepository;

public interface TaskRepository extends JpaRepository<Task, Long> {

}
```

TaskSubmissionRepository.java

```
package com.example.courseplatform.repository;

import com.example.courseplatform.model.TaskSubmission;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import java.util.List;

public interface TaskSubmissionRepository extends JpaRepository<TaskSubmission, Long> {

    // Метод для получения решений по ID задания
    @Query("SELECT ts FROM TaskSubmission ts WHERE ts.task.id = :taskId")
    List<TaskSubmission> findByTaskId(@Param("taskId") Long taskId);

}
```

```

        // Метод для удаления решений по ID задания
        @Modifying
        @Query("DELETE FROM TaskSubmission ts WHERE ts.task.id
= :taskId")
        void deleteByTaskId(@Param("taskId") Long taskId);

        List<TaskSubmission> findByUserIdAndTaskId(Long userId, Long
taskId);
        boolean existsByUserIdAndTaskId(Long userId, Long taskId);
        List<TaskSubmission> findByUserId(Long id);
    }

```

CourseReviewService.java

```

import java.time.LocalDateTime;
import java.util.List;
@Service
public class CourseReviewService {
    @Autowired
    private CourseReviewRepository reviewRepository;
    @Autowired
    private CourseRepository courseRepository;
    public CourseReview addReview(Long courseId, int rating, String
comment, User user) {
        Course course = courseRepository.findById(courseId)
            .orElseThrow(() -> new RuntimeException("Курс не
найден"));
        CourseReview review = new CourseReview();
        review.setUser(user);
        review.setCourse(course);
        review.setRating(rating);
        review.setComment(comment);
        review.setCreatedAt(LocalDateTime.now());
        return reviewRepository.save(review);
    }
}

```

```

    public List<CourseReview> getReviewsByCourseId(Long courseId) {
        return reviewRepository.findByCourseId(courseId);
    }
    public Double getAverageRating(Long courseId) {
        Double avgRating =
reviewRepository.getAverageRatingByCourseId(courseId);
        return avgRating != null ? avgRating : 0.0;
    }
}

```

CourseplatformApplication.java

```

package com.example.courseplatform;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication

public class CourseplatformApplication {
    public static void main(String[] args) {
        SpringApplication.run(CourseplatformApplication.class, args);
    }
}

```

Login.js

```

import { useState } from "react";
import { useNavigate, Link } from "react-router-dom";
import "../index.css";

function Login() {
    const [email, setEmail] = useState("");
    const [password, setPassword] = useState("");
    const [error, setError] = useState("");
    const navigate = useNavigate();}

```

```

const handleLogin = async (e) => {
  e.preventDefault();
  try {
    const credentials = btoa(`${email}:${password}`);
    const response = await
fetch("http://localhost:8081/api/users/me", {
  headers: {
    Authorization: `Basic ${credentials}`,
  },
});
if (response.status === 401) {
  throw new Error("Неверный email или пароль");
}
if (!response.ok) {
  throw new Error("Ошибка входа");
}
localStorage.setItem("credentials", credentials);
setError("");
navigate("/tasks");
} catch (err) {
  setError(err.message);
}
};

return (
  <div className="container">
    <h1 className="heading">Вход</h1>
    <form onSubmit={handleLogin} className="form">
      <div className="form-group">
        <label>Email:</label>
        <input
          type="email"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
          placeholder="Введите email"
          required/

```

```

    </div>
    <div className="form-group">
      <label>Пароль:</label>
      <input
        type="password"
        value={password}
        onChange={(e) => setPassword(e.target.value)}
        placeholder="Введите пароль"
        required
      />
    </div>
    {error && <p className="error">{error}</p>}
    <button type="submit" className="button">
      Войти
    </button>
    <p className="form-text">
      Нет аккаунта?{" "}
      <Link to="/register" className="link">
        Зарегистрироваться
      </Link>
    </p>
  </form>
</div>
);
}
export default Login;

```

Register.js

```

import { useState } from "react";
import { useNavigate, Link } from "react-router-dom";
import "../index.css";
function Register() {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [role, setRole] = useState("ROLE_STUDENT");

```

