

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА
на тему
Web-платформа образовательных курсов
компаний-работодателей

Студент _____ Логвин Артём Вячеславович
(подпись, дата) (ФИО полностью)

Руководитель _____ Михалев А.Г.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой _____ Козлов А.Ю.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК _____ Попова Н.А.
(подпись) (фамилия, инициалы)

Пенза, 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов
«20» декабря 2024 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Логвин Артём Вячеславович гр. 21ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>
2. Тема работы Web-платформа образовательных курсов компаний-работодателей

- Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____
2. Руководитель работы Михалев А.Г. _____
3. Задание на работу (назначение разработки, исходные данные и т.п.)_
- 1) Разработать Web-приложение, обеспечивающее
деятельность образовательного портала.
- 2) Среда выполнения приложения: операционная система Windows 10 и выше
Среда выполнения сервера: операционная система Linux
Языковые средства разработки: JavaScript, React, NodeJs, HTML
Инструментальные средства разработки: PostgreSQL,
Microsoft Visual Studio Code, Microsoft Project, Visual Paradigm
Технологии разработки: REST API, ООП
- 3) Основные функции Web-приложения:
- а) запись на курс и вакансию;
- б) просмотр списка курсов и вакансий;
- в) просмотр личного кабинета пользователя;
- г) оставление отзывов о курсах.

4. Перечень подлежащих разработке вопросов_

1) *Анализ предметной области и постановка задачи*

2) *Проектирование*

3) *Реализация*

4) *Контроль качества программного обеспечения*

5) *Оформление пояснительной записки*

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2025 – 14.04.2025	
2	Проектирование программных средств	20%	15.04.2025 – 25.04.2025	
3	Реализация программных средств	30%	26.04.2025 – 12.05.2025	
4	Тестирование программных средств	20%	13.05.2025 – 22.05.2025	
5	Оформление пояснительной записки	20%	23.05.2025 – 03.06.2025	

Дата выдачи задания «20» декабря 2024г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 159 листов, 42 рисунка, 21 таблица, 21 использованных источника, 2 приложения.

WEB-ПРИЛОЖЕНИЕ, ОБРАЗОВАТЕЛЬНЫЙ ПОРТАЛ, LINUX, POSTGRESQL, JAVASCRIPT, NODEJS, REACT, HTML, REST API, UML

Объектом разработки является Web-приложение для реализации деятельности образовательной платформы курсов компаний-работодателей.

Цель работы – разработка Web-приложения для реализации деятельности образовательной платформы курсов компаний-работодателей, позволяющего создавать курсы, вакансии и записываться на них, смотреть информацию о курсах и вакансиях, просматривать личный кабинет и управлять отзывами.

Разработка серверной части происходила на языке программирования NodeJS. В качестве среды разработки использовалась Visual Studio Code. В качестве СУБД была использована СУБД PostgreSQL.

Разработка клиентской части производилась на языке программирования JavaScript с использованием React. В качестве среды разработки использовалась среда программирования Visual Studio Code. Приложение предназначено для использования под управлением ОС Windows 10 и выше, а его серверная часть работает под управлением операционной системы Linux.

					ПГУ 09.03.04 – 08БР211.16 ПЗ			
Изм.	Лист	№ докум.	Подп.	Дата	Web-платформа образовательных курсов компаний-работодателей.	Лист.	Лист	Листов
Разраб.		Логвин А.В.					4	159
Пров.		Михалев А.Г.			Пояснительная записка	Группа 21ВП1		
Н. контр.		Такташкин Д.В.						
Утверд.		Козлов А.Ю.						

Содержание

Введение.....	8
1 Анализ требований и постановка задачи.....	10
1.1 Анализ предметной области	10
1.2 Анализ рынка программного обеспечения для ведения образовательных курсов.....	14
1.3 Анализ функциональных требований.....	16
1.4 Анализ нефункциональных требований.....	18
1.5 Выбор средств разработки	19
1.6 Постановка задачи	20
1.7 Планирование проекта и оценка бюджета	20
2 Проектирование образовательного портала.....	24
2.1 Проектирование архитектуры образовательного портала.....	24
2.2 Проектирование пользовательского интерфейса	25
2.3 Проектирование базы данных	27
2.3.1 Разработка концептуальной модели базы данных	28
2.3.2 Разработка логической модели базы данных.....	30
2.3.3 Разработка физической модели базы данных	32
2.3.4 Развертывание базы данных в среде pgAdmin 4.....	34
2.4 Проектирование серверной части образовательного портала	36
2.4.1 Проектирование сервиса пользователей	37
2.4.2 Проектирование сервиса курсов	38
2.4.3 Проектирование сервиса обучения	40

2.4.4	Проектирование сервиса администрирования.....	41
2.4.5	Проектирование сервиса сообщений	42
2.5	Проектирование алгоритмов.....	43
2.5.1	Алгоритм записи студента на курс	43
2.5.2	Алгоритм прохождения урока и обновления прогресса.....	45
2.5.6	Алгоритм рекомендаций курсов	46
3	Реализация образовательного портала	50
3.1	Реализация пользовательского функционала для роли студента ...	50
3.2	Реализация пользовательского функционала для роли преподавателя.....	52
3.4	Реализация пользовательского функционала для роли работодателя	54
3.5	Реализация пользовательского функционала для роли администратора	57
3.6	Реализация взаимодействия с базой данных.....	59
3.7	Реализация клиентской (пользовательской) части.....	61
3.8	Реализация модуля подбора вакансий и учебных курсов.....	64
3.9	Реализация средств оценки прохождения курса.....	66
4	Тестирование образовательного портала	69
4.1	Функциональное тестирование	69
4.2	Нефункциональное тестирование	72
4.3	Метрики кода.....	73
4.3.1	Размер кода (SLOC)	74
4.3.2	Метрика Холстеда.....	74

4.3.3 Интерпретация результатов	77
Заключение	78
Список использованных источников	80
Приложение А (обязательное) Листинг приложения	82
Приложение Б (рекомендуемое) Таблицы	147

Введение

Современный рынок труда предъявляет высокие требования к профессиональным навыкам специалистов, что делает актуальным развитие образовательных платформ, ориентированных на профессиональное обучение. Одним из перспективных направлений является создание web-платформ, которые объединяют интересы студентов, преподавателей, работодателей и администраторов, предоставляя возможности для обучения, управления курсами и оценки прогресса. Актуальность темы обусловлена необходимостью интеграции образовательных процессов с потребностями рынка труда, а также недостаточной представленностью платформ, ориентированных на курсы от компаний-работодателей [1].

Целью данной дипломной работы является разработка web-платформы образовательных курсов компаний-работодателей, обеспечивающей взаимодействие студентов, преподавателей, работодателей и администраторов.

Для достижения поставленной цели решаются следующие задачи:

- проведение анализа предметной области и функциональных требований;
- сравнительный анализ существующих решений на рынке образовательных платформ;
- проектирование архитектуры, пользовательского интерфейса и базы данных платформы;
- разработка алгоритмов взаимодействия пользователей с системой;
- планирование проекта и оценка бюджета.

Объектом исследования является процесс онлайн-обучения, а предметом исследования – web-платформа образовательных курсов компаний-работодателей.

В работе используются следующие методы исследования:

- анализ литературы и существующих решений;
- моделирование и проектирование с использованием UML и ER-диаграмм;
- экспериментальная разработка прототипа платформы.

Научная новизна работы заключается в разработке платформы, интегрирующей курсы от работодателей с системой управления ролями пользователей, что позволяет эффективно связывать образовательные процессы с потребностями рынка труда.

Практическая значимость работы состоит в возможности использования разработанной платформы в образовательных учреждениях, компаниях-работодателях и центрах профессиональной подготовки.

Структура работы включает введение, три главы, заключение, список литературы и приложения. В первой главе проводится анализ требований и постановка задачи. Во второй главе описывается проектирование платформы. В третьей главе представлено планирование проекта и оценка бюджета.

1 Анализ требований и постановка задачи

1.1 Анализ предметной области

Предметная область данной работы охватывает сферу онлайн-обучения, ориентированного на профессиональное развитие студентов с учетом потребностей компаний-работодателей. Онлайн-обучение представляет собой процесс передачи знаний и навыков через интернет с использованием специализированных платформ, обеспечивающих доступ к учебным материалам, тестирование и взаимодействие между участниками образовательного процесса [2]. Важной особенностью современных образовательных платформ является их интеграция с рынком труда, что позволяет студентам приобретать навыки, востребованные работодателями.

На основе анализа функциональных требований можно выделить следующие классы пользователей:

- гость – пользователь, не зарегистрированный в системе, имеющий ограниченный доступ к функционалу (просмотр каталога курсов, информации о курсах);
- студент – зарегистрированный пользователь, использующий платформу для обучения, прохождения курсов, получения сертификатов и взаимодействия с другими участниками;
- преподаватель/работодатель – пользователь, создающий и управляющий курсами, проверяющий прогресс студентов, анализирующий отзывы;
- администратор – пользователь с расширенными правами, управляющий каталогом курсов, аккаунтами пользователей, статистикой платформы и контентом.

Далее рассмотрим основные функции системы.

Основные функции системы, соответствующие классам пользователей, включают:

Для гостя:

- просмотр каталога курсов;
- использование параметров поиска курсов;
- просмотр информации о курсе;
- регистрация в системе.

Для студента:

- авторизация в системе;
- просмотр каталога курсов и использование поиска;
- запись на курс;
- просмотр пройденных курсов и прогресса;
- прохождение уроков (освоение материала, выполнение заданий,

прохождение тестов);

- получение сертификата по завершении курса;
- оставление комментариев и оценок курсов;
- общение с другими участниками портала.

Для преподавателя/работодателя:

- авторизация в системе;
- создание и редактирование курсов;
- просмотр списка записавшихся студентов и их прогресса;
- проверка заданий студентов;
- анализ отзывов о курсе;
- общение с другими участниками портала.

Особые функциональные требования можно выделить для роли администратора.

Для администратора:

- авторизация в системе;
- управление каталогом курсов (добавление, редактирование, удаление);
- просмотр статистики платформы;
- изменение категорий курсов;
- проверка контента портала;
- управление аккаунтами пользователей (блокировка, изменение ролей).

Контекстная диаграмма описывает взаимодействие системы с внешними сущностями (рисунок 1). Основные потоки данных включают:

- входные данные: запросы на регистрацию, авторизацию, поиск курсов, выполнение заданий, создание курсов и другие;
- выходные данные: информация о курсах, сертификаты, статистика, результаты проверки заданий и другие.

Система взаимодействует с различными пользователями, включая студентов, преподавателей и администраторов. Каждая из этих ролей имеет свой набор разрешений и функций в системе. Например, преподаватель может создавать и редактировать курсы, а студент – записываться на курсы и проходить задания. Также может быть предусмотрено взаимодействие с внешними сервисами, такими как платёжные шлюзы, сервисы проверки оригинальности работ или системы рассылки уведомлений.

Контекстная диаграмма также учитывает взаимодействие с различными ролями внутри системы. Студенты используют платформу для получения знаний, прохождения курсов и получения подтверждающих документов.

Преподаватели управляют учебными материалами, создают задания, тесты и проводят оценивание. Администраторы обеспечивают общий контроль над

системой, модерируют контент и поддерживают техническую стабильность её работы.

Такая архитектура позволяет обеспечить высокую масштабируемость, расширяемость и удобство работы с системой для всех категорий пользователей, удовлетворяющее потребности современных систем.

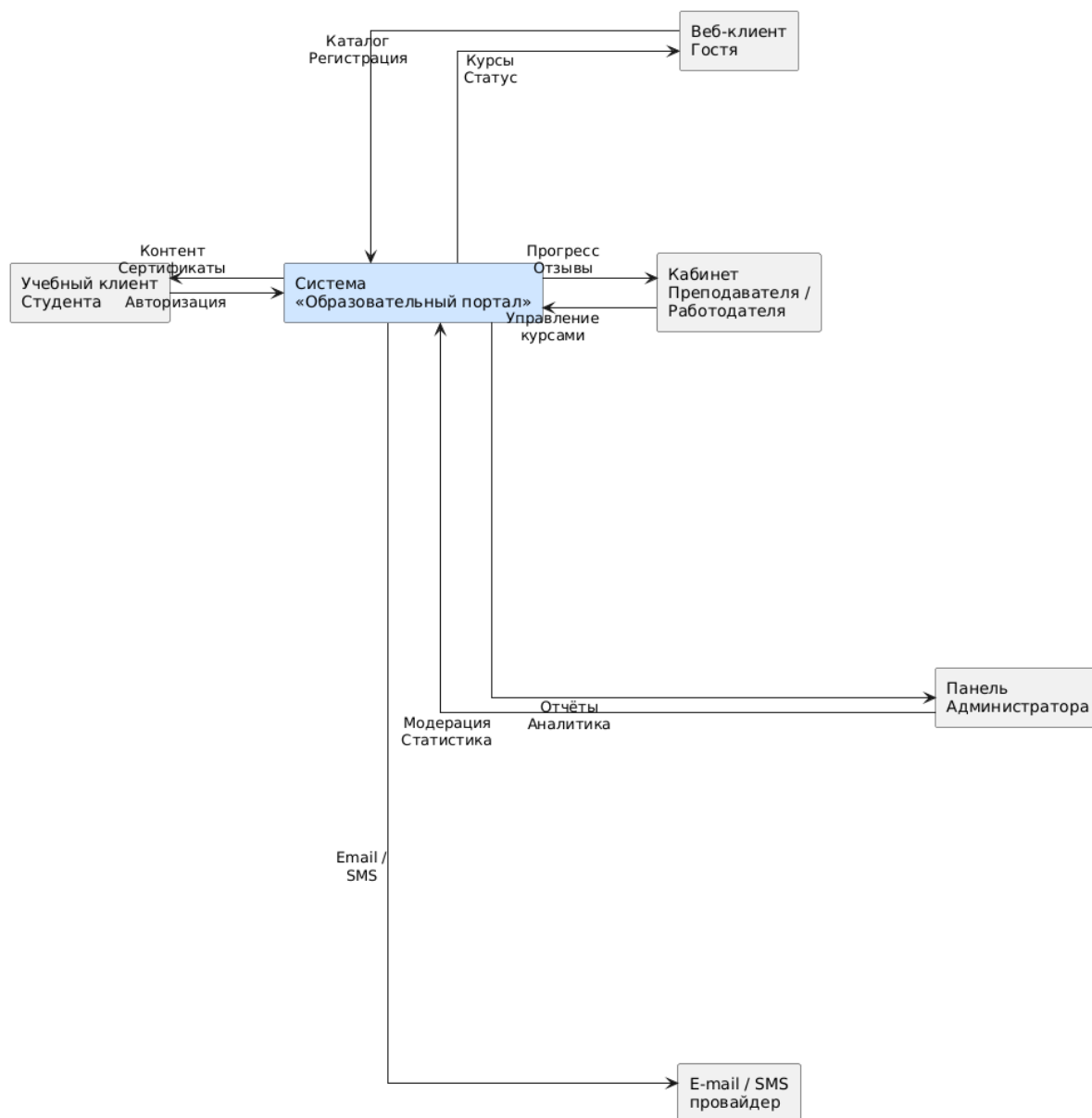


Рисунок 1 – Контекстная диаграмма системы образовательного портала

Также была разработана диаграмма потоков данных уровня 0.

Диаграмма потоков данных уровня 0 (DFD0) (рисунок 2) детализирует внутренние процессы системы, показывая, как данные перемещаются между процессами, хранилищами данных и внешними актерами. Это помогает понять внутреннюю архитектуру, выявить точки взаимодействия и определить, как реализовать функциональность.

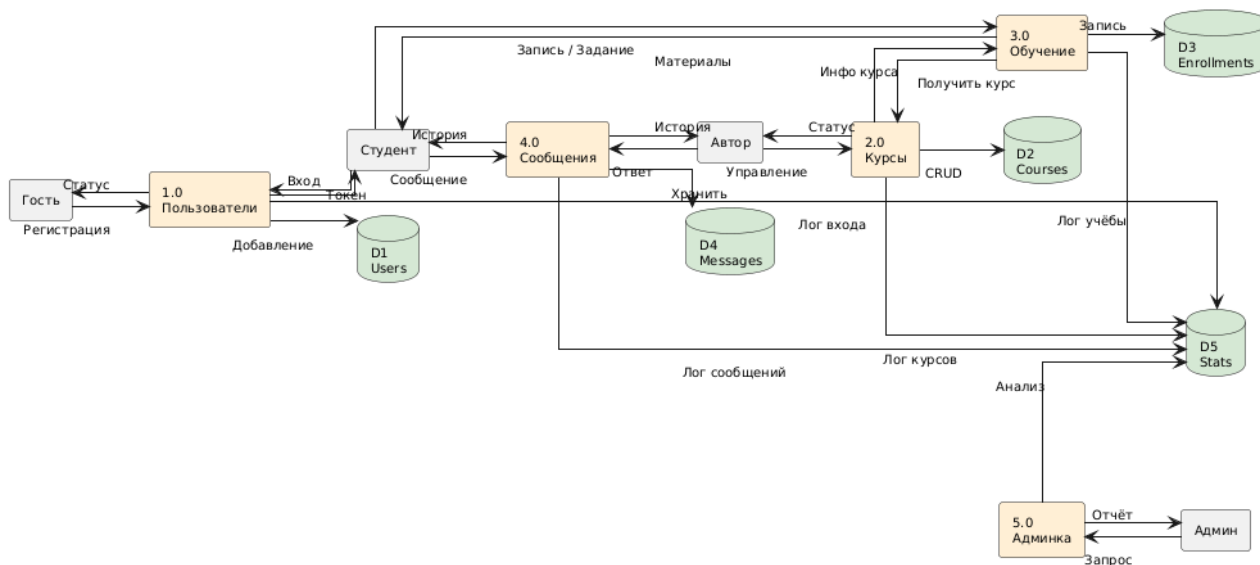


Рисунок 2 – Диаграмма потоков данных DFD0

В результате можно отметить, что сфера онлайн-обучения, ориентированная на профессиональное развитие студентов, приобретает всё большее значение в условиях быстро меняющегося рынка труда. Учет потребностей работодателей и интеграция современных образовательных технологий способствует повышению эффективности подготовки специалистов и улучшению их конкурентоспособности.

1.2 Анализ рынка программного обеспечения для ведения образовательных курсов

Для определения конкурентных преимуществ разрабатываемой платформы был проведен анализ существующих решений на рынке

образовательных платформ. Ниже представлена сравнительная таблица, включающая основные характеристики популярных платформ (таблица 1).

Таблица 1 – Сравнение образовательных платформ

Платформа/Функционал	Наличие бесплатных курсов	Сертификат по окончании	Практические задания	Проверка заданий экспертами	Карьерная поддержка	Гибкий график обучения	Форум/Чат для общения	Доступ к материалам после курса	Наличие игровых элементов
Яндекс.Практикум	+	+	+	+	+	+	+	+	+
Нетологии	+	+	+	+	+	+	+	+	-
GeekBrains	+	+	+	+	+	+	+	+	-
Skillbox	+	+	+	+	+	+	+	+	+
Coursera	+	+	+	-	-	+	+	+	-
Udemy (русскоязычные курсы)	+	+	+	-	-	+	-	+	-
Stepik	+	+	+	-	-	+	+	+	+
OTUS	-	+	+	+	+	+	+	+	-
Eduson Academy	-	+	+	+	+	+	+	-	-
Hexlet	+	+	+	+	-	+	+	+	+
Codecademy (англоязычная)	+	+	+	-	-	+	+	+	+
ВКонтакте Университет	+	-	-	-	-	+	+	+	-
Открытое образование	+	+	+	-	-	+	-	+	-
Habr	+	-	-	-	+	+	+	+	-

Общее описание платформ:

- популярность (трафик): Coursera, Skillbox и Открытое образование – лидеры по количеству пользователей;
- разнообразие курсов: Coursera и Открытое образование предлагают самые разнообразные курсы, охватывая широкий спектр тем;
- удобство использования: Coursera и Яндекс.Учебник выделяются удобным и интуитивным интерфейсом;
- качество контента: Coursera, Skillbox, и Habr предлагают высококачественные материалы и преподавателей;
- геймификация и вовлеченность: Кодварс и Хекслет – лучшие по уровню геймификации, мотивируя пользователей через игровые элементы;
- персонализация обучения: Skyeng и Яндекс.Учебник обеспечивают наиболее адаптированные траектории на основе прогресса и предпочтений учащихся;
- социальное взаимодействие: ВКонтакте.Университет и GitHub предлагают хорошие возможности для общения и совместной работы;
- гибкость траектории: Coursera и Яндекс.Учебник предлагают максимальную гибкость в обучении.

На основе анализа можно сделать вывод, что разрабатываемая платформа обладает уникальными преимуществами, такими как интеграция с рынком труда, поддержка курсов от работодателей и гибкое управление ролями пользователей; данные для анализа были получены с официальных сайтов платформ [3, 4].

1.3 Анализ функциональных требований

Функциональные требования описывают варианты использования системы (Use Cases). Основные варианты использования (ДВИ) с описанием в

виде таблицы приведены в приложении Б и в виде диаграммы вариантов использования на рисунке 3.