```

const [error, setError] = useState(null);
const navigate = useNavigate();

const handleRegister = async (e) => {
  e.preventDefault();
  if (!email || !password || password.length < 6) {
    setError("Введите корректный email и пароль (не менее 6
СИМВОЛОВ)");
    return;
  }

  try {
    const response = await
fetch("http://localhost:8081/api/auth/register", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify({ email, password, role }),
});

    if (!response.ok) {
      const text = await response.text();
      let errorMessage = "Ошибка регистрации";
      try {
        const errorJson = JSON.parse(text);
        errorMessage = errorJson.message || errorMessage;
      } catch (e) {
        errorMessage = text || errorMessage;
      }
      throw new Error(errorMessage);
    }
  }
}

```

```

setError(null);
    alert("Регистрация успешна! Войдите в аккаунт.");
    navigate("/login");
} catch (err) {
    setError(err.message);
}
};
return (
    <div className="container">
        <h1 className="heading">Регистрация</h1>
        <form onSubmit={handleRegister} className="form">
            <div className="form-group">
                <label>Email:</label>
                <input
                    type="email"
                    value={email}
                    onChange={(e) => setEmail(e.target.value)}
                    placeholder="Введите email"
                    required
                />
            </div>
            <div className="form-group">
                <label>Пароль:</label>
                <input
                    type="password"
                    value={password}
                    onChange={(e) => setPassword(e.target.value)}
                    placeholder="Введите пароль (мин. 6 символов)"
                    required
                />
            </div>
            <div className="form-group">
                <label>Роль:</label>
                <select value={role} onChange={(e) =>

```

```

<option value="ROLE_STUDENT">Студент</option>
      <option value="ROLE_TEACHER">Преподаватель</option>
    </select>
  </div>
  {error && <p className="error">{error}</p>}
  <button type="submit" className="button">
    Зарегистрироваться
  </button>
  <p className="form-text">
    Уже есть аккаунт?{" "}
    <Link to="/login" className="link">
      Войти
    </Link>
  </p>
</form>
</div>
);
}
export default Register;
App.js
import {
  BrowserRouter as Router,
  Routes,
  Route,
  Link,
  Navigate,
  useNavigate,
} from "react-router-dom";
import Courses from "../components/Courses";
import CourseDetails from "../components/CourseDetails";
import Tasks from "../components/Tasks";
import Leaderboard from "../components/Leaderboard";
import Login from "../components/Login";
import Register from "../components/Register";

```

```

function ProtectedRoute({ children }) {
  const credentials = localStorage.getItem("credentials");
  return credentials ? children : <Navigate to="/login" />;
}

function App() {
  const navigate = useNavigate();
  const isAuthenticated = !!localStorage.getItem("credentials");

  const handleLogout = () => {
    localStorage.removeItem("credentials");
    navigate("/login");
  };

  return (
    <div className="app">
      <header className="navbar">
        <div className="navbar-container">
          <Link to="/" className="navbar-logo">
            Course Platform
          </Link>
          <nav className="navbar-links">
            <Link to="/" className="navbar-link">
              Курсы
            </Link>
            <Link to="/tasks" className="navbar-link">
              Задания
            </Link>
            <Link to="/leaderboard" className="navbar-link">
              Лидеры
            </Link>
            {isAuthenticated ? (

```

```

<button onClick={handleLogout} className="navbar-button">
    Выйти
</button>
) : (
    <>
        <Link to="/login" className="navbar-link">
            Войти
        </Link>
        <Link to="/register" className="navbar-link">
            Регистрация
        </Link>
    </>
    </>
    </nav>
</div>
</header>
<main className="main-content">
    <Routes>
        <Route path="/login" element={<Login />} />
        <Route path="/register" element={<Register />} />
        <Route
            path="/"
            element={
                <ProtectedRoute>
                    <Courses />
                </ProtectedRoute>
            }
        />
    </Route>