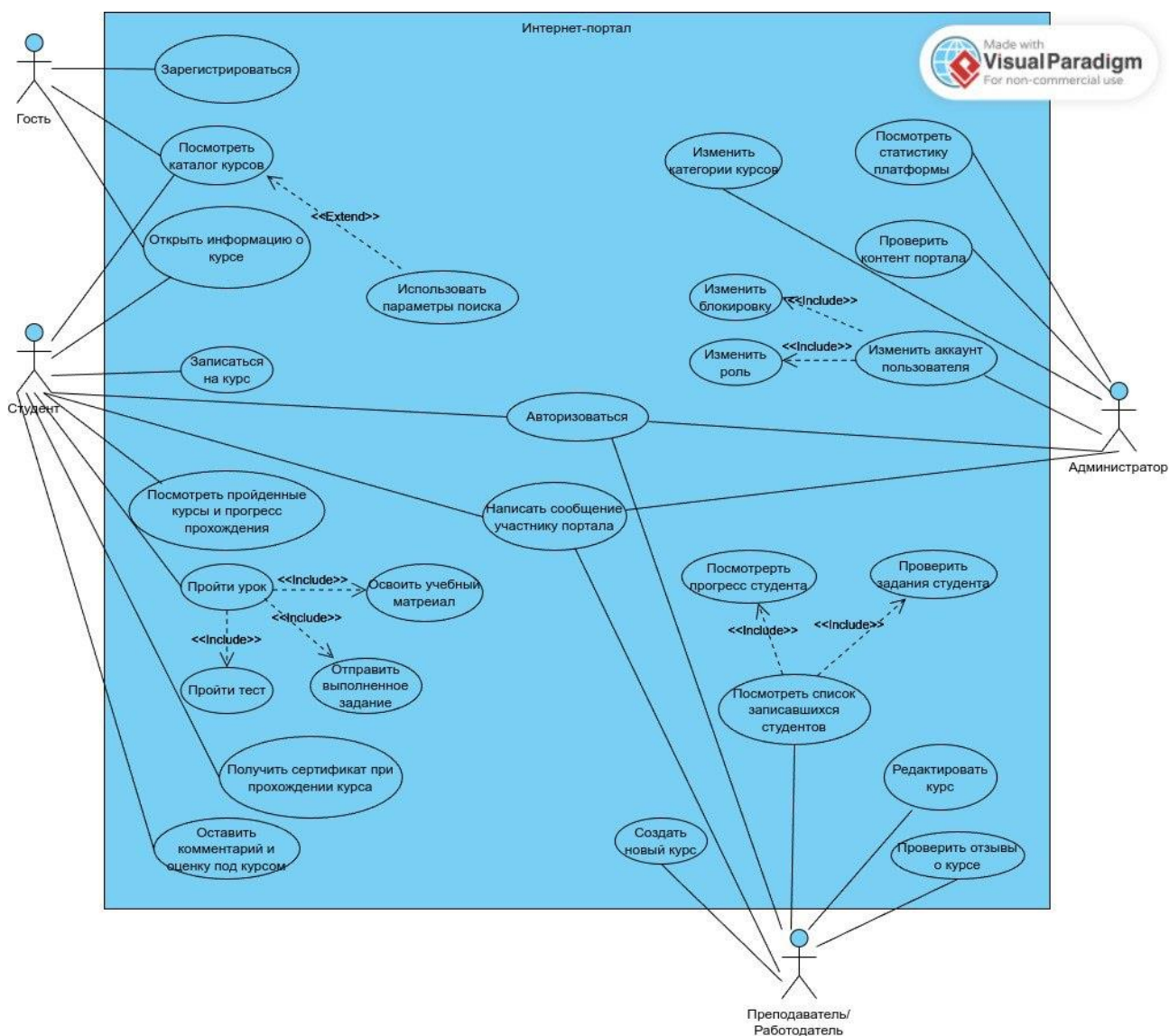


Рисунок 3 – Диаграмма вариантов использования

Функциональные требования можно разделить на 4 основные группы, основываясь на их ролевой принадлежности: к гостю, студенту, преподавателю или работодателю, администратору.

Описание вариантов использования выполнено в соответствии со стандартами UML [5].

1.4 Анализ нефункциональных требований

Нефункциональные требования определяют качество системы, ее интерфейс и безопасность в соответствии со стандартом ISO/IEC 25010 [6].

Требования к интерфейсу:

- платформа должна иметь интуитивно понятный и адаптивный пользовательский интерфейс, поддерживающий работу на различных устройствах (ПК, планшеты, смартфоны);
- время отклика интерфейса не должно превышать 2 секунд при нормальной нагрузке;
- дизайн должен соответствовать современным стандартам UX/UI, включая использование единого стиля, цветовой палитры и типографики, а также принципам доступности WCAG 2.1 [7].

Качество:

- система должна обеспечивать высокую доступность (uptime не менее 99,5%);
- производительность системы должна поддерживать одновременную работу не менее 1000 пользователей без значительного снижения скорости;
- система должна быть масштабируемой для увеличения числа пользователей и курсов.

Безопасность:

- все данные пользователей (логины, пароли, персональная информация) должны быть защищены с использованием шифрования (HTTPS для передачи данных, алгоритм bcrypt для хэширования паролей);
- доступ к функционалу должен быть ограничен в соответствии с ролями пользователей (RBAC – Role-Based Access Control);

Также система должна поддерживать защиту от распространенных атак, с использованием средств фреймворка Express.

Нефункциональные требования играют важную роль в обеспечении надежности, удобства использования и защищенности системы, создавая основу для её соответствия современным стандартам качества.

1.5 Выбор средств разработки

Для реализации платформы были выбраны следующие средства разработки:

- HTML, CSS, JavaScript – для создания пользовательского интерфейса [8];
- React.js – для построения адаптивного интерфейса [9];
- Bootstrap – для обеспечения единого стиля и адаптивности;
- Node.js (Express) – для разработки серверной части благодаря высокой скорости разработки, простоте и поддержке REST API [10];
- Sequelize – для работы с базой данных в Node.js;

База данных:

- PostgreSQL – как реляционная СУБД, обеспечивающая высокую производительность, надежность и поддержку сложных запросов;
- Nginx – как веб-сервер и обратный прокси;

В качестве инструментов разработки были выбраны:

- Git – для контроля версий;
- Visual Studio Code – как основная среда разработки;
- Postman – для тестирования API;
- pgAdmin – для управления базой данных PostgreSQL;

В итоге были выбраны все необходимые средства для разработки системы образовательного портала, которые в полной мере удовлетворяют потребности разработки приложения.

1.6 Постановка задачи

На основе проведенного анализа сформулирована следующая задача:

Разработать web-платформу образовательных курсов компаний-работодателей, обеспечивающую выполнение следующих функций:

- регистрацию и авторизацию пользователей с различными ролями (гость, студент, преподаватель/работодатель, администратор);
- управление каталогом курсов, включая создание, редактирование и просмотр курсов;
- прохождение курсов студентами, включая выполнение заданий, прохождение тестов и получение сертификатов;
- управление прогрессом студентов и проверку заданий преподавателями/работодателями;
- администрирование платформы, включая управление пользователями, категориями курсов и статистикой.

В итоге проведенного анализа были выделены необходимые функции и требования для будущего приложения.

1.7 Планирование проекта и оценка бюджета

Для планирования проекта были продуманы и добавлены в Microsoft Project 2007 основные задачи, приведенные на рисунке 4.

Каждая из задач была детализирована с указанием предполагаемой продолжительности, распределением ресурсов и определением ключевых зависимостей между этапами проекта.

При этом использовалась иерархическая структура задач, что позволило сгруппировать связанные между собой действия в логические блоки, упрощающие контроль и управление ходом выполнения работ.

		Название задачи	Длительность	Начало	Окончание	Предшественники
1		Начало реализации проекта	0 дней	Пн 07.04.25	Пн 07.04.25	
2		Анализ предметной области и требований	5 дней?	Пн 07.04.25	Пт 11.04.25	1
3		Анализ программных аналогов	1 день?	Пн 07.04.25	Пн 07.04.25	
4		Постановка задач на разработку	1 день?	Пн 07.04.25	Пн 07.04.25	
5		Анализ функциональных требований	2 дней	Вт 08.04.25	Ср 09.04.25	4
6		Анализ нефункциональных требований	1 день?	Чт 10.04.25	Чт 10.04.25	5
7		Выбор инструментальных средств	1 день?	Пт 11.04.25	Пт 11.04.25	6
8		Анализ предметной области и требований завершен	0 дней	Пт 11.04.25	Пт 11.04.25	7
9		Проектирование системы	7 дней	Пн 14.04.25	Вт 22.04.25	2
10		Проектирование сервисов	2 дней	Пн 14.04.25	Вт 15.04.25	
11		Проектирование клиентской части	2 дней	Ср 16.04.25	Чт 17.04.25	10
12		Проектирование базы данных	2 дней	Пт 18.04.25	Пн 21.04.25	11
13		Проектирование средств контейнеризации	1 день	Вт 22.04.25	Вт 22.04.25	12
14		Проектирование системы завершено	0 дней	Вт 22.04.25	Вт 22.04.25	13
15		Реализация системы	9 дней	Ср 23.04.25	Пн 05.05.25	9
16		Реализация базы данных	2 дней	Ср 23.04.25	Чт 24.04.25	
17		Реализация клиентской части	3 дней	Пт 25.04.25	Вт 29.04.25	16
18		Реализация серверной части	4 дней	Ср 30.04.25	Пн 05.05.25	17
19		Реализация системы завершена	0 дней	Вт 29.04.25	Вт 29.04.25	17
20		Тестирование системы и анализ кода	8 дней?	Вт 06.05.25	Чт 15.05.25	15
21		Функциональное тестирование	2 дней	Вт 06.05.25	Ср 07.05.25	
22		Структурное тестирование	2 дней	Чт 08.05.25	Пт 09.05.25	21
23		Модульное тестирование	3 дней	Пн 12.05.25	Ср 14.05.25	22
24		Анализ качества кода	1 день?	Чт 15.05.25	Чт 15.05.25	23
25		Завершение тестирования и анализа кода	0 дней	Чт 15.05.25	Чт 15.05.25	24
26		Завершение проекта	8 дней?	Пт 16.05.25	Вт 27.05.25	20
27		Составление документации	7 дней	Пт 16.05.25	Пн 26.05.25	
28		Внедрение проекта	1 день?	Вт 27.05.25	Вт 27.05.25	27
29		Проект завершен	0 дней	Вт 27.05.25	Вт 27.05.25	28
30		Конец проекта	0 дней	Вт 27.05.25	Вт 27.05.25	26;2

Рисунок 4 – Задачи для реализации проекта

Основные ресурсы, необходимые для реализации проекта, приведены на рисунке 5.

		Название ресурса	Тип	Единицы измерения материалов	Краткое название	Группа	Макс. единиц	Стандартная ставка	Ставка сверхурочных	Затраты на исполыз.	Начисление	Базовый календарь
1		Программист 1	Трудовой		П		100%	300,00 Р/ч	400,00 Р/ч	0,00 Р	Пропорциональное	Стандартный
2		Программист 2	Трудовой		П		100%	200,00 Р/ч	300,00 Р/ч	0,00 Р	Пропорциональное	Стандартный
3		Аналитик	Трудовой		А		100%	200,00 Р/ч	250,00 Р/ч	0,00 Р	Пропорциональное	Стандартный
4		Тестировщик	Трудовой		Т		100%	180,00 Р/ч	230,00 Р/ч	0,00 Р	Пропорциональное	Стандартный

Рисунок 5 – Ресурсы на реализацию проекта

Для каждой задачи были распределены ресурсы. Результат приведен на рисунке 6.

	Название задачи	Длительность	Начало	Окончание	Предшественники	Названия ресурсов
1	Начало реализации проекта	0 дней	Пн 07.04.25	Пн 07.04.25		
2	Анализ предметной области и требований	6 дней?	Пн 07.04.25	Пн 14.04.25	1	
3	Анализ программных аналогов	1 день?	Пн 07.04.25	Пн 07.04.25		Аналитик
4	Постановка задач на разработку	1 день?	Пн 07.04.25	Пн 07.04.25		Аналитик
5	Анализ функциональных требований	2 дней	Вт 08.04.25	Ср 09.04.25	4	Аналитик
6	Анализ нефункциональных требований	1 день?	Чт 10.04.25	Чт 10.04.25	5	Аналитик
7	Выбор инструментальных средств	2 дней?	Пт 11.04.25	Пн 14.04.25	6	Аналитик; Программист 1[50%]
8	Анализ предметной области и требований завершен	0 дней	Пн 14.04.25	Пн 14.04.25	7	
9	Проектирование системы	6,33 дней	Вт 15.04.25	Ср 23.04.25	2	
10	Проектирование сервисов	1,33 дней	Вт 15.04.25	Ср 16.04.25		Программист 1; Программист 2[50%]
11	Проектирование клиентской части	2 дней	Ср 16.04.25	Пт 18.04.25	10	Программист 1; Программист 2[50%]
12	Проектирование базы данных	2 дней	Пт 18.04.25	Вт 22.04.25	11	Программист 1; Программист 2[50%]
13	Проектирование средств контейнеризации	1 день	Вт 22.04.25	Ср 23.04.25	12	Программист 1; Программист 2[50%]
14	Проектирование системы завершено	0 дней	Ср 23.04.25	Ср 23.04.25	13	
15	Реализация системы	8,43 дней	Ср 23.04.25	Пн 05.05.25	9	
16	Реализация базы данных	1,43 дней	Ср 23.04.25	Чт 24.04.25		Программист 1[70%]; Программист 2
17	Реализация клиентской части	3 дней	Чт 24.04.25	Вт 29.04.25	16	Программист 1[70%]; Программист 2
18	Реализация серверной части	4 дней	Вт 29.04.25	Пн 05.05.25	17	Программист 1; Программист 2
19	Реализация системы завершена	0 дней	Вт 29.04.25	Вт 29.04.25	17	
20	Тестирование системы и анализ кода	7,67 дней?	Пн 05.05.25	Чт 15.05.25	15	
21	Функциональное тестирование	2 дней	Пн 05.05.25	Ср 07.05.25		Тестировщик
22	Структурное тестирование	2 дней	Ср 07.05.25	Пт 09.05.25	21	Тестировщик
23	Модульное тестирование	3 дней	Пт 09.05.25	Ср 14.05.25	22	Тестировщик
24	Анализ качества кода	0,67 дней?	Ср 14.05.25	Чт 15.05.25	23	Тестировщик; Программист 1[50%]
25	Завершение тестирования и анализа кода	0 дней	Чт 15.05.25	Чт 15.05.25	24	
26	Завершение проекта	5,67 дней?	Чт 15.05.25	Пт 23.05.25	20	
27	Составление документации	4,67 дней	Чт 15.05.25	Чт 22.05.25		Программист 1[50%]; Программист 2[50%]; Анал
28	Внедрение проекта	1 день?	Чт 22.05.25	Пт 23.05.25	27	Программист 1
29	Проект завершен	0 дней	Пт 23.05.25	Пт 23.05.25	28	
30	Конец проекта	0 дней	Пт 23.05.25	Пт 23.05.25	26;2	

Рисунок 6 – Ресурсы задач проекта

Была построена диаграмма Ганта, которая приведена на рисунке 7.

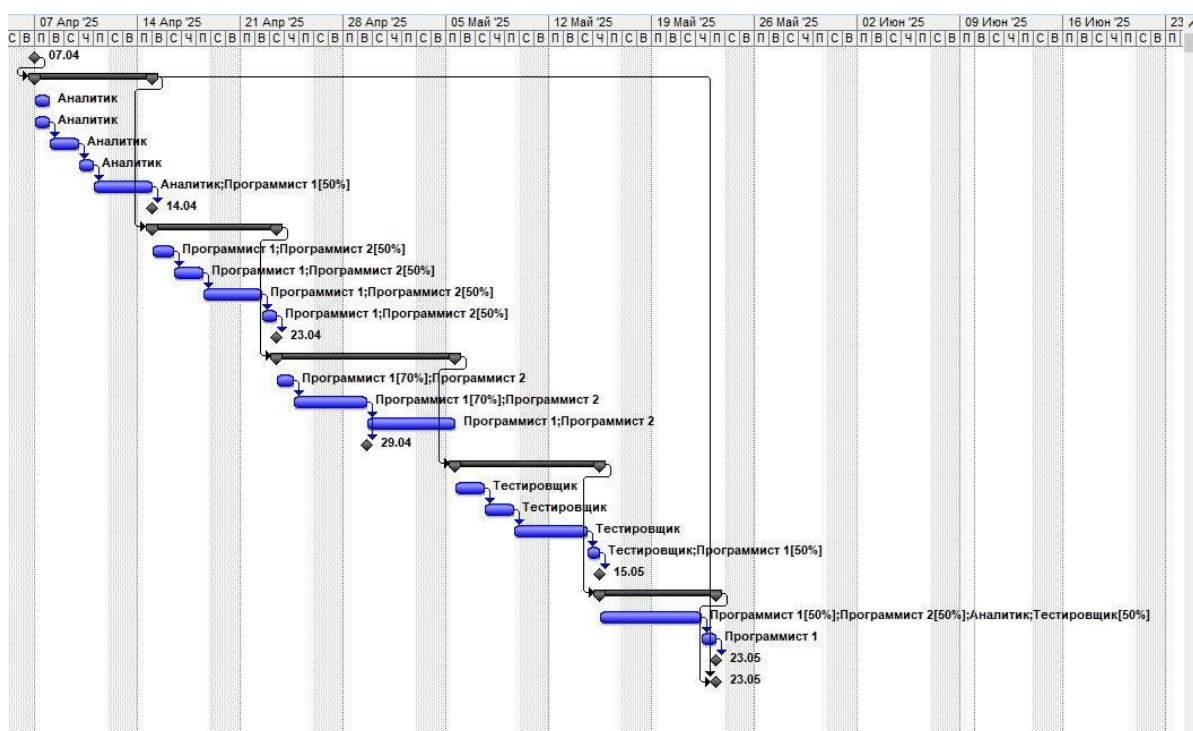


Рисунок 7 – Диаграмма Ганта

Общая статистика по проекту приведена на рисунке 8.

Статистика проекта для 'Диплом'

	Начало	Окончание
Текущее	Пн 07.04.25	Пт 23.05.25
Базовое	НД	НД
Фактическое	НД	НД
Отклонение	0д	0д

	Длительность	Трудозатраты	Затраты
Текущие	34,1д?	371,2ч	86 346,67 Р
Базовые	0д?	0ч	0,00 Р
Фактические	0д	0ч	0,00 Р
Оставшиеся	34,1д?	371,2ч	86 346,67 Р

Процент завершения

Длительность: 0% Трудозатраты: 0%

Заккрыть

Рисунок 8 – Статистика по проекту

Таким образом, из полученной статистики можно сделать вывод, что длительность реализации проекта составляет 34 дня, все поставленные задачи решены полностью, и проект завершен на 100%. Затраты составляют 86346 руб., что говорит об экономической выгоде покупки разработанного продукта относительно существующих на рынке наиболее популярных аналогов.

2 Проектирование образовательного портала

2.1 Проектирование архитектуры образовательного портала

Архитектура платформы основана на клиент-серверной модели с использованием REST API для взаимодействия между клиентской и серверной частями [18]. Серверная часть реализована в виде сервисной архитектуры, где функционал разделен на независимые микросервисы, взаимодействующие с единой базой данных. Выбор REST API обусловлен его простотой, масштабируемостью и широкой поддержкой в современных фреймворках [12]. Альтернативы, такие как GraphQL, были отклонены из-за большей сложности реализации.

Кроме того, архитектура построена с учетом принципов масштабируемости и отказоустойчивости. Каждый микросервис может быть развернут и масштабирован независимо, что позволяет гибко реагировать на рост нагрузки и обеспечивать стабильную работу системы.

Основные компоненты архитектуры:

- клиентская часть приложения: отвечает за отображение интерфейса и взаимодействие с пользователем, реализована с использованием React.js и Bootstrap [19];
- серверная часть приложения: отвечает за обработку запросов, бизнес-логику и взаимодействие с базой данных, реализована с использованием Node.js (Express) и разделена на микросервисы [20];
- база данных: хранит информацию о пользователях, курсах, прогрессе, сертификатах и других сущностях; реализована с использованием PostgreSQL [21].

Архитектура системы в виде диаграммы компонентов представлена на рисунке 9.

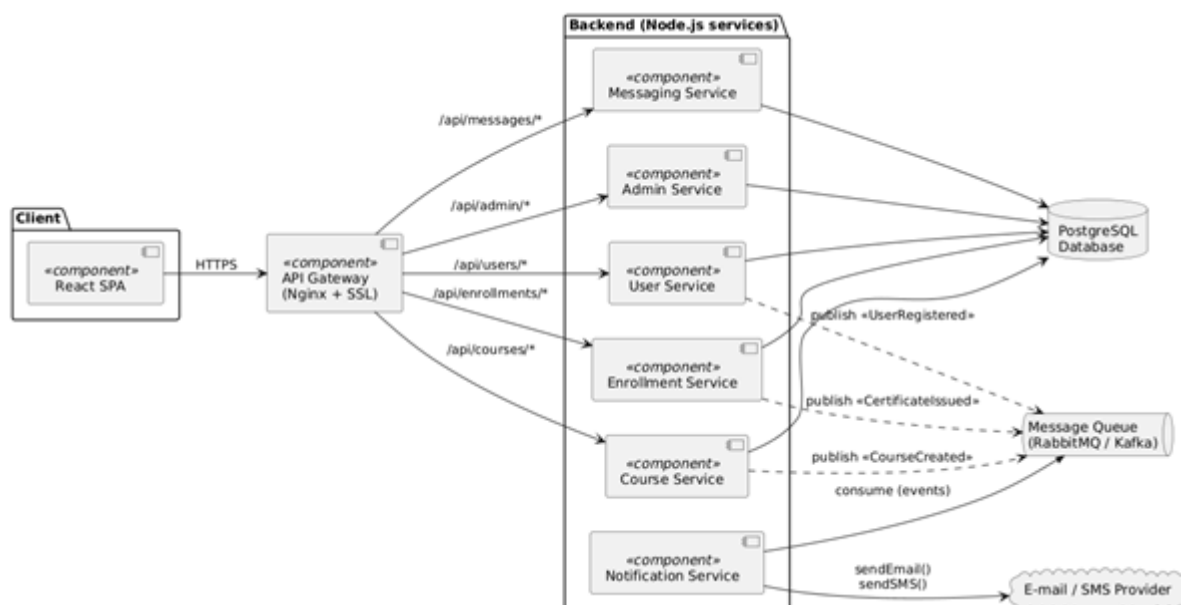


Рисунок 9 – Диаграмма компонентов

Далее рассматривается архитектура каждого компонента более подробно.

2.2 Проектирование пользовательского интерфейса

Пользовательский интерфейс проектируется с учетом принципов UX/UI, обеспечивая удобство и интуитивность.

Основные элементы интерфейса:

- главная страница: отображает каталог курсов, поиск и фильтры;
- страница курса: содержит описание курса, список уроков, отзывы и кнопку записи;
- личный кабинет студента: отображает прогресс, пройденные курсы, сертификаты и настройки профиля;
- личный кабинет преподавателя/работодателя: содержит список курсов, студентов, статистику и инструменты редактирования;
- панель администратора: включает управление пользователями, курсами, категориями и статистикой.

В проектировании используются принципы доступности WCAG 2.1 [7].

Макеты пользовательского интерфейса для домашней страницы и панели администратора приведены на рисунках 10, 11 соответственно.

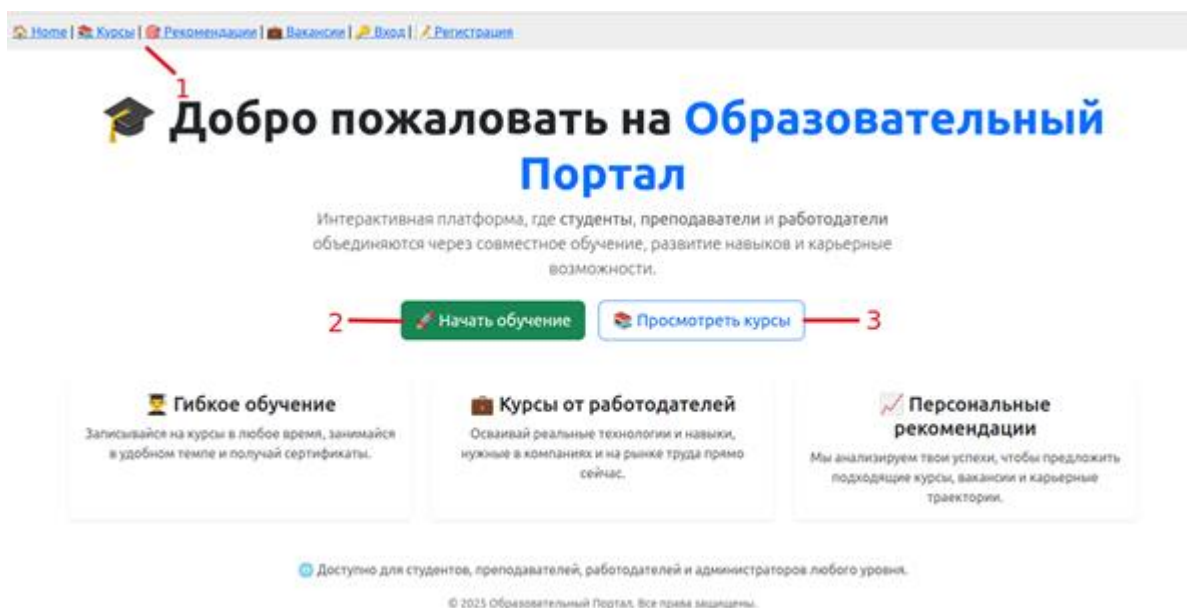


Рисунок 10 – Макет домашней страницы

Макет домашней страницы состоит из:

- 1) панель навигации;
- 2) кнопка перехода к странице личного кабинета или регистрации;
- 3) кнопка перехода к странице курсов.

Домашняя страница ориентирована на конечного пользователя — обучающегося, и предоставляет быстрый доступ к основным разделам платформы, таким как каталог курсов, новости, рекомендации.

Особое внимание уделено адаптивности дизайна — макет предполагает корректное отображение как на компьютерах, так и на мобильных устройствах. Визуальные элементы, такие как кнопки, иконки и баннеры, выполнены в едином стиле, что обеспечивает целостность пользовательского опыта.

Далее рассмотрим макет панели администратора.

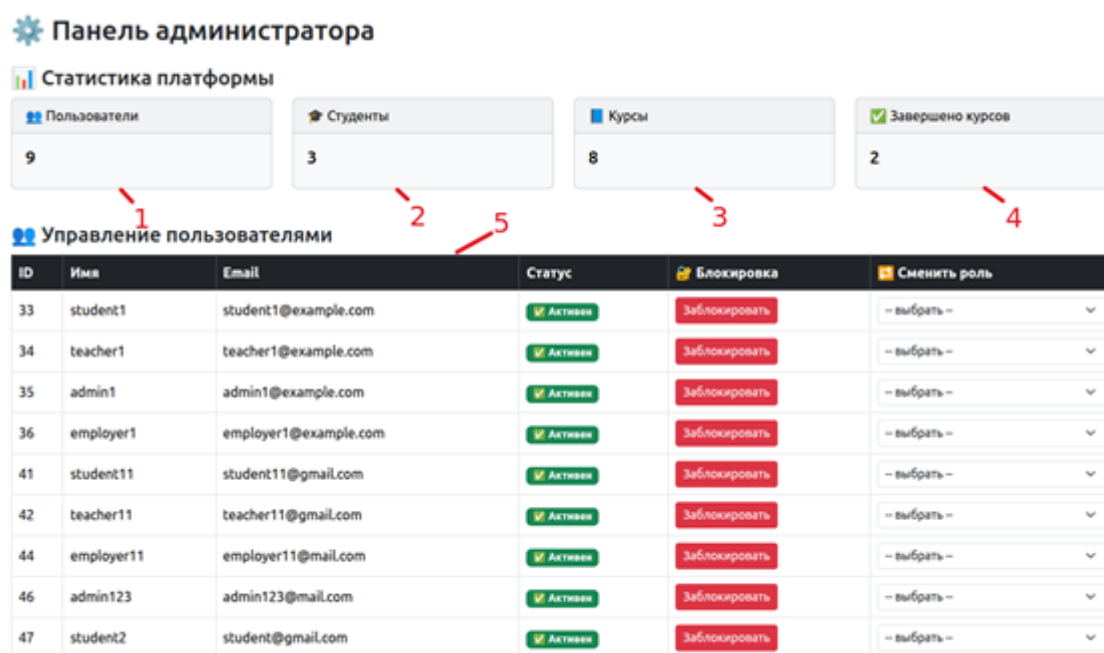


Рисунок 11 – Макет панели администратора

Макет панели администратора состоит из:

- 1) панель количества зарегистрированных на портале пользователей;
- 2) панель количества студентов на портале;
- 3) панель количества курсов на портале;
- 4) панель количества завершенных студентами курсов на портале.

Проектирование пользовательского интерфейса с опорой на принципы UX/UI и актуальные гайдлайны способствует созданию удобной, понятной и доступной среды взаимодействия для всех категорий пользователей.

2.3 Проектирование базы данных

Проектирование базы данных включает разработку концептуальной, логической и физической моделей, а также развертывание базы данных в среде управления PostgreSQL. Проектирование выполнено с использованием

методологии ER-диаграмм и нормализации данных до третьей нормальной формы (3NF) для обеспечения целостности и минимизации избыточности [14].

2.3.1 Разработка концептуальной модели базы данных

Концептуальная модель базы данных представляет собой высокоуровневое описание данных и их взаимосвязей, независимое от конкретной СУБД. На этом этапе определены основные сущности, их атрибуты и связи между ними.

Основные сущности, выделенные в рамках платформы, включают:

- пользователи (Users) – хранит информацию о зарегистрированных пользователях системы;
- роли пользователей (Admins, Students, Teachers, Employers) – определяют роли пользователей с дополнительными атрибутами, специфичными для каждой роли;
- курсы (Courses) – содержат информацию о курсах, предлагаемых на платформе;
- модули (Modules) – описывают структуру курсов, включая последовательность и обязательность модулей;
- уроки (Lessons) – содержат учебные материалы, входящие в модули.
- записи на курсы (Enrollments) – фиксируют участие студентов в курсах;
- сертификаты (Certificates) – хранят данные о выданных сертификатах;
- навыки (Skills) – описывают профессиональные навыки, связанные с курсами и студентами;
- требования к вакансиям (Vacancy Requirements).

Также:

- навыки для вакансий (Vacancy Skills) – связывают вакансии с требуемыми навыками;
- заявки на вакансии (Applications) – фиксируют заявки студентов на вакансии;
- документы к заявкам (Application Documents) – содержат документы, приложенные к заявкам;
- траектории (Role Trajectories) – описывают карьерные пути, связанные с курсами и студентами;
- курсы траекторий (Trajectory Courses) – связывают траектории с курсами;
- траектории студентов (Student Trajectories) – фиксируют участие студентов в траекториях;
- достижения студентов (Student Achievements) – содержат информацию о достижениях студентов.

Описание сущностей концептуальной модели представлено в виде таблицы в приложении Б.

Связи между сущностями определены на основе функциональных требований. Например, сущность Users связана с сущностями Admins, Students, Teachers, Employers через отношение один-к-одному, а сущность Courses связана с сущностью Teachers через отношение многие-к-одному. Спецификация связей между сущностями представлена в виде таблицы в приложении Б.

Концептуальная модель данных представлена в виде ER-диаграммы на рисунке 12.

Концептуальная модель базы данных проектировалась согласно последним стандартам сферы и в полной мере выполняет задачу концептуального описания базы данных системы.

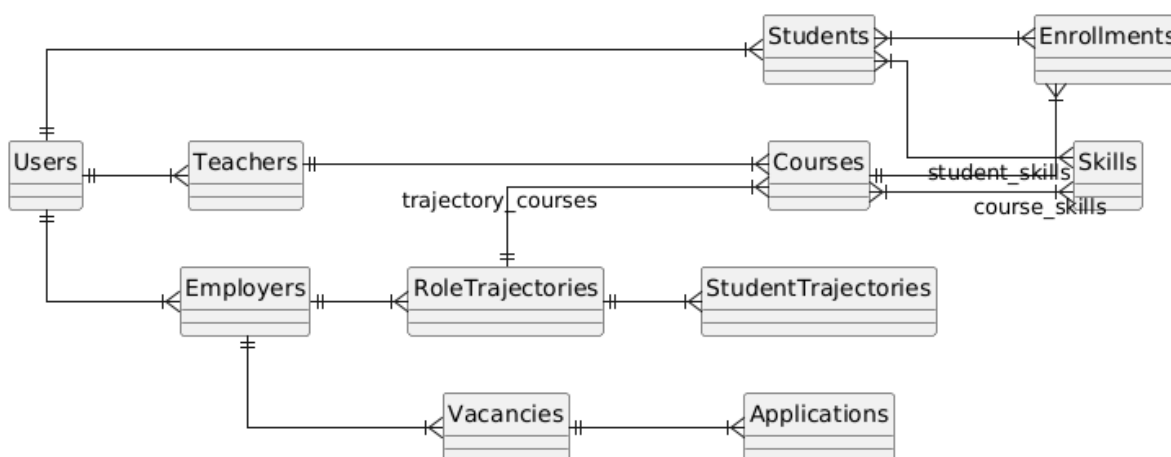


Рисунок 12 – Концептуальная модель базы данных системы образовательный портал

Концептуальная модель позволяет систематизировать структуру данных и заложить основу для дальнейшего проектирования базы данных вне зависимости от выбранной СУБД, обеспечивая логичность и полноту представления предметной области.

2.3.2 Разработка логической модели базы данных

Логическая модель базы данных детализирует концептуальную модель, определяя таблицы, их атрибуты, типы данных и связи, но оставаясь независимой от конкретной СУБД. На этом этапе данные нормализованы до третьей нормальной формы (3NF) для устранения избыточности и обеспечения целостности.

Атрибуты таблиц имеют чётко определённые типы данных, соответствующие их назначению: числовые типы – для идентификаторов и баллов, символьные типы – для имён и описаний, логические – для флагов активности и состояния, а также даты и временные метки – для отслеживания событий (например, даты регистрации, завершения курса, времени прохождения

теста).

Пример описания основных таблиц:

- таблица `users` содержит поля: `id` (целочисленный идентификатор, первичный ключ), `username` (строка, уникальное имя пользователя), `email` (строка, уникальный адрес электронной почты), `password_hash` (строка, хэш пароля), `is_blocked` (логическое значение, статус блокировки), `created_at` (дата и время создания записи);
- таблица `courses` содержит поля: `id` (целочисленный идентификатор, первичный ключ), `title` (строка, название курса), `description` (текст, описание курса), `teacher_id` (целочисленный идентификатор преподавателя, внешний ключ), `created_at` (дата и время создания записи), `thumbnail_url` (строка, URL изображения), `difficulty` (строка, уровень сложности).

Связи между таблицами реализованы через внешние ключи. Например, поле `teacher_id` в таблице `courses` ссылается на поле `user_id` в таблице `teachers`, что обеспечивает связь многие-к-одному. Для обеспечения уникальности записей в таблицах, представляющих отношения многие-ко-многим (например, `trajectory_courses`), используются составные первичные ключи. Спецификация отношений логической модели представлена в виде таблицы в приложении Б.

Таким образом, логическая модель базы данных представляет собой целостную, взаимосвязанную систему, охватывающую как основные учебные процессы, так и адаптивные элементы взаимодействия с пользователем. Её чёткая структура, основанная на принципах реляционного моделирования и поддержке ссылочной целостности, обеспечивает надёжную основу для дальнейшей физической реализации и масштабирования образовательной платформы.

Логическая модель данных представлена на рисунке 13, с подробным описанием атрибутов.

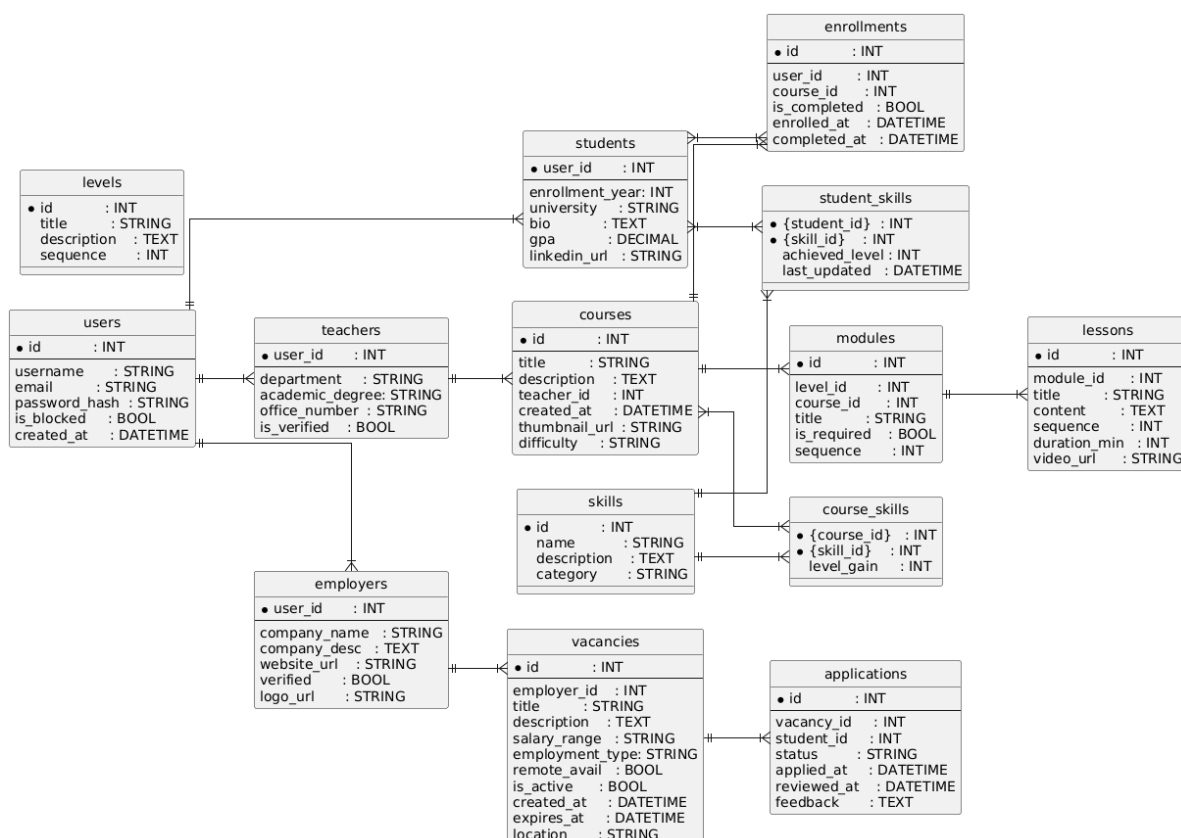


Рисунок 13 – Логическая модель базы данных для системы образовательного портала

Логическая модель базы данных обеспечивает более точное представление структуры данных, позволяя оптимизировать хранение информации и повысить её целостность за счёт нормализации и четко определённых связей между сущностями.

2.3.3 Разработка физической модели базы данных

Физическая модель базы данных определяет реализацию логической модели в конкретной СУБД (PostgreSQL), включая типы данных, индексы, ограничения и оптимизацию производительности.

Пример описания физической реализации основных таблиц:

Таблица users состоит из:

- поле `id` имеет тип `SERIAL` (автоинкрементный целочисленный идентификатор) и является первичным ключом;
- поле `username` имеет тип `VARCHAR(255)` с ограничением уникальности (`UNIQUE`);
- поле `email` имеет тип `VARCHAR(255)` с ограничением уникальности (`UNIQUE`);
- поле `password_hash` имеет тип `VARCHAR(255)` с ограничением `NOT NULL`;
- поле `is_blocked` имеет тип `BOOLEAN` с значением по умолчанию `FALSE`;
- поле `created_at` имеет тип `TIMESTAMP` с значением по умолчанию `NOW()`.

Таблица `users` используется для хранения данных пользователей: студентов, преподавателей, работодателей и администраторов образовательной платформы.

Таблица `courses` состоит из:

- поле `id` имеет тип `SERIAL` и является первичным ключом;
- поле `title` имеет тип `VARCHAR(255)` с ограничением `NOT NULL`;
- поле `description` имеет тип `TEXT` с ограничением `NOT NULL`;
- поле `teacher_id` имеет тип `INTEGER` с ограничением `NOT NULL` и является внешним ключом, ссылающимся на таблицу `teachers`;
- поле `created_at` имеет тип `TIMESTAMP` с значением по умолчанию `NOW()`;
- поле `thumbnail_url` имеет тип `VARCHAR(255)`;
- поле `difficulty` имеет тип `VARCHAR(255)`.

Полная физическая модель данных представлена на рисунке 14.

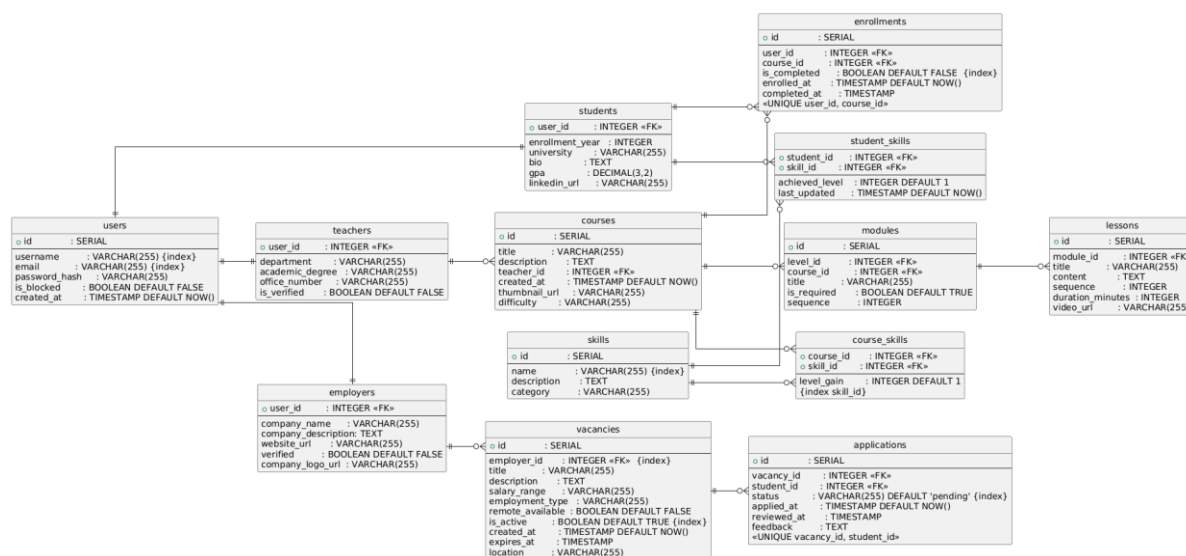


Рисунок 14 – Физическая модель базы данных системы образовательного портала

Для повышения производительности созданы индексы на часто используемых полях, таких как `users.username`, `users.email`, `enrollments.is_completed`. Целостность данных обеспечивается с использованием внешних ключей с политикой удаления `ON DELETE CASCADE` и транзакций, поддерживаемых PostgreSQL.

2.3.4 Развертывание базы данных в среде pgAdmin 4

Развертывание базы данных выполнено в среде управления PostgreSQL – pgAdmin 4, которая предоставляет графический интерфейс для администрирования баз данных.

Преимуществом использования pgAdmin 4 является возможность выполнения SQL-запросов и скриптов прямо в интерфейсе, что значительно упрощает наполнение базы данных тестовыми данными, проведение отладки и выполнение миграций. Также в процессе развёртывания применялись механизмы резервного копирования, экспорта и импорта данных, что обеспечило дополнительную надёжность работы с системой.

Процесс развертывания включает следующие шаги:

- 1) установка PostgreSQL и pgAdmin 4 на локальной машине;
- 2) создание новой базы данных с именем edu_platform через интерфейс pgAdmin 4 (вкладка Databases, Create, Database);
- 3) создание SQL-скрипта, содержащего команды для создания таблиц, индексов и ограничений, на основе физической модели;
- 4) выполнение SQL-скрипта в pgAdmin 4 через вкладку Query Tool для создания структуры базы данных;
- 5) проверка корректности развертывания путем выполнения тестовых запросов (например, `SELECT * FROM users;`) и просмотра структуры таблиц в интерфейсе pgAdmin 4;
- 6) настройка пользователя базы данных с ограниченными правами для использования в приложении (например, создание пользователя app_user с правами на чтение и запись в таблицы).

После создания объектов базы данных, была сгенерирована диаграмма данных, представленная на рисунке 15.

Особое внимание при проектировании базы данных было уделено соблюдению принципов нормализации, что позволило минимизировать избыточность данных и улучшить целостность информации. При этом структура остаётся достаточно гибкой для масштабирования и расширения функциональности в будущем. Например, может быть легко добавлена поддержка новых типов заданий или форм сертификации без кардинальных изменений в архитектуре.

Диаграмма данных включает основные сущности, такие как пользователи, курсы, задания, результаты тестов, сертификаты, а также таблицы для управления ролями, доступом и системными настройками.

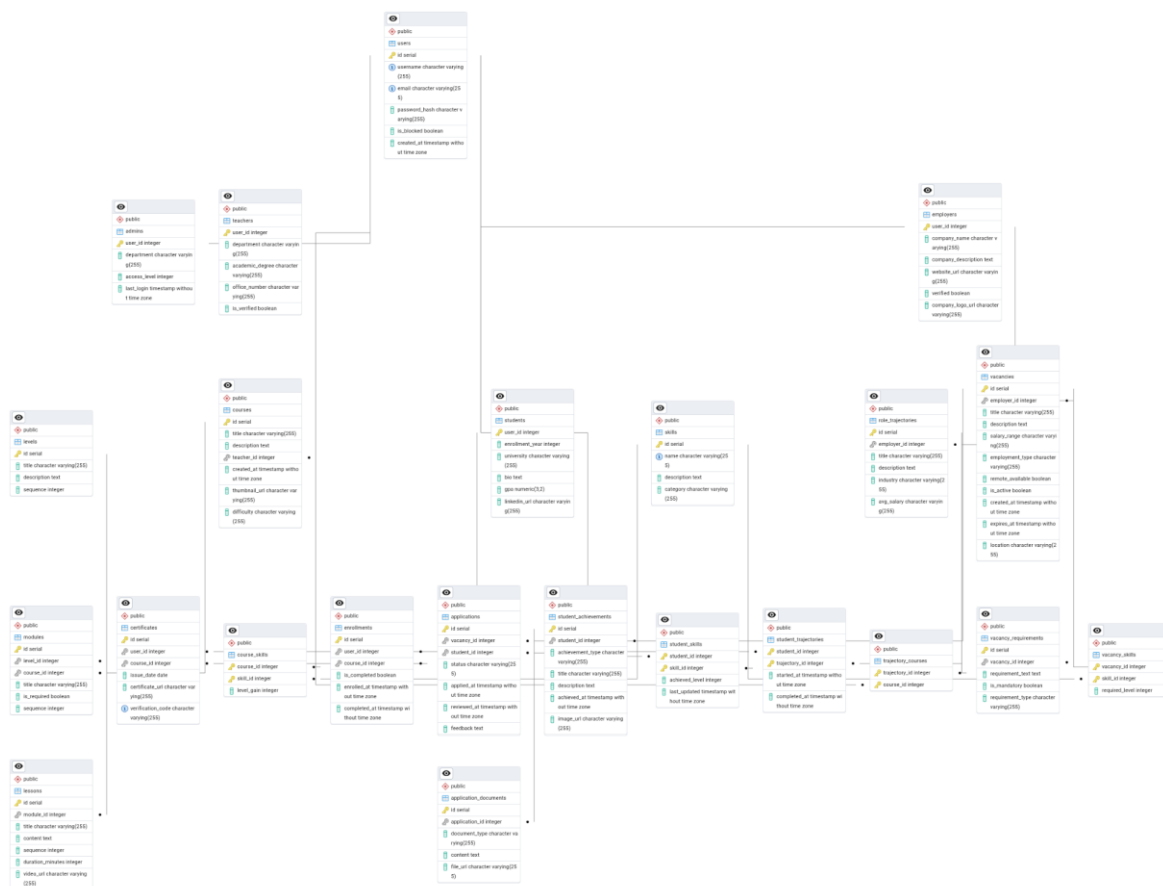


Рисунок 15 – Схема базы данных для образовательного портала

Развернутая база данных готова к интеграции с серверной частью приложения, реализованной на Node.js с использованием ORM Sequelize для упрощения взаимодействия с PostgreSQL.