    <Route
        path="/tasks"
        element={
            <ProtectedRoute>

```

```

    <Tasks />
        </ProtectedRoute>
    <Route
        path="/leaderboard"
        element={
            <ProtectedRoute>
                <Leaderboard />
            </ProtectedRoute>
        }
    </Routes>
</main>
<footer className="footer">
    <div className="footer-content">
        <p>© 2025 Course Platform. Все права защищены.</p>
        <div className="footer-links">
            <a href="/about" className="footer-link">
                0 проекте
            </a>
            <a href="/contact" className="footer-link">
                Контакты
            </a>
        </div>
    </div>
</footer>
</div>
);
}
export default function AppWrapper() {
    return (
        <Router>
            <App />
        </Router>
    );
}

```

Приложение В

(обязательное)

Сервис для управления учебными курсами

Презентация

					ПГУ 1.090301.04.001 ПЗ	Лист
						108
Изм.	Лист	№ докум.	Подпись	Дата		

Сервис для управления учебными курсами

Автор работы: ст. гр. 21ВВП1 Асянов Илгиз
Касимович
Научный руководитель: доцент кафедры ВТ
ПГУ Кузнецов Алексей Валерьевич

Пенза, 2025

Цель и задачи работы



Цель работы: разработка веб-сервиса для управления курсами, который позволит автоматизировать учебный процесс, предоставить пользователям удобный и интуитивно понятный интерфейс с необходимыми функциями



Создать сервисное приложение (Spring, MySQL)



Разработать клиентский интерфейс (Vue.js)



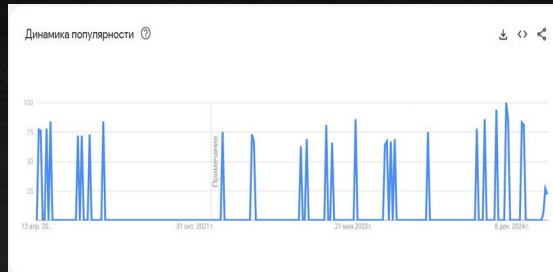
Реализовать REST API аутентификацию



Обеспечить ролевой доступ

СТРАНИЦА

- Рост популярности онлайн образования
- Потребность в автоматизации управления курсами.
- Необходимость ролевого доступа для преподавателей и студентов.
- Цифровизация образовательный процессов в вузах



Актуальность темы

СТРАНИЦА

Стек технологий

- Сервер:** Spring Boot, Spring Security, Spring Data JPA.



- Клиент:** React, JavaScript, HTML, CSS.