2.4 Проектирование серверной части образовательного портала

Серверная часть платформы отвечает за обработку запросов, бизнес-логику и взаимодействие с базой данных. Разработка API выполнена в соответствии с RESTful-принципами [15]. Серверная часть реализована в виде сервисной архитектуры, где функционал разделен на независимые микросервисы, взаимодействующие с единой базой данных PostgreSQL.

Основные компоненты серверной части:

- маршруты (Routes): определяют доступные конечные точки API;
- контроллеры (Controlllers): обрабатывают HTTP-запросы и возвращают ответы в формате JSON;
- модели (Models): описывают структуру данных и взаимодействие с базой данных через Sequelize.

Тестирование серверной части будет проводиться с использованием метода попарного тестирования и нагрузочных тестов с помощью инструментов JMeter и Postman.

2.4.1 Проектирование сервиса пользователей

Сервис пользователей (User Service) отвечает за регистрацию, авторизацию, управление ролями и профилями пользователей. Сервис реализован как отдельное Express-приложение, взаимодействующее с базой данных через Sequelize.

Основные функции сервиса:

- регистрация пользователей с различными ролями (студент, преподаватель/работодатель, администратор);
- аутентификация пользователей с использованием JWT (JSON Web Tokens);
- управление профилями пользователей (просмотр, редактирование);
- управление ролями пользователей (для администратора).

Основные конечные точки API: POST /api/users/register – регистрация пользователя; POST /api/users/login – авторизация; GET /api/users/:user_id – получение профиля пользователя; PUT /api/users/:user_id – обновление профиля; PUT /api/users/:user_id/role – изменение роли (для администратора).

Аутентификация реализована с использованием JWT-токенов, которые

передаются в заголовке каждого запроса (Authorization: Bearer <token>). На рисунке 16 представлена диаграмма классов для сервиса пользователей.

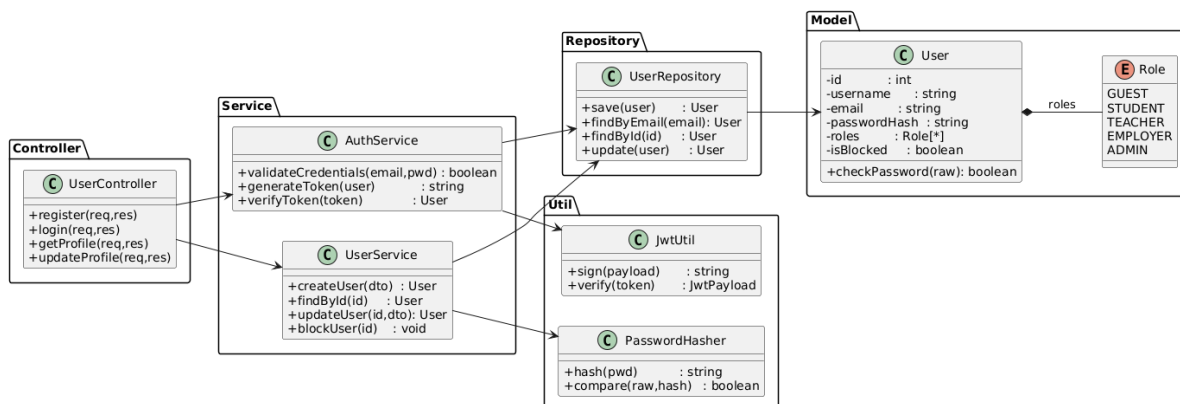


Рисунок 16 – Диаграмма классов сервиса пользователей

Защита от атак обеспечивается с использованием библиотеки `helmet` для защиты от XSS и `csurf` для защиты от CSRF.

2.4.2 Проектирование сервиса курсов

Сервис курсов (Course Service) отвечает за управление курсами, модулями, уроками и отзывами. Сервис реализован как отдельное Express-приложение, взаимодействующее с базой данных через Sequelize.

Основные функции сервиса:

- создание и редактирование курсов (для преподавателя/работодателя);
- просмотр каталога курсов с фильтрами (для всех пользователей);
- управление отзывами о курсах (для студентов и преподавателей/работодателей).

Основные конечные точки API: GET /api/courses – получение списка курсов (с фильтрами); POST /api/courses – создание курса (для преподавателя/работодателя); GET /api/courses/:course_id – получение

информации о курсе; PUT /api/courses/:course_id – редактирование курса; GET /api/courses/:course_id/reviews – получение отзывов о курсе.

На рисунке 17 представлена диаграмма классов для сервиса курсов.

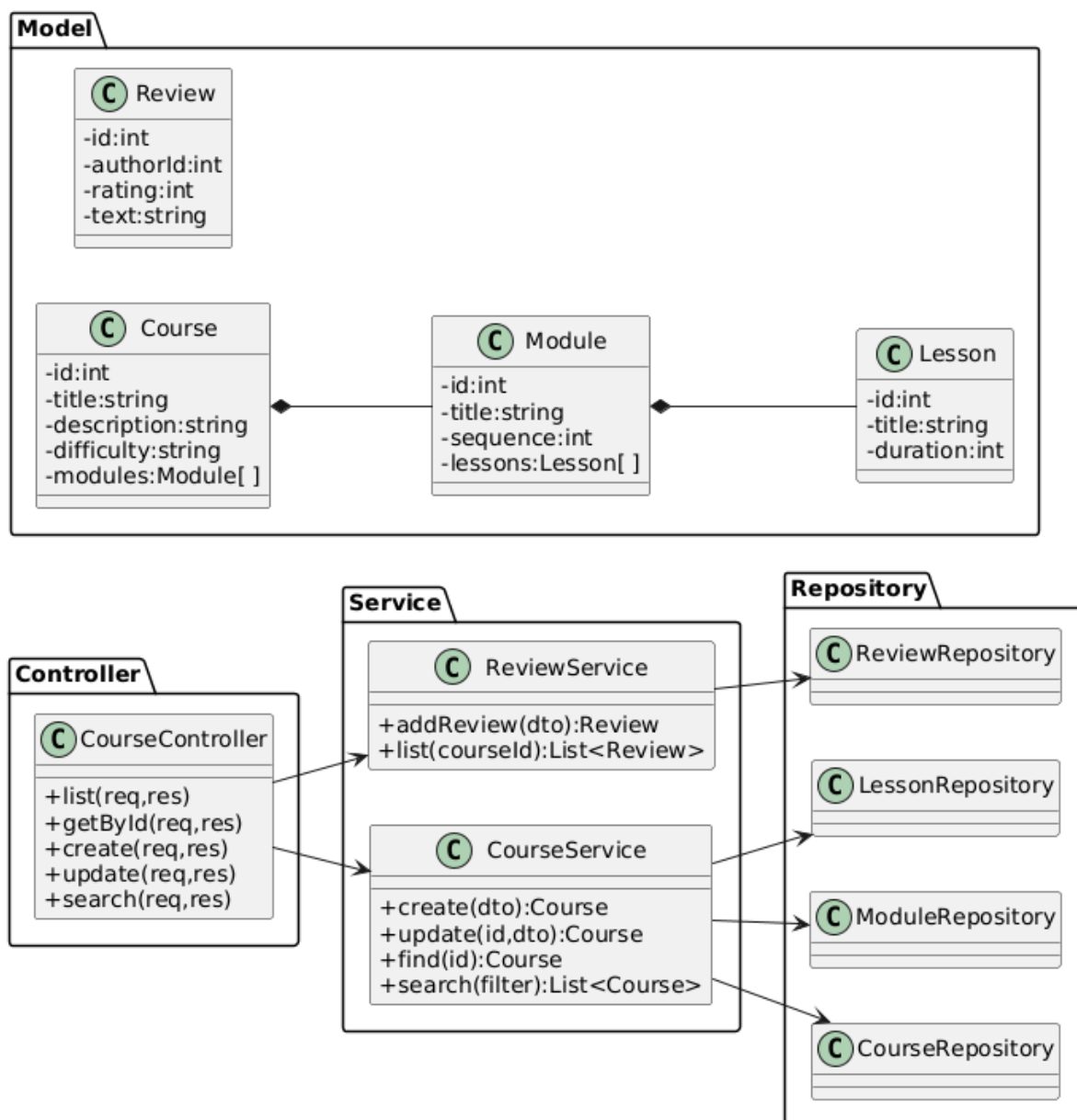


Рисунок 17 – Диаграмма классов сервиса курсов

Сервис взаимодействует с сервисом пользователей для проверки ролей (например, только преподаватель/работодатель может создавать курсы).

2.4.3 Проектирование сервиса обучения

Сервис обучения (Enrollment Service) отвечает за запись на курсы, отслеживание прогресса и выдачу сертификатов. Сервис реализован как отдельное Express-приложение, взаимодействующее с базой данных через Sequelize.

Основные функции сервиса:

- запись студентов на курсы;
- отслеживание прогресса студентов (пройденные уроки, выполненные задания);
- выдача сертификатов по завершении курса.

Основные конечные точки API: POST /api/enrollments – запись на курс; GET /api/enrollments/:user_id – получение списка курсов пользователя; PUT /api/enrollments/:enrollment_id/progress – обновление прогресса; POST /api/certificates – выдача сертификата.

На рисунке 18 представлена диаграмма классов для сервиса обучения.

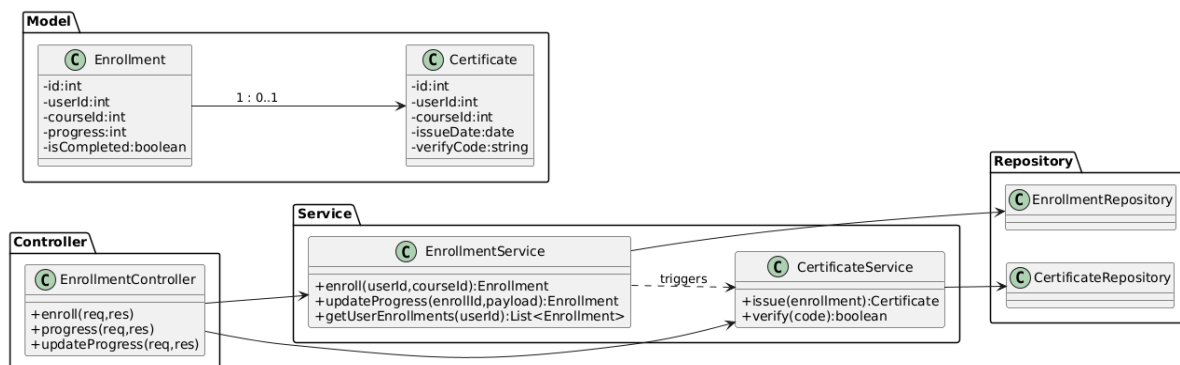


Рисунок 18 – Диаграмма классов для сервиса обучения

Сервис взаимодействует с сервисом пользователей для проверки ролей (например, только студент может записаться на курс) и с сервисом курсов для получения информации о курсах.

2.4.4 Проектирование сервиса администрирования

Сервис администрирования (Admin Service) отвечает за управление платформой, включая категории курсов, статистику и модерацию контента. Сервис реализован как отдельное Express-приложение, взаимодействующее с базой данных через Sequelize.

Основные функции сервиса:

- управление категориями курсов (создание, редактирование, удаление);
- просмотр статистики платформы (количество пользователей, курсов, завершенных курсов);
- модерация контента (проверка курсов, отзывов, профилей пользователей).

Основные конечные точки API: GET /api/statistics – получение статистики платформы; POST /api/categories – создание категории курсов; PUT /api/categories/:category_id – редактирование категории; GET /api/moderation/content – получение контента для модерации.

На рисунке 19 представлена диаграмма классов для сервиса администрирования.

На диаграмме отражено разделение логики между уровнями: контроллеры, отвечающие за обработку входящих запросов, обращаются к соответствующим сервисам, а те, в свою очередь, взаимодействуют с репозиториями для получения данных из базы.

Диаграмма классов позволяет разработчикам и архитекторам системы наглядно представить внутреннюю структуру кода, определить ключевые компоненты и их ответственность, а также выявить устойчивые зависимости и потенциальные узкие места. Это упрощает процесс реализации, тестирования и

сопровождения программного обеспечения, особенно в условиях командной разработки и масштабирования проекта в будущем.

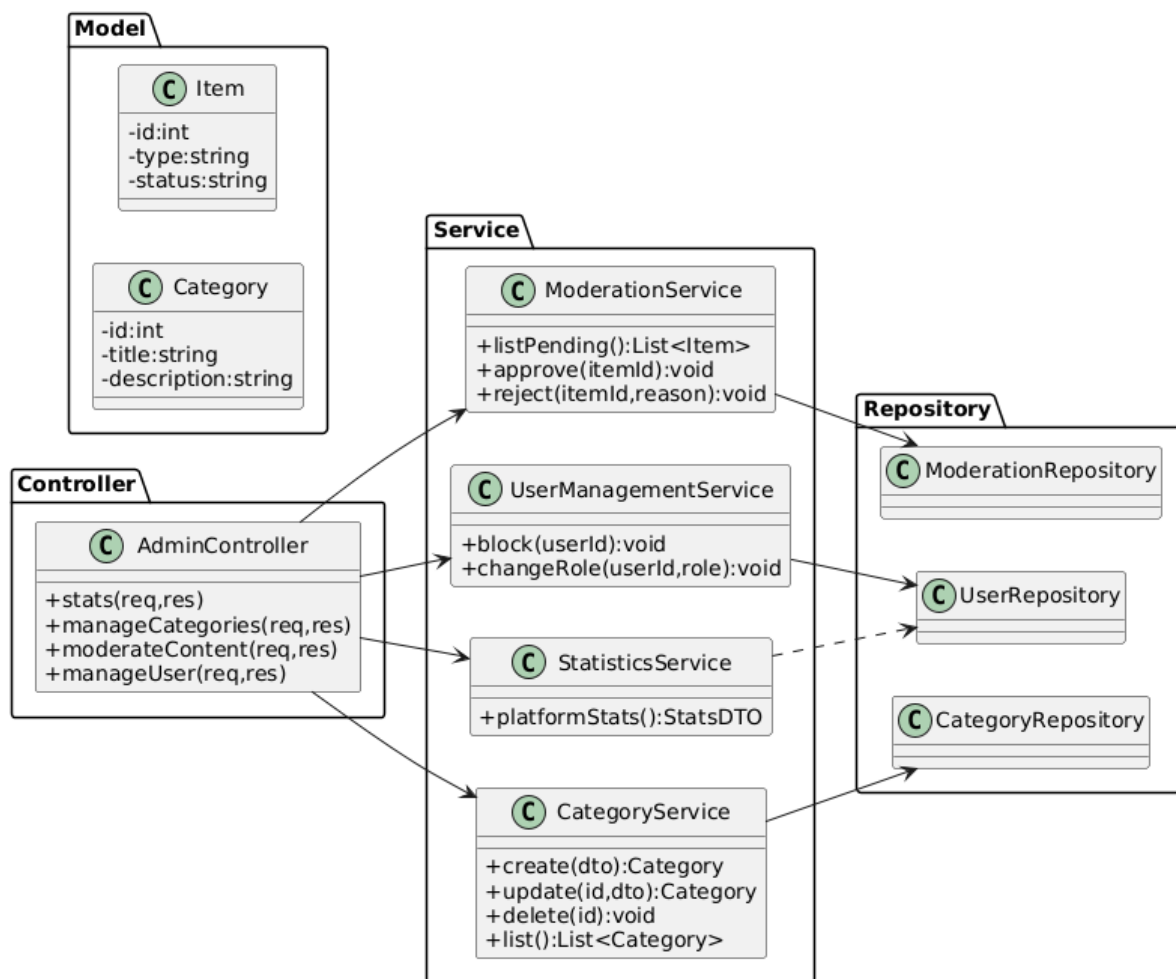


Рисунок 19 – Диаграмма классов для сервиса администрирования

Сервис взаимодействует с сервисом пользователей для проверки роли администратора.

2.4.5 Проектирование сервиса сообщений

Сервис сообщений (Messaging Service) отвечает за общение между пользователями. Сервис реализован как отдельное Express-приложение, взаимодействующее с базой данных через Sequelize.

Основные функции сервиса: отправка сообщений между пользователями, просмотр списка сообщений пользователя.

Основные конечные точки API: POST /api/messages – отправка сообщения; GET /api/messages/:user_id – получение списка сообщений пользователя.

На рисунке 20 представлена диаграмма классов для сервиса сообщений.

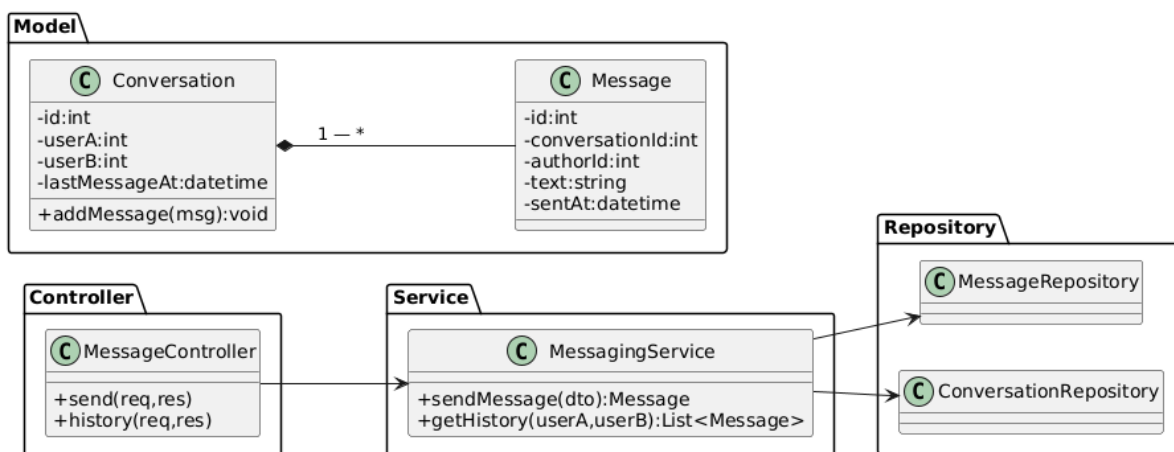


Рисунок 20 – Диаграмма классов для сервиса сообщений

Сервис взаимодействует с сервисом пользователей для проверки ролей и получения информации о пользователях.

2.5 Проектирование алгоритмов

В данном разделе представлены три ключевых алгоритма, обеспечивающие базовую бизнес-логику образовательного портала. Для каждого алгоритма приведено текстовое описание и диаграмма деятельности.

2.5.1 Алгоритм записи студента на курс

Алгоритм описывает последовательность действий, выполняемых системой при попытке студента записаться на выбранный курс. После успешной записи система возвращает на клиентской части подтверждение, и пользователь видит соответствующее сообщение, а курс появляется в его личном кабинете

Работа алгоритма проходит в несколько этапов:

- 1) Студент инициирует запрос «Записаться» на странице курса;
- 2) Сервис аутентификации проверяет JWT-токен пользователя;
- 3) Система обращается к сервису обучения (Enrollment Service) для проверки: не записан ли студент на этот курс ранее; доступен ли курс для записи (статус «Открыт», есть свободные места);
- 4) При успешной проверке в таблицу enrollments добавляется новая запись, фиксирующая user_id, course_id и enrolled_at;
- 5) Система отправляет уведомление студенту (e-mail + push) о успешной записи;
- 6) В ответ на front-end возвращается JSON с кодом успеха.

На рисунке 21 представлена диаграмма деятельности данного алгоритма.

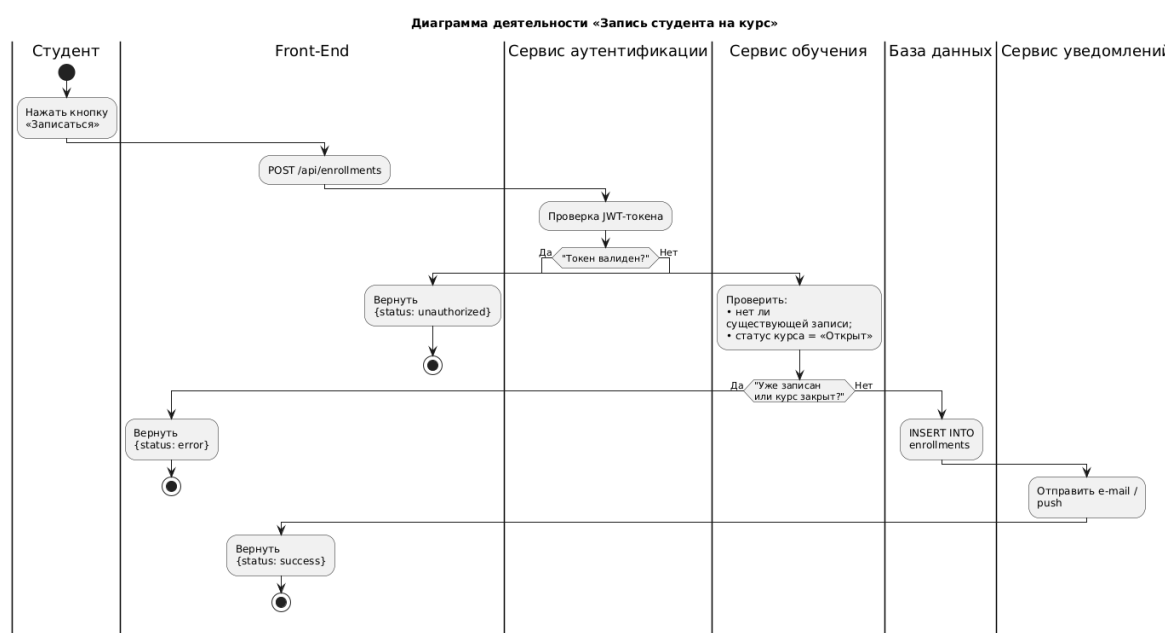


Рисунок 21 – Диаграмма деятельности алгоритма записи студента на курс

Алгоритм фиксирует порядок операций, обеспечивая обработку запроса на запись, проверку необходимых условий и добавление студента на курс.

2.5.2 Алгоритм прохождения урока и обновления прогресса

Данный алгоритм демонстрирует, как система отслеживает прохождение студентом урока и обновляет показатели прогресса.

- 1) Студент открывает урок; front-end запрашивает контент урока;
- 2) После изучения материала студент нажимает «Завершить урок»;
- 3) Сервис обучения проверяет наличие непреданных заданий и тестов: если есть незавершённые задания – система предлагает сдать; если тест завершён с проходным баллом – переход к п. 4;
- 4) В таблице enrollments поле progress увеличивается на вес урока (например, +5 %);
- 5) Если progress = 100 %, запускается процедура выдачи сертификата;
- 6) Студент получает уведомление о завершении урока и прогрессе.

Данный алгоритм демонстрирует, как система отслеживает прохождение студентом урока и обновляет показатели прогресса. Когда студент открывает урок, клиентская часть направляет запрос на сервер с целью получения контента, включающего текстовые и мультимедийные материалы, задания и тесты. После изучения материала студент нажимает кнопку «Завершить урок», тем самым инициируя процесс проверки состояния прохождения. Сервис обучения анализирует наличие незавершённых заданий и тестов. Если такие элементы не завершены, система предлагает студенту их сдать.

Таким образом, алгоритм обеспечивает полноту и точность фиксации результатов обучения, снижает вероятность случайного пропуска важных учебных элементов и способствует формированию структурированной образовательной траектории. Автоматизация процесса прогрессирования и выдачи сертификатов повышает мотивацию студентов и упрощает контроль за успеваемостью со стороны администратора или преподавателя.

На рисунке 22 представлена диаграмма деятельности данного алгоритма.

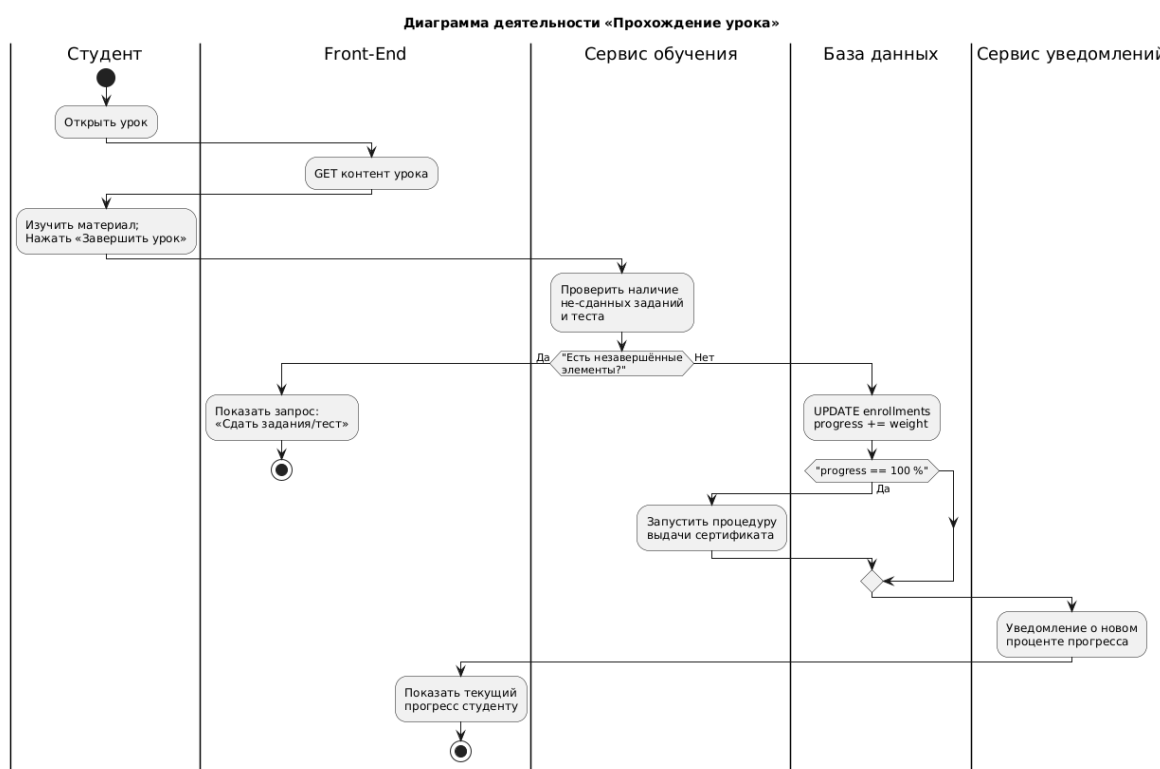


Рисунок 22 – Диаграмма деятельности алгоритма прохождения урока

Алгоритм обеспечивает последовательное отслеживание активности студента, позволяя системе оперативно обновлять данные о прогрессе и обеспечивать актуальность информации в процессе обучения.

2.5.6 Алгоритм рекомендаций курсов

Алгоритм предназначен для формирования персонального списка курсов на основании освоенных навыков и карьерных предпочтений студента.

При активации функции рекомендаций система инициирует сбор данных о текущем профиле пользователя. В первую очередь анализируются уже освоенные навыки, зафиксированные в базе данных, а также история пройденных курсов, полученные оценки и результаты тестирования. Дополнительно учитываются личные предпочтения студента, указанные в его

профиле – такие как интересующие профессиональные направления, желаемая специализация, целевая должность или уровень квалификации.

Пошаговая реализация алгоритма:

- 1) Студент открывает раздел «Рекомендации»;
- 2) Front-end отправляет запрос GET /api/recommendations с user_id;
- 3) Recommendation Engine выполняет: выборку навыков студента из student_skills; анализ целей обучения (поле «interests» профиля); фильтрацию активных курсов по релевантным навыкам и уровню сложности;
- 4) Для каждого подходящего курса вычисляется рейтинг score;
- 5) Сервис сортирует курсы по score и возвращает Top-N позиций;
- 6) Front-end выводит персонализированный список с кнопками «Подробнее» / «Записаться».

Система персональных рекомендаций способствует не только индивидуализации обучения, но и повышению мотивации студентов за счёт осознанного выбора курсов, приближающих их к профессиональным целям.

В основе алгоритма лежит анализ профиля студента, включающий перечень уже пройденных курсов, полученные оценки, результаты тестирования, а также указанные интересы и целевые карьерные направления.

В перспективе возможно использование машинного обучения для более точного прогнозирования успеха прохождения того или иного курса на основании похожих траекторий других студентов. Таким образом, система может не только подсказывать полезные курсы, но и предсказывать вероятность их успешного завершения, повышая релевантность рекомендаций.

В дальнейшем алгоритм может адаптироваться с учётом новых действий пользователя: после прохождения очередного курса или изменения карьерных предпочтений.

На рисунке 23 представлена диаграмма деятельности данного алгоритма.

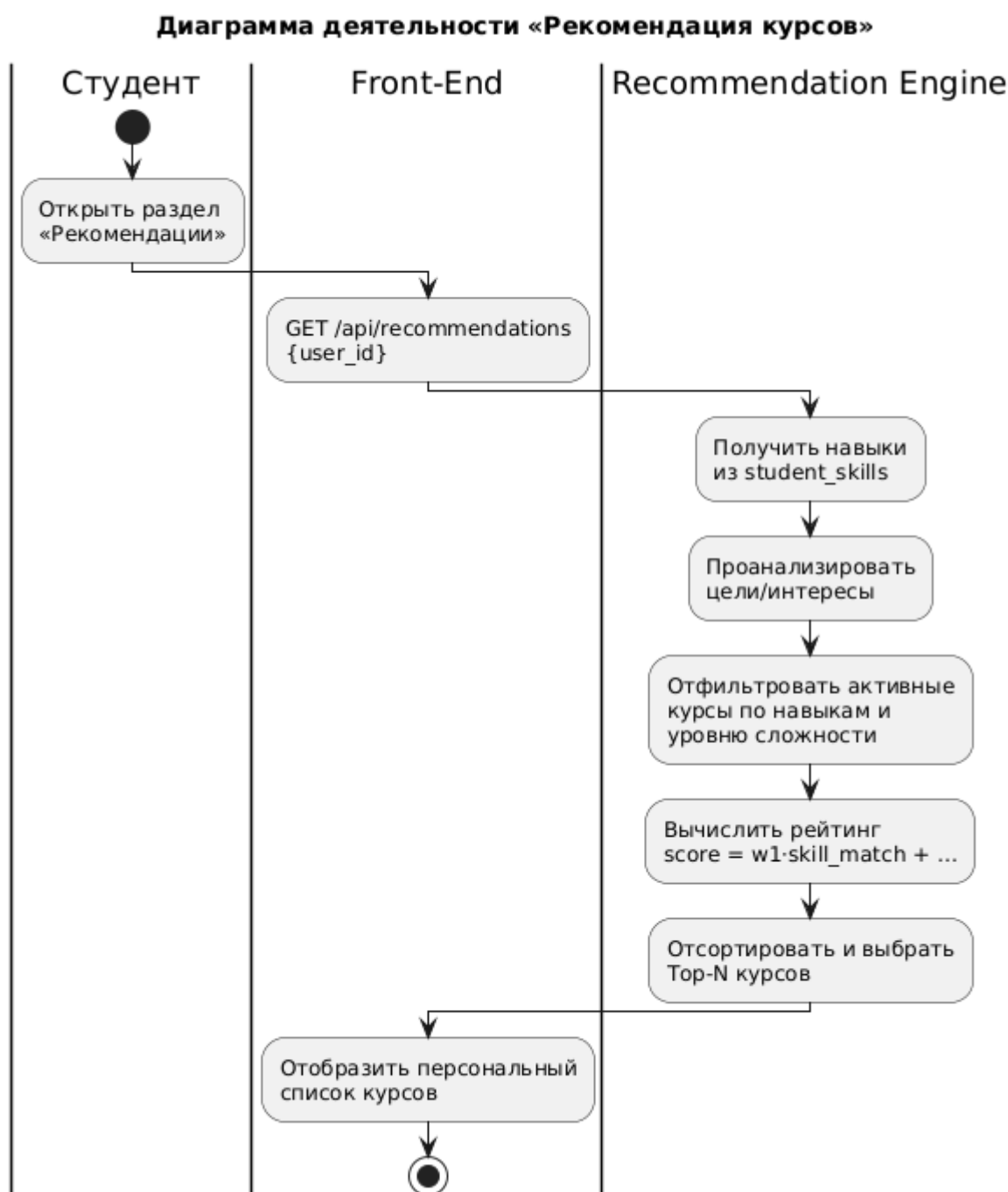


Рисунок 23 – Диаграмма деятельности алгоритма рекомендации курсов

Алгоритм обеспечивает индивидуализированный подход к обучению, формируя список курсов с учетом уже приобретённых навыков и

профессиональных интересов студента, что способствует более целенаправленному развитию.

3 Реализация образовательного портала

3.1 Реализация пользовательского функционала для роли студента

В рамках реализации функциональности для роли «Студент» предусмотрены следующие возможности:

- регистрация и авторизация с присвоением роли;
- просмотр доступных курсов (каталог), фильтрация и подробная информация по курсу;
- запись на курс (листинг 1) и получение доступа к его материалам;
- прохождение курса, отправка итоговой работы;
- получение сертификата в случае успешного окончания курса;
- просмотр вакансий и отклик на них;
- просмотр персональных карьерных траекторий.

Для обеспечения перечисленных возможностей используется модуль REST API, в частности маршруты: GET /courses, POST /enrollments, POST /enrollments/:id/submit, GET /certificates, GET /students/:id/recommendations.

Листинг 1 – Алгоритм записи студента на курс

```
exports.enroll = async (req, res) => {
  const userId = req.user.id;
  const { course_id } = req.body;

  const result = await db.query(
    `INSERT INTO enrollments (user_id, course_id)
     VALUES ($1, $2)
     RETURNING *`, [userId, course_id]
  );

  res.status(201).json(result.rows[0]);
};
```

Диаграмма последовательности для алгоритма записи студента на курс представлена на рисунке 24.

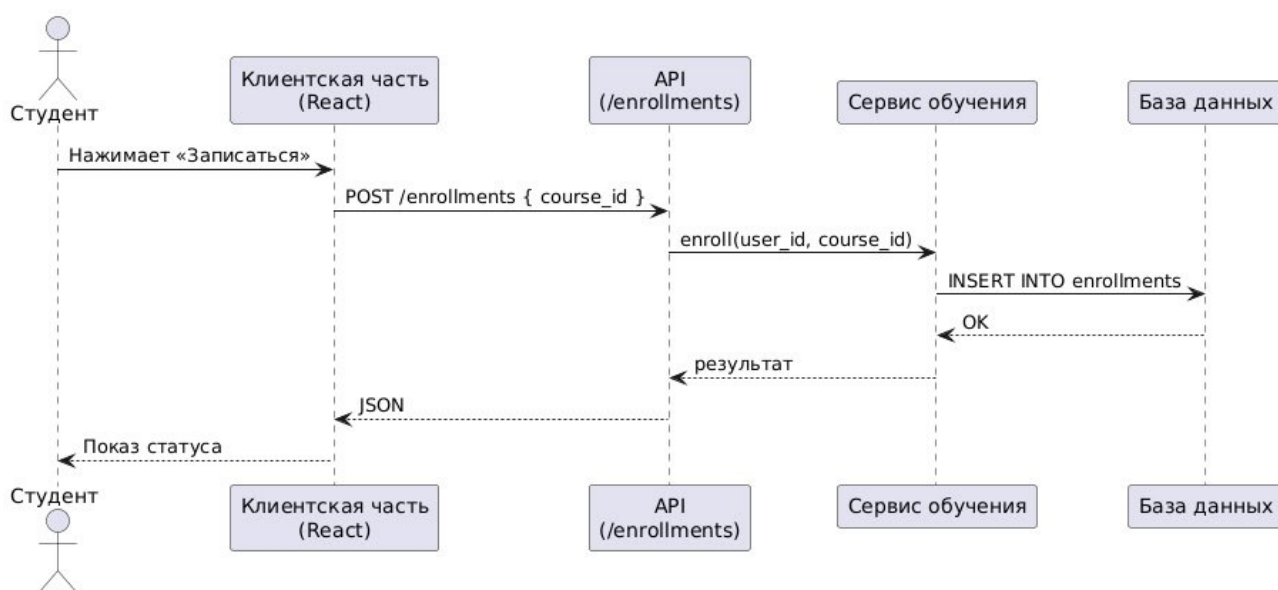


Рисунок 24 – Диаграмма последовательности для алгоритма записи студента на курс

Макет страницы курса приведен на рисунке 25.

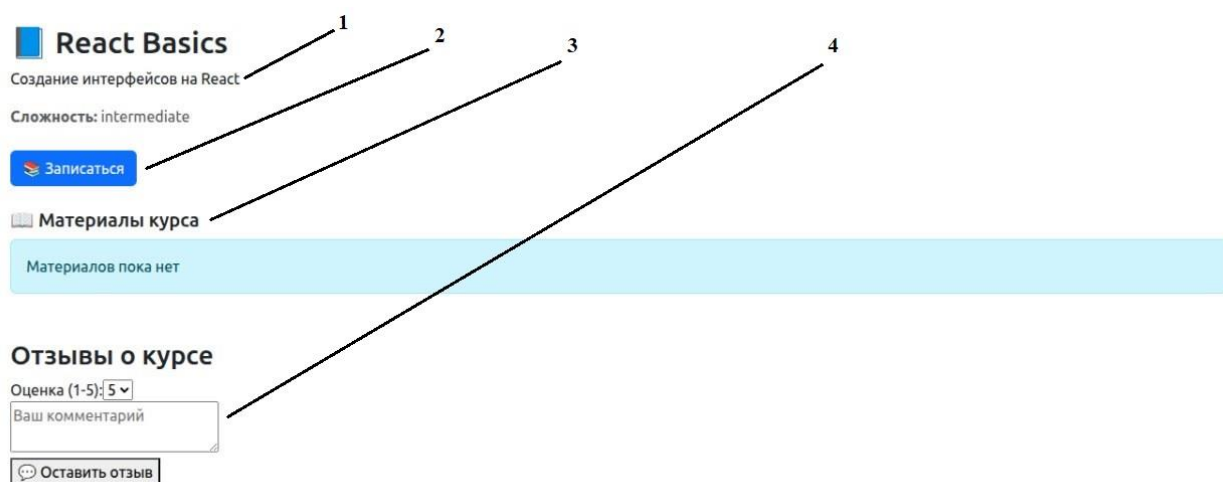


Рисунок 25 – Макет страницы курса

Рассмотрим основные элементы макета страницы курса разработанного приложения, представленные на рисунке 25, описывающая 4 элемента интерфейса:

- 1 – описание курса;
- 2 – кнопка записи;
- 3 – панель материалов курса;
- 4 – панель отзывов.

Роль пользователя определяется автоматически на серверной части посредством связанной таблицы students в базе данных, связанной с users по идентификатору.

3.2 Реализация пользовательского функционала для роли преподавателя

Роль «Преподаватель» обладает доступом к функциям управления образовательными материалами. В частности, реализованы следующие возможности:

- создание и редактирование курсов (листинг 2);
- загрузка материалов для курса;
- просмотр списка студентов, записанных на курс;
- приём и проверка итоговых работ;
- одобрение завершения курса.

Листинг 2 – Алгоритм создания курса преподавателем

```
const createCourse = async ({ title, description, teacher_id }) =>
{
  const result = await db.query(
    `INSERT INTO courses (title, description, teacher_id)
    VALUES ($1, $2, $3) RETURNING *`,
    [title, description, teacher_id]
  );
  return result.rows[0];
};
```

Диаграмма последовательности для алгоритма создания курса преподавателем представлена на рисунке 26.

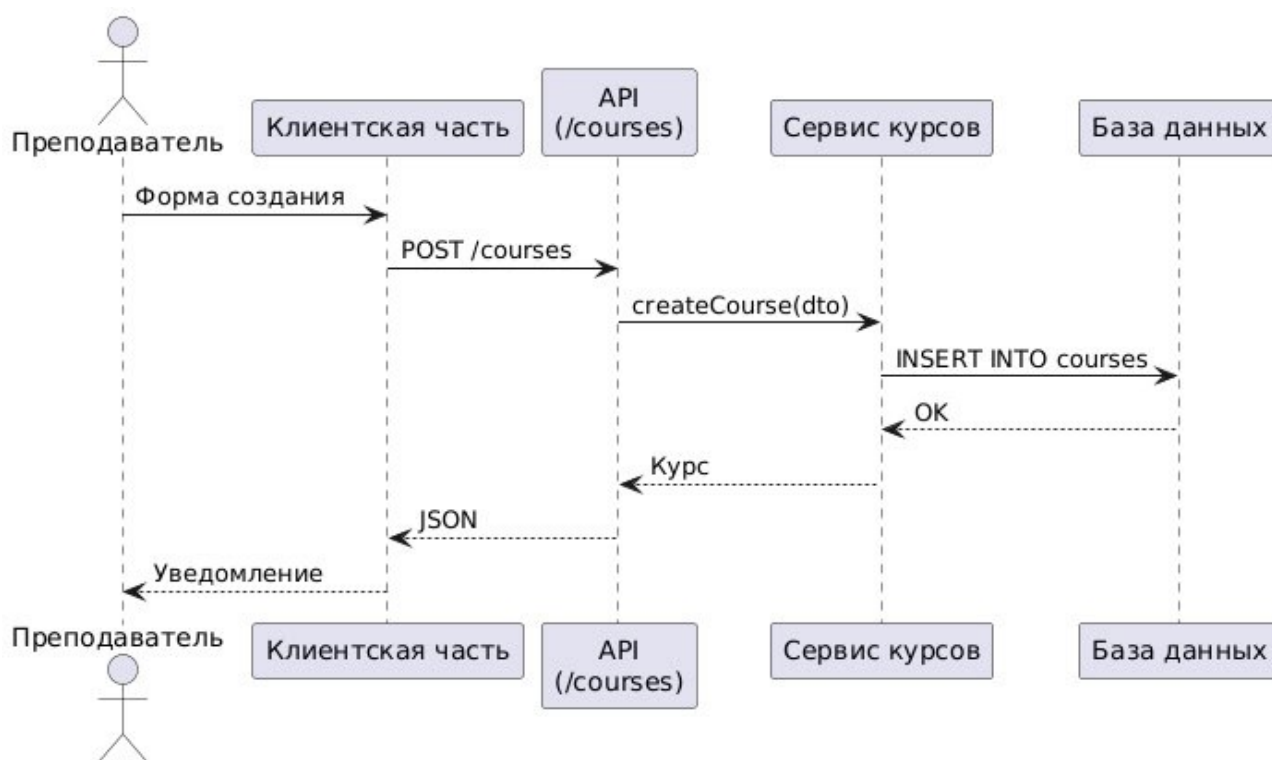


Рисунок 26 – Диаграмма последовательности для алгоритма создания курса преподавателем

Макет страницы создания курса приведен на рисунке 27.

Создание нового курса

Название курса:

Описание:

Уровень сложности: Начальный

Рисунок 27 – Макет страницы создания курса

Рассмотрим основные элементы макета страницы создания курса разработанного приложения, представленные на рисунке 27:

- 1 – поле названия нового курса;
- 2 – поле описания нового курса;
- 3 – панель выбора уровня сложности создаваемого курса;
- 4 – кнопка создания курса.

Доступ к действиям преподавателя устанавливается с помощью промежуточного слоя авторизации. Проверка роли происходит через запрос к таблице teachers.

3.4 Реализация пользовательского функционала для роли работодателя

Функциональность работодателя включает инструментарий, позволяющий публиковать карьерные предложения, а также создавать учебные курсы от лица компании (листинг 3). Реализованы следующие функции:

- создание карьерных траекторий;
- публикация вакансий;
- установка требований к навыкам;
- просмотр откликов студентов и резюме.

Листинг 3 – Алгоритм создания вакансии работодателем

```
exports.createVacancy = async (req, res) => {
  const { title, description, salary_range } = req.body;
  const employerId = req.user.id;
  await db.query(`
    INSERT INTO vacancies (employer_id, title, description,
salary_range)
    VALUES ($1, $2, $3, $4)\`,
    [employerId, title, description, salary_range]
  `);

  res.status(201).json({ message: "Вакансия создана" });
};
```

Диаграмма последовательности для алгоритма создания вакансии работодателем представлена на рисунке 28.