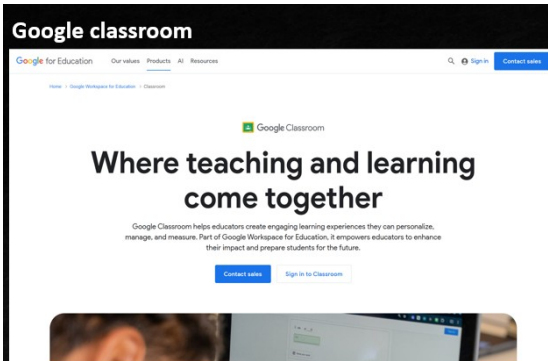
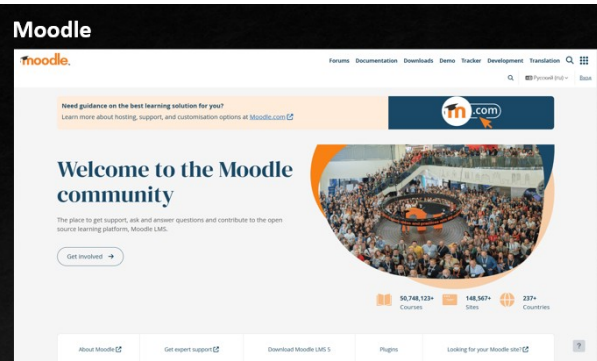
- База данных:** MySQL.



- Протокол:** REST API (HTTP).

СТРАНИЦА 5

					ПГУ 1.090301.04.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		110



Плюсы: простой и понятный интерфейс, интеграция с Google Drive и Meet, бесплатный доступ.

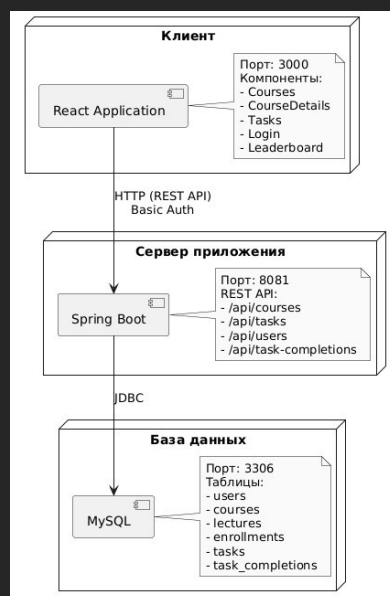
Минусы: ограниченный функционал для сложных курсов, отсутствие поддержки SCORM.

Аналоги

Плюсы: бесплатный open-source, гибкая настройка, поддержка множества форматов контента (PDF, видео, тесты).

Минусы: сложный интерфейс для новичков, требует настройки сервера.

СТРАНИЦА 4



Архитектура сервиса

СТРАНИЦА

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.04.001 ПЗ

Лист

111

Среды разработки

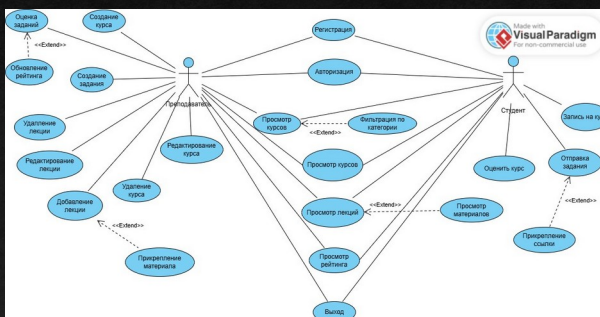
- IDE: IntelliJ IDEA (Spring Boot), VS Code (React).



СТРАНИЦА 6

Основной функционал

- Преподаватель (ROLE_TEACHER)**
 - Создание/редактирование курсов, лекций, задач.
- Студент (ROLE_STUDENT)**
 - Запись на курсы, выполнение задач, просмотр лекций.
- Общее**
 - Аутентификация (Authentication)
 - Хранение данных в MySQL.



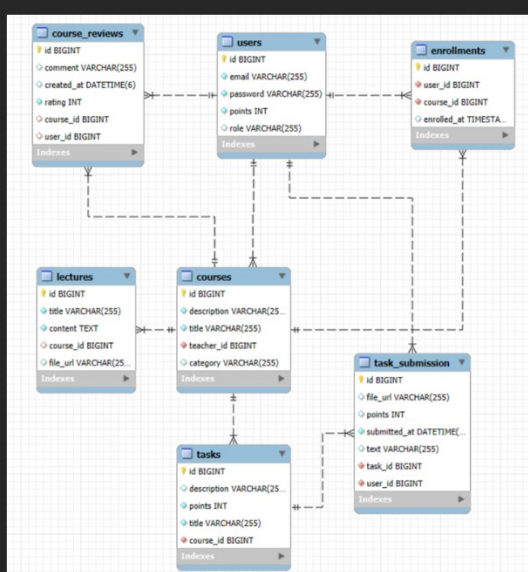
СТРАНИЦА

					ПГУ 1.090301.04.001 ПЗ	Лист
						112
Изм.	Лист	№ докум.	Подпись	Дата		

Результат разработки

- ▣ Разработана веб-система для управления курсами.
- ▣ Реализованы:
 - ▣ Сервер (Spring Boot, REST API).
 - ▣ Клиент (React, графический интерфейс).
 - ▣ База данных.
- ▣ Протестирована функциональность ролевого доступа.

СТРАНИЦА



Структура базы данных

СТРАНИЦА 10

					ПГУ 1.090301.04.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		113

Заключение

- Достигнута цель: создание веб-системы для курсов.
- Использованы современные технологии (Spring Boot, React, MySQL).
- Система готова к использованию и дальнейшему развитию.
- Перспективы: добавление JWT, мобильного приложения.

СТРАНИЦА 1

Спасибо за внимание!

СТРАНИЦА 2

					ПГУ 1.090301.04.001 ПЗ	Лист
						114
Изм.	Лист	№ докум.	Подпись	Дата		