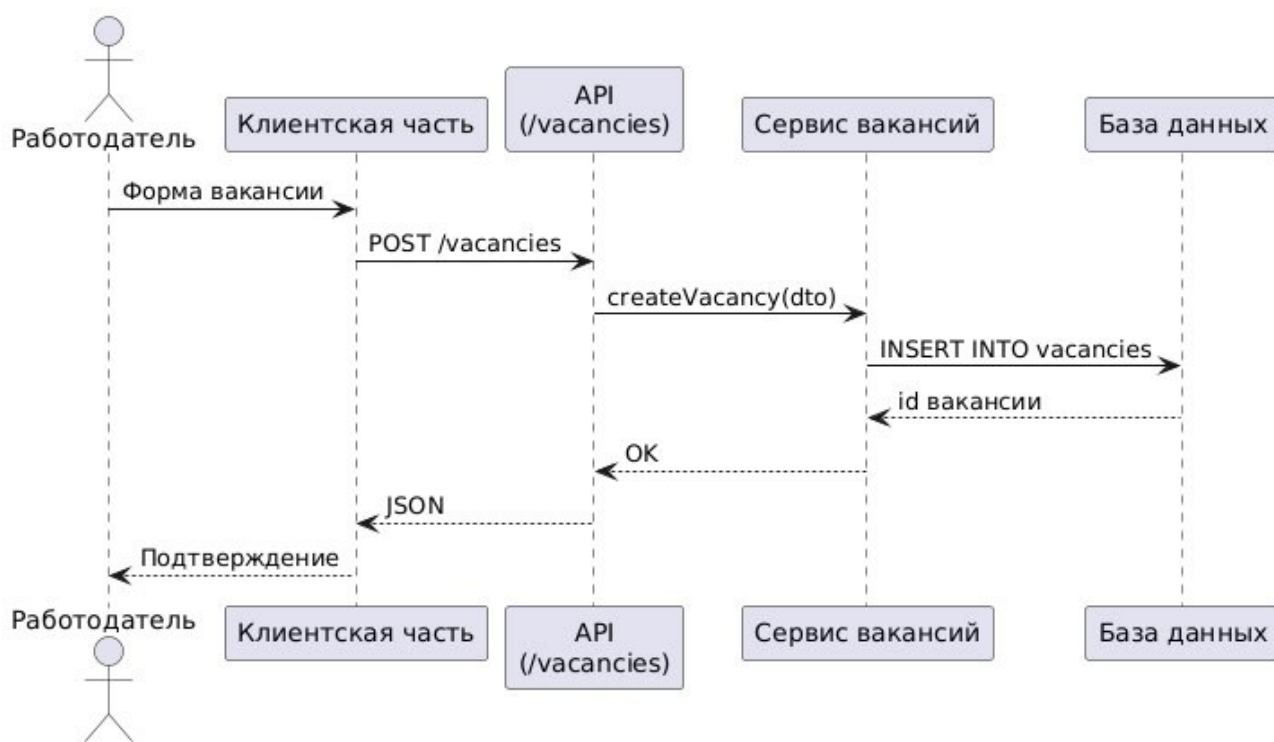


Рисунок 28 – Диаграмма последовательности для алгоритма создания вакансии работодателем

Макет страницы создания вакансии приведен на рисунке 29.

Он отражает структуру пользовательского интерфейса, предназначенного для размещения работодателями новых вакансий на платформе. Данный макет разработан с учётом удобства ввода информации и соблюдения пользовательских сценариев, обеспечивающих простоту и логичность взаимодействия с системой.

Основное внимание в макете уделено полям, необходимым для детального описания вакансии.

Макет страницы создания вакансии, представленный на рисунке 29, отражает продуманный и удобный интерфейс, способствующий быстрому и точному размещению предложений о работе, видимые работодателю на странице личного кабинета.

Создание вакансии

1 Название должности:

2 Описание работы:

3 Уровень зарплаты:
например: 80k-120k

4 Тип занятости:
Полная занятость

☐ Удалённая работа возможна

Локация / город:

Необходимые навыки:
Django
Git
HTML & CSS
Python
Python Basics

Удерживайте Ctrl (или Cmd) для выбора нескольких

5 Создать вакансию

6

7

Рисунок 29 – Макет страницы создания вакансии

Рассмотрим основные элементы макета страницы создания вакансии разработанного приложения, представленные на рисунке 29:

- 1 – поле названия должности;
- 2 – поле описания вакансии;
- 3 – поле ввода заработной платы вакансии;
- 4 – панель выбора занятости;
- 5 – кнопка создания вакансии;
- 6 – панель выбора навыков;
- 7 – поле ввода адреса.

Роль «Работодатель» реализована через таблицу `employers`, обеспечивающую хранение информации о компании, включая описание, проверку и логотип.

3.5 Реализация пользовательского функционала для роли администратора

Роль администратора предоставляет расширенные права для управления платформой, включая:

- просмотр общей статистики;
- управление статусом пользователей: блокировка, разблокировка (листинг 4);
- изменение ролей пользователей;
- модерацию курсов и контента;
- просмотр отчётной информации.

Функциональность реализована в административной панели с соответствующими маршрутами: GET /admin/stats, GET /admin/users.

Листинг 4 – Алгоритм блокировки и разблокировки пользователя

```
exports.toggleBlock = async (userId) => {
  const result = await db.query(`
    UPDATE users
    SET is_blocked = NOT is_blocked
    WHERE id = $1
    RETURNING id, is_blocked
  `, [userId]);

  return result.rows[0];
};
```

Диаграмма последовательности для алгоритма блокировки и разблокировки пользователя представлена на рисунке 30.

Такая диаграмма служит важным элементом проектной документации, позволяя визуализировать логику выполнения бизнес-процесса во времени и установить чёткую последовательность вызовов между объектами.

Диаграмма последовательности позволяет чётко понять взаимодействие между слоями системы при блокировке и разблокировке пользователя и способствует эффективной реализации и отладке данного функционала.

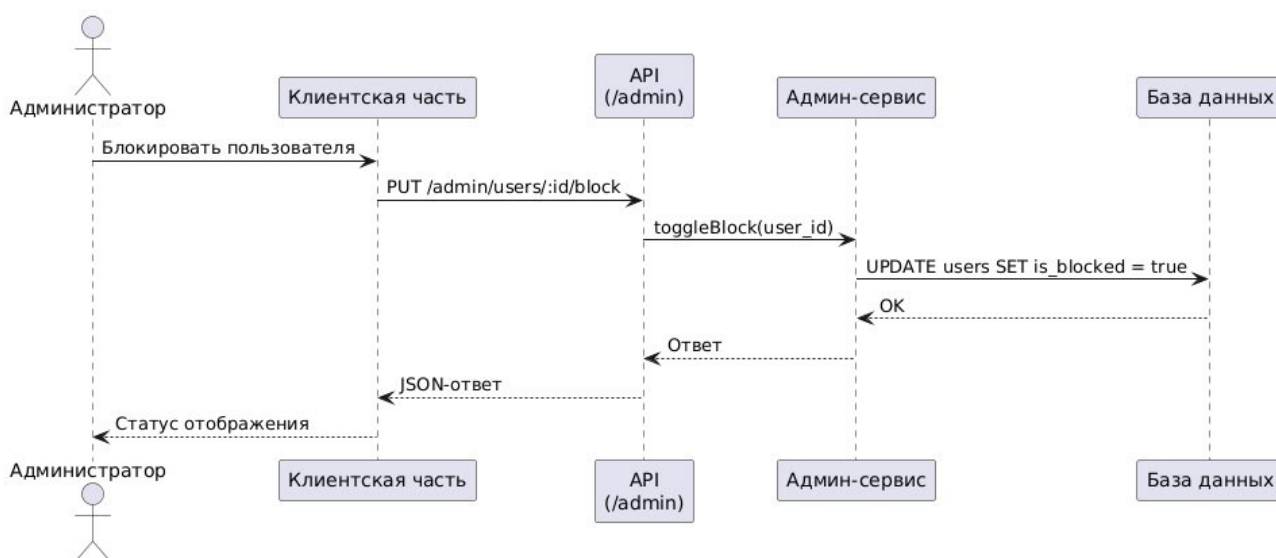


Рисунок 30 – Диаграмма последовательности для алгоритма блокировки и разблокировки пользователя

Такая диаграмма служит важным элементом проектной документации, позволяя визуализировать логику выполнения бизнес-процесса во времени и установить чёткую последовательность вызовов между объектами. Диаграмма последовательности позволяет чётко понять взаимодействие между слоями системы при блокировке и разблокировке пользователя и способствует эффективной реализации и отладке данного функционала.

Диаграмма последовательности позволяет чётко понять взаимодействие между слоями системы при блокировке и разблокировке пользователя и способствует эффективной реализации и отладке данного функционала.

Такая наглядность значительно облегчает коммуникацию между разработчиками, тестировщиками и аналитиками, а также способствует снижению количества ошибок в ходе реализации процесса.

Далее другие алгоритмы будут рассмотрены с примерами соответствующих диаграмм

Макет страницы администратора приведен на рисунке 31, с объяснением элементов.

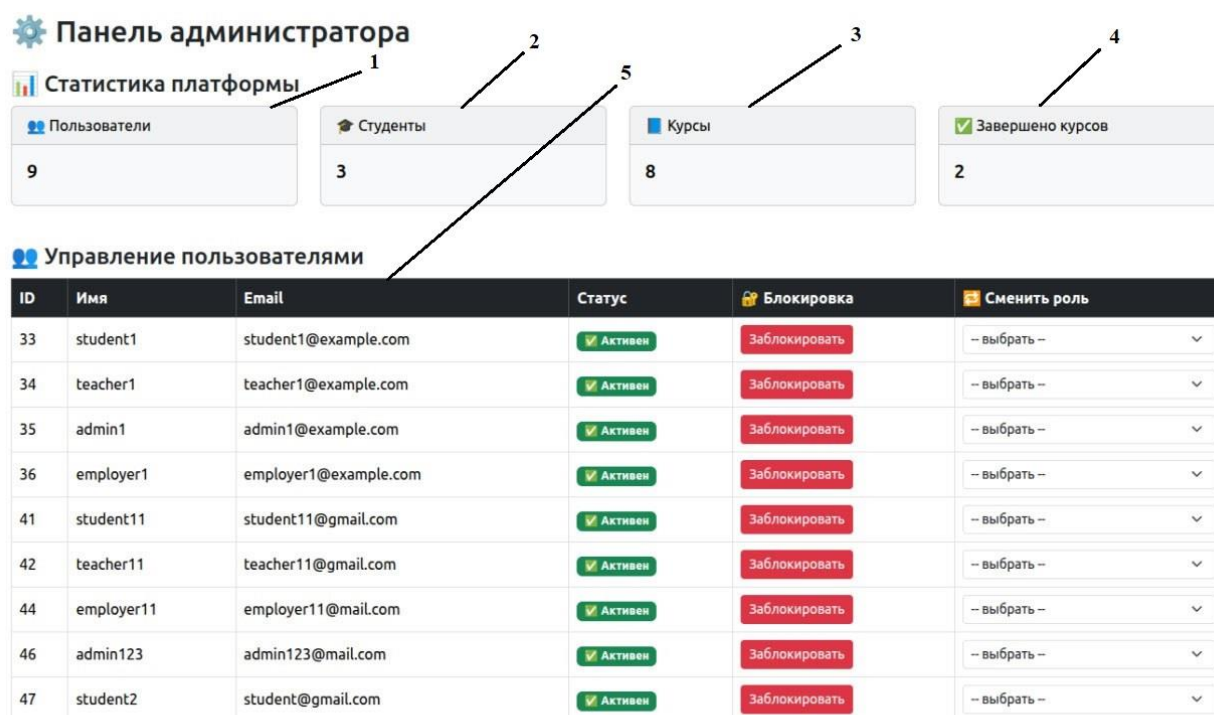


Рисунок 31 – Макет страницы администратора

Рассмотрим основные элементы макета страницы администратора разработанного приложения, представленные на рисунке 31:

- 1 – панель количества зарегистрированных пользователей;
- 2 – панель количества студентов;
- 3 – панель общего количества курсов;
- 4 – панель количества завершенных курсов;
- 5 – панель управления пользователями.

Доступ осуществляется только при наличии записи в связанной таблице admins.

3.6 Реализация взаимодействия с базой данных

Взаимодействие серверной части с базой данных реализовано посредством библиотеки pg для PostgreSQL. Все запросы к базе данных построены с

использованием параметризованных SQL-инструкций, что обеспечивает защиту от SQL-инъекций.

Организация БД реализована на основе заранее спроектированной физической модели, приведённой во второй главе пояснительной записки. Используются связи вида «один-к-одному», «один-ко-многим» и «многие-ко-многим», реализованные через внешние ключи и вспомогательные таблицы. На листинге 5 приведен алгоритм получения пользователей по email.

Листинг 5 – Алгоритм получения пользователя по email

```
const getUserByEmail = async (email) => {
  const result = await db.query(
    SELECT * FROM users WHERE email = $1,
    [email]
  );
  return result.rows[0];
};
```

Диаграмма последовательности для алгоритма взаимодействия сервисов с БД представлена на рисунке 32.



Рисунок 32 – Диаграмма последовательности для алгоритма взаимодействия сервисов с БД

Макет страницы вакансий приведен на рисунке 33.

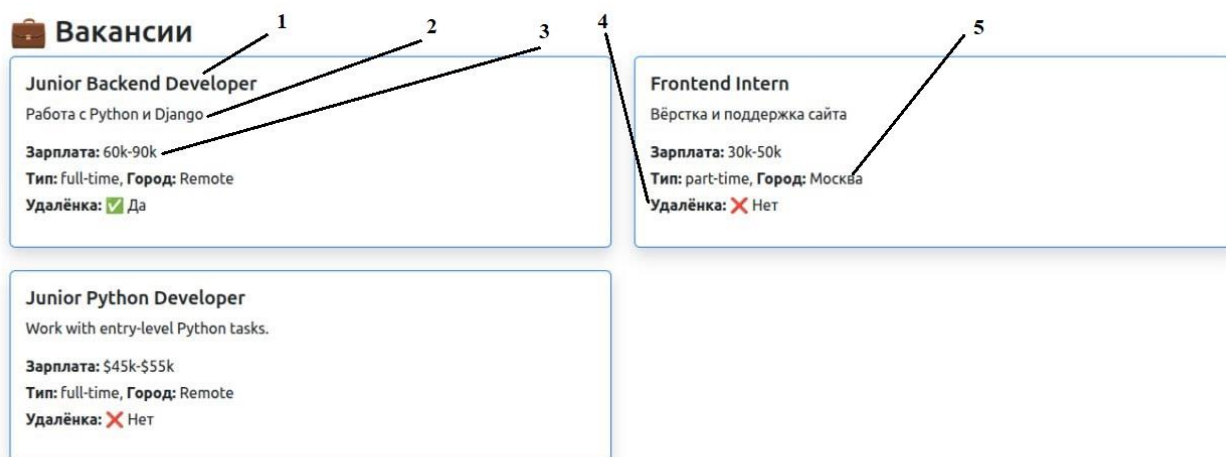


Рисунок 33 – Макет страницы вакансий

Рассмотрим основные элементы макета страницы вакансий разработанного приложения, представленные на рисунке 33:

- 1 – название вакансии;
- 2 – описание вакансии;
- 3 – уровень заработной платы;
- 4 – возможность удаленной работы;
- 5 – адрес работы.

Сущности пользователя определены через таблицу users, подразделяемую на роли с помощью таблиц students, teachers, employers, admins.

3.7 Реализация клиентской (пользовательской) части

Клиентская часть реализована в виде одностраничного приложения (SPA) на JavaScript-фреймворке React. Визуальное оформление осуществляется с использованием библиотеки Bootstrap 5, а взаимодействие с сервером – с помощью библиотеки axios, что обеспечивает корректную и быструю работу между серверной и клиентской частями.

Структура приложения логически разделена на компоненты:

- App.js – конфигурация маршрутов;
- Navbar.js – горизонтальное меню;
- PrivateRoute.js – защищённые маршруты;
- страницы личного кабинета (DashboardPage.js), курсов, вакансий и администрирования.

Аутентификация пользователей реализована с помощью JWT-токенов, сохраняемых в localStorage. При каждом запросе токен передаётся в HTTP-заголовке Authorization.

Аутентификация пользователей реализована с помощью JWT (JSON Web Token). После успешного входа пользователя токен сохраняется в localStorage, откуда извлекается и подставляется в заголовок Authorization каждого HTTP-запроса, отправляемого через axios. Таким образом обеспечивается бесшовная авторизация при взаимодействии с серверной частью API и сохраняется сессия пользователя даже при перезагрузке страницы.

Для повышения безопасности также реализована проверка срока действия токена. В случае его истечения пользователь автоматически перенаправляется на страницу авторизации. Определённая логика отображения интерфейса основана на ролях пользователей (например, студент, преподаватель, администратор), что реализовано как на уровне маршрутов, так и в самих компонентах интерфейса. На листинге 6 приведен алгоритм загрузки списка курсов.

Листинг 6 – Алгоритм загрузки списка курсов

```
useEffect(() => {
  API.get('/courses').then(res => {
    setCourses(res.data);
  }).catch(err => {
    console.error("Ошибка получения курсов:", err);
  });
}, []);
```

Диаграмма последовательности для алгоритма загрузки списка курсов представлена на рисунке 34.



Рисунок 34 – Диаграмма последовательности для алгоритма загрузки списка курсов

Макет страницы каталога курсов приведен на рисунке 35.

Макет служит визуальным представлением пользовательского интерфейса, предназначенного для просмотра и выбора образовательных курсов, доступных на платформе. Данный макет разработан с учётом требований удобства поиска, фильтрации и навигации, обеспечивая пользователю интуитивно понятный и комфортный доступ к учебному контенту.

Макет страницы каталога курсов отражает продуманный пользовательский интерфейс, ориентированный на широкий круг обучающихся.

Таким образом, макет каталога курсов предоставляет пользователю все необходимые инструменты для эффективного взаимодействия с образовательным контентом, способствует повышению вовлечённости и облегчает построение индивидуальной учебной траектории.

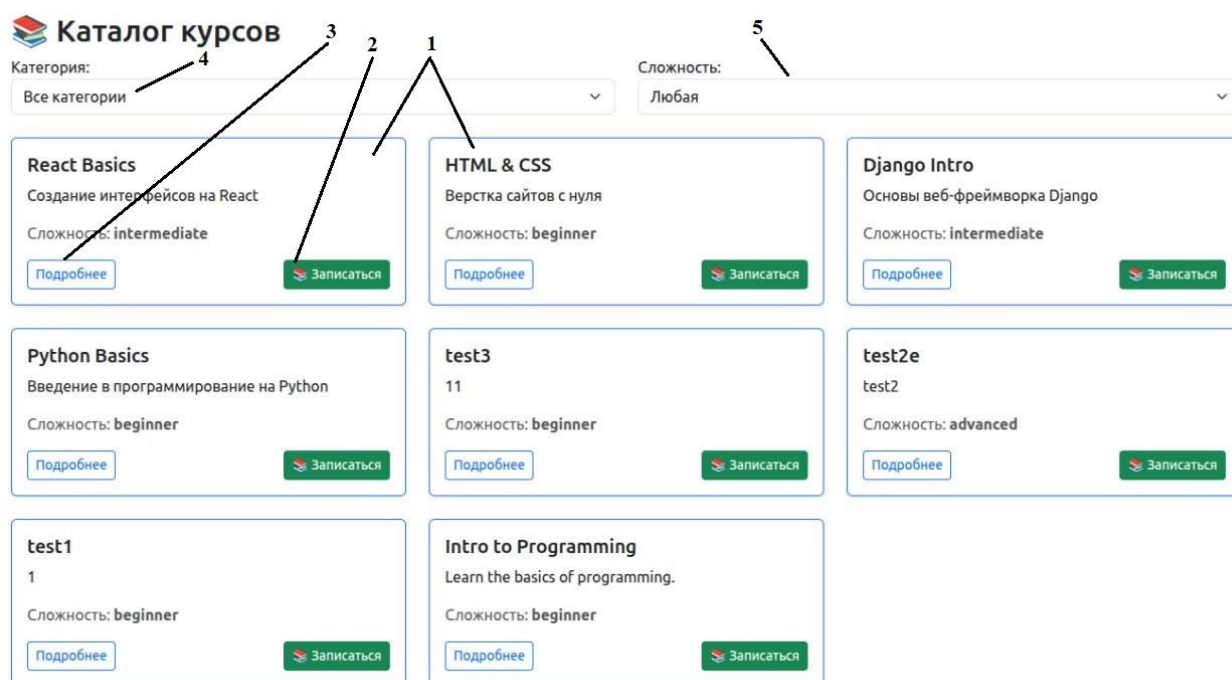


Рисунок 35 – Макет страницы каталога курсов

Рассмотрим основные элементы макета страницы каталога курсов разработанного приложения, представленные на рисунке 35:

- 1 – панель отдельного курса;
- 2 – кнопка записи;
- 3 – кнопка перехода на детальную страницу курса;
- 4 – панель выбора категории курсов;
- 5 – панель выбора сложности курсов.

Формы регистрации, входа в систему, создания курса и подачи отклика оформлены в соответствии с современными принципами UX/UI.

3.8 Реализация модуля подбора вакансий и учебных курсов

Подбор вакансий и карьерных рекомендаций реализован на основе сопоставления навыков, освоенных студентом, и требований, указанных работодателем (листинг 7).

Алгоритм реализуется с помощью SQL-запросов, подсчитывающих количество совпадений между таблицами `student_skills` и `vacancy_skills`, а также `trajectory_courses` и `enrollments`.

Листинг 7 – Алгоритм получения рекомендованных вакансий для студента

```
const result = await db.query(`
  SELECT v.*
  FROM vacancies v
  JOIN vacancy_skills vs ON v.id = vs.vacancy_id
  JOIN student_skills ss ON ss.skill_id = vs.skill_id
  WHERE ss.student_id = $1
`, [studentId]);
```

Диаграмма последовательности для алгоритма получения рекомендованных вакансий для студента представлена на рисунке 36.

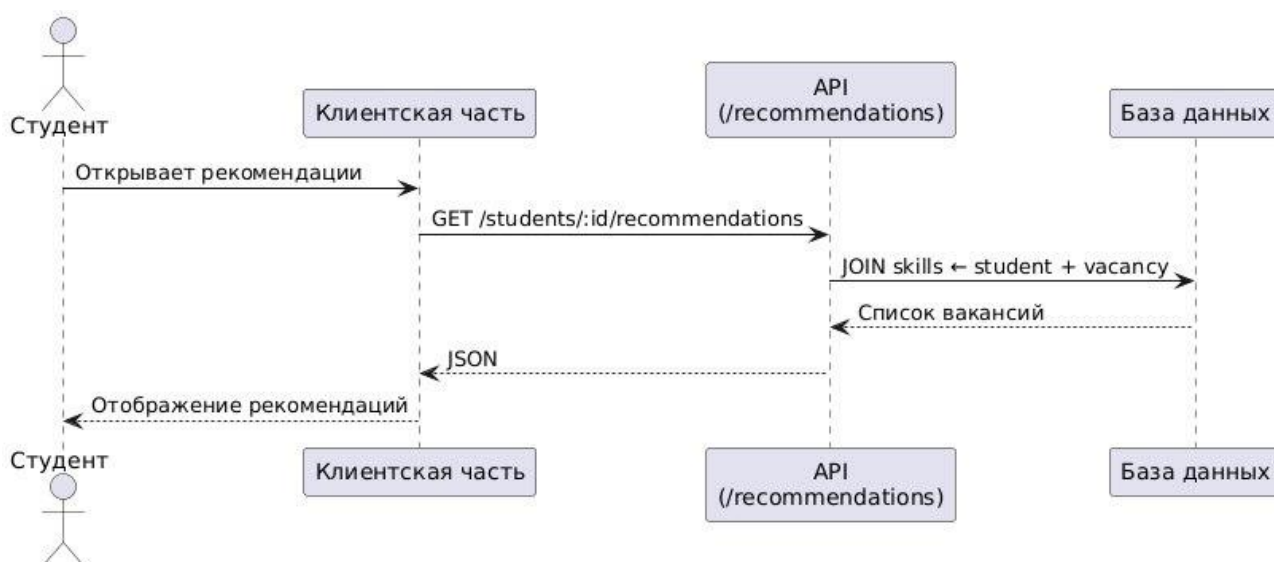


Рисунок 36 – Диаграмма последовательности для алгоритма получения рекомендованных вакансий для студента

Макет страницы рекомендаций приведен на рисунке 37.

Макет страницы рекомендаций, представленный на рисунке 37, демонстрирует удобный и функциональный инструмент персонализации обучения.

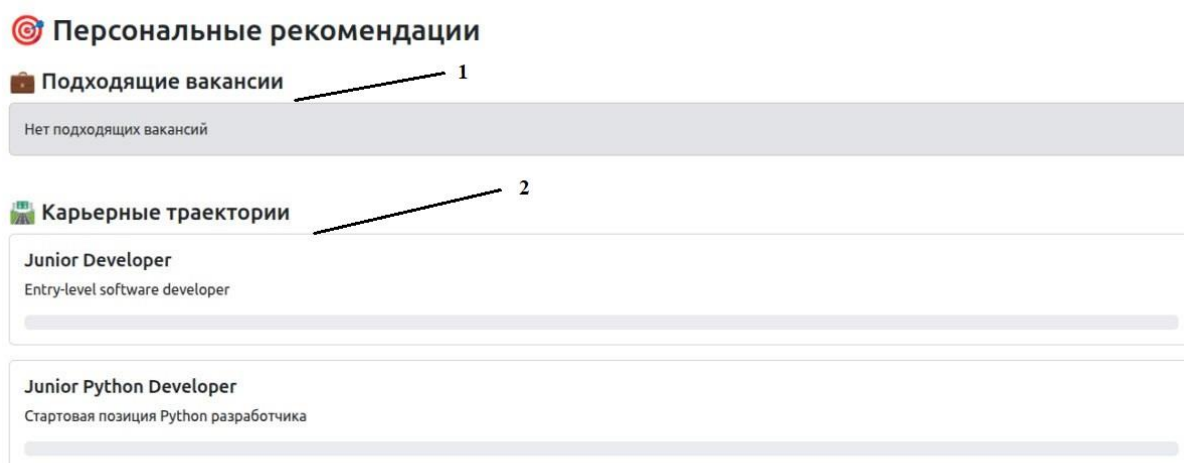


Рисунок 37 – Макет страницы рекомендаций

Рассмотрим основные элементы макета страницы рекомендаций разработанного приложения, представленные на рисунке 37:

- 1 – панель доступных вакансий;
- 2 – панель карьерных траекторий.

На основе этих данных в интерфейсе отображаются персональные рекомендации.

3.9 Реализация средств оценки прохождения курса

Оценка прохождения курса реализована поэтапно:

- 1) Студент записывается на курс;
 - 2) После прохождения материалов и выполнения заданий загружает работу (POST /enrollments/:id/submit);
 - 3) Преподаватель проверяет работу и принимает решение о её одобрении (POST /submissions/:id/approve);
 - 4) Студент вручную подтверждает завершение курса (PUT /enrollments/:id/progress);
 - 5) После завершения курса автоматически создаётся запись в таблице.
- Эти этапы обеспечивают достаточную логику для реализации оценки.

Контроль осуществляется через проверку наличия одобренной работы (`is_approved = true`). Без её прохождения завершение курса блокируется на уровне сервера (листинг 8).

Листинг 8 – Алгоритм выдачи сертификата при завершении курса

```
if (isCompleted && !certificateExists) {
  await db.query(`
    INSERT INTO certificates (user_id, course_id, issue_date)
    VALUES ($1, $2, NOW())`,
    [userId, courseId]
  );
}
```

Диаграмма последовательности для алгоритма записи студента на курс представлена на рисунке 38.

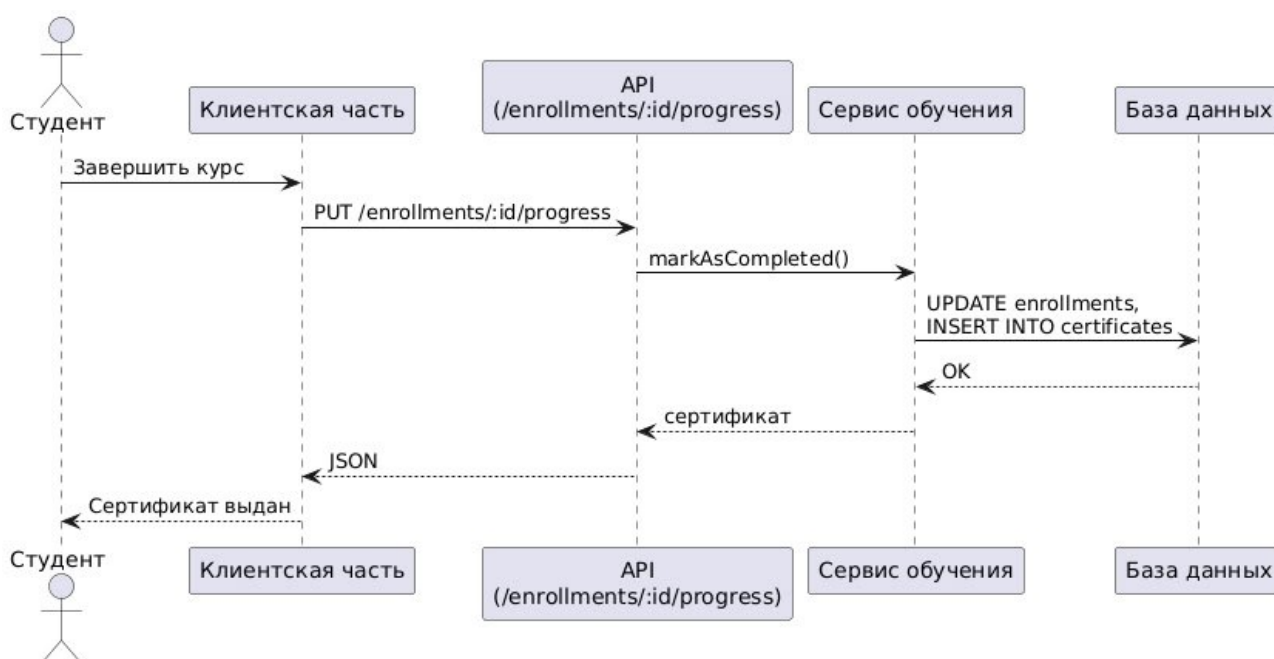


Рисунок 38 – Диаграмма последовательности для алгоритма выдачи сертификата при завершении курса

Макет страницы сертификатов приведен на рисунке 39.

Макет представляет собой интерфейс, предназначенный для отображения всех сертификатов, полученных пользователем по завершению курсов на образовательной платформе.

Мои сертификаты

Курс	Дата	ID Код	Ссылка
test1	6/6/2025	671A5166	Открыть

1 2 3 4

Рисунок 39 – Макет страницы сертификатов

Рассмотрим основные элементы макета страницы сертификатов разработанного приложения, представленные на рисунке 39:

- 1 – название курса;
- 2 – дата получения сертификата;
- 3 – уникальный код сертификата;
- 4 – ссылка на документ.

Сертификат сопровождается уникальным кодом верификации и URL-ссылкой. Он может быть проверен работодателем вручную или автоматически.

4 Тестирование образовательного портала

4.1 Функциональное тестирование

Функциональное тестирование было направлено на проверку корректности реализации пользовательских сценариев, обработки действий во всех ролевых зонах, а также соблюдение ограничений доступа.

Для формирования тестовых сценариев использован метод попарного тестирования (pairwise testing), который позволяет на основе анализа всех значимых параметров построить комбинированные проверки таким образом, чтобы каждая пара возможных значений параметров была встречена хотя бы раз.

Переменные для тестирования:

- Роль пользователя: Студент, Преподаватель, Работодатель, Администратор;
- Форма взаимодействия: Авторизация, Доступ к API, Запись на курс, Создание курса, Отклик на вакансию;
- Компонент интерфейса: /courses, /auth, /enrollments, /admin, /vacancies.

На основе анализа этих переменных была сформирована таблица входных данных (таблица 15).

Таблица 15 – Входные данные для попарного тестирования

№	Назначение запроса	HTTP	Маршрут API	Роль	Входные данные
1	Регистрация студента	POST	/auth/register	гость	username, email, password, role = student
2	Авторизация студента	POST	/auth/login	студент	email, password
3	Регистрация преподавателя	POST	/auth/register	гость	username, email, password, role = teacher
4	Авторизация преподавателя	POST	/auth/login	преподаватель	email, password
5	Создание курса	POST	/courses	преподаватель	title, description, difficulty

Продолжение таблицы 15

№	Назначение запроса	HTTP	Маршрут API	Роль	Входные данные
6	Запись студента на курс	POST	/enrollments	студент	course_id
7	Просмотр каталога	GET	/courses	любой	—

Для проведения функционального тестирования веб-приложения был выбран модульный набор инструментов, обеспечивающий удобство построения, автоматизации и документирования тестов для REST API [16].

В качестве основного средства тестирования использовалась среда Postman – популярный инструмент для разработки, отправки и отладки HTTP-запросов. Postman предоставляет возможность создания коллекций запросов, организации их в группы, управления тестовыми данными, применения переменных окружения, а также написания автотестов с использованием встроенного JavaScript API.

Каждый сценарий тестирования включал:

- выполнение запроса (с токеном соответствующей роли);
- контроль корректности ответа (код статуса и тело ответа);
- проверку ожидаемого результата в базе данных (через SELECT-запрос или отображение на клиентской части).

Результаты тестирования приведены в таблице 16 и рисунке 40.

Таблица 16 – Результаты тестирования

№	Название теста	Метод / Маршрут	Код ответа	Фактический результат	Статус
1	Регистрация студента	POST /auth/register	201	Пользователь создан	Успешно
2	Авторизация студента	POST /auth/login	200	Токен получен, сохранён	Успешно
3	Регистрация преподавателя	POST /auth/register	201	Преподаватель добавлен	Успешно
4	Авторизация преподавателя	POST /auth/login	200	Токен получен и сохранён	Успешно

Продолжение таблицы 16

№	Название теста	Метод / Маршрут	Код ответа	Фактический результат	Статус
5	Создание курса	POST /courses	201	Курс создан, id сохранён	Успешно
6	Запись студента на курс	POST /enrollments	201	Запись подтверждена	Успешно
7	Просмотр курсов (гость)	GET /courses	200	Вернулся список JSON	Успешно

Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	EduPlatform Testing	1	804ms	7	42 ms

Method	Test Name	URL	Status	Response Time	Response Size
POST	1. Регистрация студента	http://localhost:5000/api/auth/register	201	85 ms	1.213 KB
POST	2. Логин студента	http://localhost:5000/api/auth/login	200	58 ms	1.227 KB
POST	3. Регистрация преподавателя	http://localhost:5000/api/auth/register	201	67 ms	1.213 KB
POST	4. Логин преподавателя	http://localhost:5000/api/auth/login	200	57 ms	1.227 KB
POST	5. Создание курса (преподаватель)	http://localhost:5000/api/courses	201	16 ms	1.124 KB
POST	6. Запись на курс (студент)	http://localhost:5000/api/enrollments	201	9 ms	1.032 KB
GET	7. Просмотр курсов (гость)	http://localhost:5000/api/courses	200	5 ms	2.942 KB

Рисунок 40 – Результаты тестирования в Postman

По результатам тестирования установлено, что все ключевые сценарии, связанные с ролевыми действиями, обрабатываются корректно, ошибки перехватываются и сопровождаются соответствующими сообщениями. На таблице 17 приведены тест-кейсы.

Тестирование проводилось согласно современным стандартам, изученным в процессе разработки.

Таблица 17 – Тест-кейсы API

№	Назначение запроса	Описание теста	Ожидаемый результат
1	Регистрация студента	Создание нового пользователя с ролью студента	201 Created
2	Авторизация студента	Получение доступа и токена авторизации	200 OK + JWT
3	Регистрация преподавателя	Создание аккаунта преподавателя	201 Created
4	Авторизация преподавателя	Получение токена для преподавателя	200 OK + JWT
5	Создание курса	Добавление нового курса преподавателем	201 Created + courseId
6	Запись на курс	Добавление записи об обучении в таблицу enrollments	201 Created
7	Просмотр курсов	Запрос списка доступных курсов	200 OK + JSON

Таким образом, проведённое функциональное тестирование подтвердило соответствие поведения системы ожидаемым сценариям пользователя.

4.2 Нефункциональное тестирование

Нефункциональные испытания были направлены на проверку качественных характеристик образовательной платформы: производительности, устойчивости и потенциальной масштабируемости. Основным видом испытаний стало нагрузочное тестирование.

Нефункциональное тестирование призвано подтвердить соответствие программного продукта установленным требованиям по производительности, надёжности и масштабируемости [17]. Испытания выполнялись методом нагрузочного моделирования при помощи ПО Apache JMeter 5.6.2.

Аппаратная платформа: 8 виртуальных ядер, 16 Гбайт ОЗУ, среда – Windows 11, WSL 2. Такой подход сводит сетевые задержки к минимуму и позволяет оценить исключительно серверную часть системы.

В файл сценария заложены следующие параметры: базовый адрес службы `http://localhost:5000`; одна группа потоков (Thread Group); количество виртуальных пользователей: 10, 20, 30, 40 и 50; время разгона (ramp-up) – 10 с; количество циклов на пользователя: 5; вызываемые методы API: `GET /api/courses`, `GET /api/categories`; проверка кода ответа (должен быть 200).

Для регистрации результатов использованы штатные средства JMeter «View Results Tree» и «Aggregate Report».

Контроль осуществлялся по показателям: среднее время отклика, 95-й перцентиль, пропускная способность, процент ошибок. Целевые значения заданы следующим образом: при 10 одновременных пользователях среднее время реакции сервера ≤ 300 мс; при увеличении нагрузки до 50 пользователей число ошибочных ответов должно оставаться < 1 %.

Результаты нагрузочного тестирования в среде JMeter приведены на рисунке 41.

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received ...	Sent KB/sec
GET /api/c...	50	4	3	6	8	45	3	45	0.00%	5.5/sec	15.77	0.67
GET /api/c...	50	3	3	4	5	11	2	11	0.00%	5.6/sec	5.88	0.69
TOTAL	100	4	3	5	8	18	2	45	0.00%	11.1/sec	21.61	1.35

Рисунок 41 – Результаты тестирования в JMeter

Таким образом, текущая версия серверного приложения удовлетворяет нефункциональным требованиям при типичной одновременной нагрузке до 40 пользователей.

4.3 Метрики кода

Для объективной оценки качества программного решения были рассчитаны количественные метрики исходного кода: размер кода – число строк (SLOC – *Source Lines of Code*), сложность кода – метрика Холстеда для одного из модулей.

4.3.1 Размер кода (SLOC)

Подсчёт выполнен при помощи утилиты `cloc v1.96`, запущенной из корня репозитория. Результаты приведены на рисунке 42.

Language	files	blank	comment	code
JSON	5	0	0	19442
JavaScript	56	415	35	2694
Markdown	2	33	0	40
HTML	1	3	20	20
SUM:	64	451	55	22196

Рисунок 42 – Результат измерения SLOC

Суммарный полезный объём кода составляет 22196 SLOC. Показатель позволяет оценить масштаб проекта и служит базовой нормой при планировании усилий на сопровождение.

4.3.2 Метрика Холстеда

Объектом расчёта выбран файл `server/controllers/course.controller.js` – ключевой модуль серверной части. На таблице 18 представлен подсчет операторов.

Таблица 18 – Подсчет операторов

№	Оператор	Назначение	Кол-во (Ni)
1	<code>.</code>	обращение к полю / методу	40
2	<code>=</code>	присваивание	30
3	<code>await</code>	ожидание Promise	16
4	<code>Return</code>	возврат значения	14
5	<code>const</code>	объявление константы	13

Продолжение таблицы 18

№	Оператор	Назначение	Кол-во (Ni)
6	if	условный оператор	12
7	()	вызов функции	9
8	{}	блок кода	8
9	;	конец инструкции	8
10	try	начало блока обработки исключений	6
11	catch	перехват исключения	6
12	throw	возбуждение исключения	4
13	>	логическое «больше»	3
14	<	логическое «меньше»	3
15	===	строгое равенство	3
16	!==	строгое неравенство	2
17	&&	логическое «И»	2
18	`		1
19	[]	обращение по индексу	2
20	? :	тернарный оператор	1
21	+	конкатенация строк	1
22	-	вычитание / отрицание	1
23	typeof	определение типа	1
24	in	проверка наличия свойства	1

Исходные данные, полученные с помощью escomplex-cli: ($n_1 = 24$; $N_1 = 158$; $n_2 = 57$; $N_2 = 213$).

Формулы Холстеда:

$$n = n_1 + n_2;$$

$$N = N_1 + N_2;$$

$$V = N \cdot \log_2 n;$$

$$D = (n_1 / 2) \cdot (N_2 / n_2);$$

$$E = V \cdot D;$$

$$T = E / 18;$$

$$B = V / 3\,000,$$

где

n_1 – число различных операторов;

n_2 – число различных операндов;

N_1 – общее число операторов;

N_2 – общее число операндов;

n – словарь программы;

N – длина программы;

V – объём программы, слов · бит;

D – трудоёмкость (Difficulty);

E – усилие (Effort), усл. ед.;

T – расчётное время реализации, с;

B – прогноз числа дефектов.

Подстановка значений:

$$n = 24 + 57 = 81;$$

$$N = 158 + 213 = 371;$$

$$V = 371 \cdot \log_2(81) \approx 371 \cdot 6,34 = 2\,353;$$

$$D = (24 / 2) \cdot (213 / 57) \approx 12 \cdot 3,74 = 44,9;$$

$$E = 2\,353 \cdot 44,9 \approx 105\,700;$$

$$T = 105\,700 / 18 \approx 5\,872 \text{ с} \approx 1,63 \text{ ч};$$

$$B = 2\,353 / 3\,000 \approx 0,78.$$

Таким образом были произведены расчеты метрик Холстеда и выявлена контролируемая сложность.

4.3.3 Интерпретация результатов

SLOC примерно равен 22000 строк кода соответствует среднему дипломному проекту и укладывается в нормативы трудоёмкости.

По Холстеду объём модуля 2,3 кбит, расчётная трудоёмкость 1,6 ч. Практическое время, затраченное на написание данного контроллера, составило 2 ч, что подтверждает адекватность модели.

Оценка дефектности меньше 1 указывает на невысокий риск ошибок логики, однако при дальнейшем расширении функциональности рекомендуется: поддерживать покрытие unit-тестами; применять разбор кода и статический анализ (ESLint, SonarQube).

Таким образом, количественные метрики показывают, что проект имеет приемлемый объём и контролируемую сложность, что положительно сказывается на сопровождаемости и качестве конечного продукта.

Заключение

В рамках выполнения данной выпускной квалификационной работы была разработана Web-платформа образовательных курсов компаний-работодателей, предназначенная для организации процесса дистанционного обучения с возможностью тесной интеграции с рынком труда. Разработка велась в соответствии с требованиями, вытекающими из анализа предметной области и целей проекта.

Проанализированы существующие программные решения в сфере онлайн-обучения, выявлены их преимущества и недостатки. На основе сопоставления было определено целевое позиционирование разрабатываемой платформы – как системы, позволяющей студентам не только осваивать профессиональные навыки, но и применять их в реальных карьерных сценариях через интеграцию с работодателями.

В процессе проектирования была спроектирована архитектура программной системы, включающая клиентскую и серверную части, а также реляционную базу данных. Разработаны диаграмма контекста, диаграммы потоков данных (DFD), компоненты REST API, модели базы данных. Использована современная стек-технологий: React.js для клиентской части, Node.js (Express) – для серверной, PostgreSQL – в качестве СУБД.

Реализована поддержка четырех ролей пользователей – студента, преподавателя, работодателя и администратора. Каждая роль имеет собственный набор функций, реализованных через ролевую систему доступа (RBAC), что обеспечивает гибкость и масштабируемость платформы. Особое внимание уделено функциональности для работодателей – создан модуль создания вакансий и карьерных траекторий, а также рекомендаций студентам курсов, повышающих шансы на трудоустройство.

Ключевыми функциональными модулями стали: управление обучением, система отзывов, генерация сертификатов, механизм откликов на вакансии и рекомендательная система подбора курсов. Также обеспечена административная панель с функциями управления пользователями и системой в целом.

Платформа прошла модульное тестирование и интеграционную проверку работоспособности ключевых компонентов. В результате достигнута основная цель проекта – реализовано минимально жизнеспособное программное решение (MVP), функциональность которого может быть расширена и адаптирована под образовательные и корпоративные потребности.

Разработанная система может быть использована в образовательных учреждениях, профессиональных центрах, а также непосредственно компаниями-работодателями, заинтересованными в росте квалифицированных кадров. Техническая архитектура и уровень реализации позволяют адаптировать и расширять её под конкретные цели бизнеса или учебного процесса.

Таким образом, можно сделать вывод, что поставленные цели и задачи выпускной квалификационной работы выполнены в полном объеме. Разработанный программный продукт соответствует современным требованиям к электронным образовательным системам и имеет высокую практическую значимость.

Список использованных источников

1. Соммервилл, И. Инженерия программного обеспечения / И. Соммервилл. – М.: Вильямс, 2018. – 784 с.
2. Фаулер, М. Архитектура корпоративных программных приложений / М. Фаулер. – М.: Вильямс, 2019. – 544 с.
3. Гарнаев, А. Ю. Анализ рынка образовательных платформ / А. Ю. Гарнаев // Вестник образования. – 2022. – № 5. – С. 45–50.
4. Официальные сайты платформ Coursera, Udemу, Stepik [Электронный ресурс]. – URL: <https://www.coursera.org/>, <https://www.udemy.com/>, <https://stepik.org/> (дата обращения: 01.11.2023).
5. Буч, Г. Объектно-ориентированный анализ и проектирование с примерами приложений / Г. Буч, Р. Максимчук, М. Энгл. – М.: Вильямс, 2010. – 720 с.
6. ISO/IEC 25010:2011. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models.
7. Web Content Accessibility Guidelines (WCAG) 2.1 [Электронный ресурс]. – URL: <https://www.w3.org/TR/WCAG21/> (дата обращения: 01.11.2023).
8. Фримен, Э. Изучаем HTML, XHTML и CSS / Э. Фримен, Э. Фримен. – СПб.: Питер, 2010. – 656 с.
9. TIOBE Index for November 2023 [Электронный ресурс]. – URL: <https://www.tiobe.com/tiobe-index/> (дата обращения: 01.11.2023).
10. Deitel, P. JavaScript for Programmers / P. Deitel, H. Deitel. – Boston: Pearson, 2019. – 640 p.
11. Официальная документация PostgreSQL [Электронный ресурс]. – URL: <https://www.postgresql.org/docs/> (дата обращения: 01.11.2023).

12. Fielding, R. T. Architectural Styles and the Design of Network-based Software Architectures / R. T. Fielding. – Irvine: University of California, 2000. – 180 p.
13. Norman, D. The Design of Everyday Things / D. Norman. – New York: Basic Books, 2013. – 368 p.
14. Коннолли, Т. Базы данных: проектирование, реализация и сопровождение / Т. Коннолли, К. Бегг. – М.: Вильямс, 2003. – 1440 с.
15. OpenAPI Specification [Электронный ресурс]. – URL: <https://swagger.io/specification/> (дата обращения: 01.11.2023).
16. Бек, К. Экстремальное программирование: разработка через тестирование / К. Бек. – СПб.: Питер, 2003. – 224 с.
17. Агапов А. И. Тестирование программного обеспечения: практика разработки и методология. – СПб.: БХВ-Петербург, 2021. – 352 с.
18. Глушаков И. В. Архитектура информационных систем. – М.: Инфра-М, 2019. – 280 с.
19. Макаров И. Д. Проектирование и разработка пользовательских интерфейсов: учебник. – М.: Дашков и К°, 2021. – 192 с.
20. Express.js – Документация по использованию [Электронный ресурс]. – URL: <https://expressjs.com/ru/> (дата обращения: 10.06.2025).
21. PostgreSQL – Официальная документация [Электронный ресурс]. – URL: <https://www.postgresql.org/docs/> (дата обращения: 10.06.2025).

Приложение А
(обязательное)

server/package.json:

```
{
  "name": "server",
  "version": "1.0.0",
  "main": "app.js",
  "scripts": {
    "start": "node app.js",
    "dev": "nodemon app.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "bcrypt": "^6.0.0",
    "cors": "^2.8.5",
    "dotenv": "^16.5.0",
    "express": "^5.1.0",
    "helmet": "^8.1.0",
    "jsonwebtoken": "^9.0.2",
    "multer": "^2.0.1",
    "pg": "^8.16.0",
    "uuid": "^11.1.0"
  },
  "devDependencies": {
    "nodemon": "^3.1.10"
  }
}
```

server/app.js:

```
const express = require("express");
const cors = require("cors");
const helmet = require("helmet");
const dotenv = require("dotenv");
const path = require("path");

dotenv.config();

const app = express();
const port = process.env.PORT || 5000;

// Подключение middleware
app.use(cors());
app.use(helmet());
app.use(express.json());

const authRoutes = require("./routes/auth.routes");
const courseRoutes = require("./routes/course.routes");
const enrollmentRoutes = require("./routes/enrollment.routes");
const userRoutes = require("./routes/user.routes");
const certificateRoutes = require("./routes/certificate.routes");
const adminRoutes = require("./routes/admin.routes");
const reviewRoutes = require("./routes/review.routes");
```

```

const categoryRoutes = require("./routes/category.routes");
const submissionRoutes = require("./routes/submission.routes");
const recommendationRoutes = require("./routes/recommendation.routes");
const vacancyRoutes = require("./routes/vacancy.routes");

app.use("/uploads", express.static(path.join(__dirname, "../uploads")));
// Подключение к БД
require("./config/db");

// Тестовый маршрут
app.get("/", (req, res) => {
  res.send("👋 Educational Platform сервер работает");
});

app.use("/api/auth", authRoutes);
app.use("/api/courses", courseRoutes);
app.use("/api/enrollments", enrollmentRoutes);
app.use("/api/users", userRoutes);
app.use("/api/certificates", certificateRoutes);
app.use("/api/admin", adminRoutes);
app.use("/api/courses/:courseId/reviews", reviewRoutes);
app.use("/api/categories", categoryRoutes);
app.use("/api/submissions", submissionRoutes);
app.use("/api/students/:id/recommendations", recommendationRoutes);
app.use("/api/vacancies", vacancyRoutes);

// Запуск сервера
app.listen(port, () => console.log(`🚀 Сервер запущен на
http://localhost:${port}`));

```

server/utils/jwt.js:

```

const jwt = require("jsonwebtoken");

const generateToken = (user) => {
  return jwt.sign(
    { id: user.id, email: user.email },
    process.env.JWT_SECRET,
    { expiresIn: "7d" }
  );
};

const verifyToken = (token) => {
  return jwt.verify(token, process.env.JWT_SECRET);
};

module.exports = { generateToken, verifyToken };

```

server/utils/password.js:

```

const bcrypt = require("bcrypt");

const hashPassword = async (password) => {
  return await bcrypt.hash(password, 10);
};

```

```

const comparePassword = async (password, passwordHash) => {
  return await bcrypt.compare(password, passwordHash);
};

module.exports = { hashPassword, comparePassword };

server/utils/fileUploader.js:

const multer = require('multer');
const path = require('path');

// хранилище для разных типов файлов
const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, '../uploads'); // папка
  },
  filename: function (req, file, cb) {
    const ext = path.extname(file.originalname);
    const uniqueName = `${Date.now()}-${Math.round(Math.random() *
1e9)}${ext}`;
    cb(null, uniqueName);
  },
});

const upload = multer({ storage });

module.exports = upload;

server/config/db.js:

const { Pool } = require("pg");
require("dotenv").config();

const pool = new Pool({
  connectionString: process.env.DATABASE_URL,
});

pool.connect()
  .then(() => console.log("✅ PostgreSQL подключен"))
  .catch((err) => console.error("❌ Ошибка подключения к БД", err));

module.exports = pool;

server/services/enrollment.service.js:

const EnrollmentModel = require("../models/enrollment.model");

const enroll = async (user_id, course_id) => {
  return await EnrollmentModel.enrollInCourse(user_id, course_id);
};

const listEnrollments = async (user_id) => {
  return await EnrollmentModel.getUserEnrollments(user_id);
};

```

```

const update = async (enroll_id, isCompleted) => {
  return await EnrollmentModel.updateProgress(enroll_id, isCompleted);
};

module.exports = {
  enroll,
  listEnrollments,
  update
};

server/services/review.service.js:

const ReviewModel = require("../models/review.model");

const add = async (userId, courseId, rating, text) => {
  return await ReviewModel.addReview(userId, courseId, rating, text);
};

const listByCourse = async (courseId) => {
  return await ReviewModel.getReviewsByCourse(courseId);
};

module.exports = { add, listByCourse };

server/services/admin.service.js:

const db = require("../config/db");

exports.getStats = async () => {
  const users = await db.query(`SELECT COUNT(*) FROM users`);
  const courses = await db.query(`SELECT COUNT(*) FROM courses`);
  const students = await db.query(`SELECT COUNT(*) FROM students`);
  const completed = await db.query(`SELECT COUNT(*) FROM enrollments WHERE
is_completed = true`);

  return {
    users: +users.rows[0].count,
    students: +students.rows[0].count,
    courses: +courses.rows[0].count,
    completedCourses: +completed.rows[0].count,
  };
};

exports.getAllUsers = async () => {
  const result = await db.query(`
    SELECT id, username, email, is_blocked FROM users ORDER BY id
  `);
  return result.rows;
};

exports.toggleBlock = async (userId) => {
  const result = await db.query(`
    UPDATE users
    SET is_blocked = NOT is_blocked
    WHERE id = $1
  `);
};

```

```

    RETURNING id, username, is_blocked
  `, [userId]);

  return result.rows[0];
};

exports.changeRole = async (userId, newRole) => {
  const validRoles = ['student', 'teacher', 'employer'];
  if (!validRoles.includes(newRole)) {
    return { success: false, message: "Недопустимая роль" };
  }

  // Удалим из всех роль-таблиц
  await db.query(`DELETE FROM students WHERE user_id = $1`, [userId]);
  await db.query(`DELETE FROM teachers WHERE user_id = $1`, [userId]);
  await db.query(`DELETE FROM employers WHERE user_id = $1`, [userId]);

  if (newRole === 'student') {
    await db.query(`INSERT INTO students (user_id) VALUES ($1)`, [userId]);
  }
  if (newRole === 'teacher') {
    await db.query(`INSERT INTO teachers (user_id) VALUES ($1)`, [userId]);
  }
  if (newRole === 'employer') {
    await db.query(`INSERT INTO employers (user_id, company_name) VALUES
($1, '')`, [userId]);
  }

  return { success: true };
};

server/services/course.service.js:

const CourseModel = require("../models/course.model");

const create = async (data) => {
  return await CourseModel.createCourse(data);
};

const list = async () => {
  return await CourseModel.getAllCourses();
};

const getOne = async (id) => {
  return await CourseModel.getCourseById(id);
};

const update = async (id, data) => {
  return await CourseModel.updateCourse(id, data);
};

const remove = async (id) => {
  return await CourseModel.deleteCourse(id);
};

module.exports = {
  create,

```

```

    list,
    getOne,
    update,
    remove,
  };

```

server/services/user.service.js:

```

const db = require("../config/db");
const { hashPassword, comparePassword } = require("../utils/password");

const createUser = async ({ username, email, password }) => {
  const passwordHash = await hashPassword(password);

  const result = await db.query(
    `INSERT INTO users (username, email, password_hash)
    VALUES ($1, $2, $3)
    RETURNING id, username, email, created_at`,
    [username, email, passwordHash]
  );

  return result.rows[0];
};

const getUserByEmail = async (email) => {
  const result = await db.query(
    `SELECT * FROM users WHERE email = $1`,
    [email]
  );
  return result.rows[0];
};

module.exports = {
  createUser,
  getUserByEmail,
};

```

server/middleware/role.js:

```

const hasRole = (role) => (req, res, next) => {
  if (!req.user) return res.status(401).json({ message: "Не авторизовано"
});

  const jwt = require("jsonwebtoken");
  const decoded = jwt.decode(req.headers['authorization']?.split("
")[1]);

  const userId = decoded?.id;

  const db = require("../config/db");

  const roleTableMap = {
    admin: "admins",
    student: "students",
    teacher: "teachers",
    employer: "employers",
  };

```

```

    const table = roleTableMap[role];
    if (!table) return res.status(403).json({ message: "Роль не найдена"
  });

  db.query(`SELECT 1 FROM ${table} WHERE user_id = $1`, [userId])
    .then(result => {
      if (result.rowCount > 0) {
        next();
      } else {
        res.status(403).json({ message: "Доступ только для " + role });
      }
    })
    .catch(err => res.status(500).json({ message: "Ошибка проверки роли"
  }));

  });

  module.exports = { hasRole };

server/middleware/auth.js:

const jwt = require("jsonwebtoken");

const verifyToken = (req, res, next) => {
  const authHeader = req.headers["authorization"];
  const token = authHeader && authHeader.split(" ")[1]; // Bearer token

  if (!token) return res.status(401).json({ message: "Нет токена" });

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded; // id, email
    next();
  } catch (err) {
    return res.status(403).json({ message: "Неверный токен" });
  }
};

module.exports = { verifyToken };

server/routes/vacancy.routes.js:

const express = require("express");
const router = express.Router();
const { verifyToken } = require("../middleware/auth");
const VacancyController = require("../controllers/vacancy.controller");

router.post("/", verifyToken, VacancyController.createVacancy);
router.post("/apply", verifyToken, VacancyController.applyToVacancy); // ✓
новый
router.get("/", async (req, res) => {
  const db = require("../config/db");
  try {
    const result = await db.query(`SELECT * FROM vacancies WHERE is_active
= true ORDER BY created_at DESC`);
    res.json(result.rows);
  }
});

```

```

    } catch (err) {
      console.error("Ошибка загрузки вакансий:", err);
      res.status(500).json({ message: "Ошибка сервера" });
    }
  });
  router.get("/:vacancyId/recommended/:studentId", verifyToken,
VacancyController.getRecommendedCourses);
  router.get("/:id/applications", verifyToken,
VacancyController.getApplicationsByVacancy);
  router.get("/my", verifyToken, VacancyController.getMyVacancies);

  module.exports = router;

server/routes/course.routes.js:

const express = require("express");
const router = express.Router();
const CourseController = require("../controllers/course.controller");
const { verifyToken } = require("../middleware/auth");
const upload = require('../utils/fileUploader');

router.get("/my", verifyToken, CourseController.getMyCourses);
// Публичные
router.get("/", CourseController.getAllCourses);
router.get("/:id", CourseController.getCourseById);

// Только авторизованные
router.post("/", verifyToken, CourseController.createCourse);
router.put("/:id", verifyToken, CourseController.updateCourse);
router.delete("/:id", verifyToken, CourseController.deleteCourse);
router.get("/:id/students", verifyToken,
CourseController.getEnrolledStudents);
router.post("/:id/material", verifyToken, upload.single('file'),
CourseController.uploadMaterial);
router.get("/:id/material", CourseController.getCourseMaterials);

module.exports = router;

server/routes/submission.routes.js:

const express = require("express");
const router = express.Router();
const { verifyToken } = require("../middleware/auth");
const SubmissionController = require("../controllers/submission.controller");

// Только автор курса
router.get("/course/:courseId", verifyToken,
SubmissionController.getSubmissionsByCourse);

// Только автор курса
router.post("/:id/approve", verifyToken,
SubmissionController.approveSubmission);

module.exports = router;

```

```
server/routes/category.routes.js:
```

```
const express = require("express");
const router = express.Router();
const CategoryController = require("../controllers/category.controller");

router.get("/", CategoryController.getAllCategories);
router.get("/skills", CategoryController.getAllSkills);
module.exports = router;
```

```
server/routes/auth.routes.js:
```

```
const express = require("express");
const router = express.Router();
const AuthController = require("../controllers/auth.controller");
```

```
router.post("/register", AuthController.register);
router.post("/login", AuthController.login);
```

```
module.exports = router;
```

```
server/routes/admin.routes.js:
```

```
const express = require("express");
const router = express.Router();
const { verifyToken } = require("../middleware/auth");
const { hasRole } = require("../middleware/role");
const AdminController = require("../controllers/admin.controller");
```

```
// Только для админов
router.use(verifyToken, hasRole('admin'));
```

```
router.get("/stats", AdminController.getPlatformStats);
router.get("/users", AdminController.getAllUsers);
router.put("/users/:id/block", AdminController.toggleBlock);
router.put("/users/:id/role", AdminController.changeRole);
```

```
module.exports = router;
```

```
server/routes/enrollment.routes.js:
```

```
const express = require("express");
const router = express.Router();
const EnrollmentController = require("../controllers/enrollment.controller");
const { verifyToken } = require("../middleware/auth");
const upload = require("../utils/fileUploader");

router.post("/", verifyToken, EnrollmentController.enroll);
router.get("/", verifyToken, EnrollmentController.list);
router.put("/:id/progress", verifyToken, EnrollmentController.updateProgress);
router.delete("/:courseId", verifyToken, EnrollmentController.unenroll);
```

```

        router.post("/:courseId/submit", verifyToken, upload.single("file"),
        EnrollmentController.submitWork);

    module.exports = router;

server/routes/review.routes.js:

const express = require("express");
const router = express.Router({ mergeParams: true });
const ReviewController = require("../controllers/review.controller");
const { verifyToken } = require("../middleware/auth");

// Получить отзывы по курсу
router.get("/", ReviewController.getReviews);

// Оставить отзыв (только авторизованные)
router.post("/", verifyToken, ReviewController.addReview);

module.exports = router;

server/routes/user.routes.js:

const express = require("express");
const router = express.Router();
const { verifyToken } = require("../middleware/auth");
const UserController = require("../controllers/user.controller");

router.get("/me", verifyToken, UserController.getMe);

module.exports = router;

server/routes/recommendation.routes.js:

const express = require("express");
const router = express.Router();
const { verifyToken } = require("../middleware/auth");
const RecommendationController = require("../controllers/recommendation.controller");

router.get("/vacancies", verifyToken, RecommendationController.getRecommendedVacancies);
router.get("/trajectories", verifyToken, RecommendationController.getTrajectoryMatches);

module.exports = router;

server/routes/certificate.routes.js:

const express = require("express");
const router = express.Router();
const { verifyToken } = require("../middleware/auth");
const CertificateController = require("../controllers/certificate.controller");

```

```
router.get("/", verifyToken, CertificateController.getMyCertificates);
```

```
module.exports = router;
```

```
server/models/review.model.js:
```

```
const db = require("../config/db");
```

```
const addReview = async (student_id, course_id, rating, text) => {
  const result = await db.query(
    `INSERT INTO course_reviews (student_id, course_id, rating, text)
    VALUES ($1, $2, $3, $4)
    ON CONFLICT (student_id, course_id)
    DO UPDATE SET rating = $3, text = $4, created_at = NOW()
    RETURNING *`,
    [student_id, course_id, rating, text]
  );
  return result.rows[0];
};
```

```
const getReviewsByCourse = async (course_id) => {
  const result = await db.query(
    `SELECT r.*, u.username
    FROM course_reviews r
    JOIN users u ON r.student_id = u.id
    WHERE r.course_id = $1
    ORDER BY r.created_at DESC`,
    [course_id]
  );
  return result.rows;
};
```

```
module.exports = {
  addReview,
  getReviewsByCourse,
};
```

```
server/models/enrollment.model.js:
```

```
const db = require("../config/db");
```

```
const { v4: uuidv4 } = require("uuid");
```

```
const enrollInCourse = async (user_id, course_id) => {
  const result = await db.query(
    `INSERT INTO enrollments (user_id, course_id)
    VALUES ($1, $2) RETURNING *`,
    [user_id, course_id]
  );
  return result.rows[0];
};
```

```
const getUserEnrollments = async (user_id) => {
  const result = await db.query(
    `SELECT e.*, c.title, c.difficulty, e.course_id
    FROM enrollments e
    JOIN courses c ON e.course_id = c.id
```

```

        WHERE e.user_id = $1
        ORDER BY e.enrolled_at DESC`,
        [user_id]
    );
    return result.rows;
};

const updateProgress = async (enrollId, isCompleted) => {
    // получаем информацию о записи
    const enrollRes = await db.query(`SELECT * FROM enrollments WHERE id =
    $1`, [enrollId]);
    const updated = enrollRes.rows[0];

    if (!updated) throw new Error("Запись не найдена");

    // Если курс завершается - проверим is_approved
    if (isCompleted) {
        const submission = await db.query(
            `SELECT is_approved FROM student_submissions
            WHERE student_id = $1 AND course_id = $2`,
            [updated.user_id, updated.course_id]
        );

        const approved = submission.rows[0]?.is_approved;

        if (!approved) {
            throw new Error("Невозможно завершить курс: работа ещё не одобрена
            преподавателем");
        }
    }

    // Обновим запись
    const result = await db.query(
        `UPDATE enrollments
        SET is_completed = $1,
            completed_at = CASE WHEN $1 THEN NOW() ELSE NULL END
        WHERE id = $2
        RETURNING *`,
        [isCompleted, enrollId]
    );

    const finalRow = result.rows[0];

    if (isCompleted && finalRow) {
        const exists = await db.query(
            `SELECT 1 FROM certificates WHERE user_id = $1 AND course_id = $2`,
            [finalRow.user_id, finalRow.course_id]
        );

        if (exists.rowCount === 0) {
            const code = uuidv4().split("-")[0].toUpperCase();
            const certUrl = `https://edu-platform.local/certificates/${code}`;

            await db.query(
                `INSERT INTO certificates (user_id, course_id, issue_date,
                verification_code, certificate_url)
                VALUES ($1, $2, NOW(), $3, $4)`,
                [finalRow.user_id, finalRow.course_id, code, certUrl]
            );
        }
    }
};

```

```

        );
    }
}

return finalRow;
};

module.exports = {
    enrollInCourse,
    getUserEnrollments,
    updateProgress,
};

server/models/course.model.js:

const db = require("../config/db");

const createCourse = async ({ title, description, difficulty, teacher_id })
=> {
    const result = await db.query(
        `INSERT INTO courses (title, description, teacher_id, difficulty,
created_at)
        VALUES ($1, $2, $3, $4, NOW()) RETURNING *`,
        [title, description, teacher_id, difficulty]
    );
    return result.rows[0];
};

const getAllCourses = async () => {
    const result = await db.query(`SELECT * FROM courses ORDER BY created_at
DESC`);
    return result.rows;
};

const getCourseById = async (id) => {
    const result = await db.query(`SELECT * FROM courses WHERE id = $1`,
[id]);
    return result.rows[0];
};

const updateCourse = async (id, payload) => {
    const { title, description, difficulty } = payload;

    const result = await db.query(
        `UPDATE courses
        SET title = $1, description = $2, difficulty = $3
        WHERE id = $4 RETURNING *`,
        [title, description, difficulty, id]
    );
    return result.rows[0];
};

const deleteCourse = async (id) => {
    await db.query(`DELETE FROM courses WHERE id = $1`, [id]);
};

module.exports = {

```

```

    createCourse,
    getAllCourses,
    getCourseById,
    updateCourse,
    deleteCourse,
  };

server/controllers/review.controller.js:

const ReviewService = require("../services/review.service");

exports.addReview = async (req, res) => {
  try {
    const userId = req.user.id;
    const courseId = req.params.courseId;
    const { rating, text } = req.body;

    // rating должен быть от 1 до 5
    if (!rating || rating < 1 || rating > 5) {
      return res.status(400).json({ message: "Оценка должна быть от 1 до 5"
});
    }

    const review = await ReviewService.add(userId, courseId, rating, text);
    res.status(201).json(review);
  } catch (err) {
    console.error("Ошибка добавления отзыва:", err);
    res.status(500).json({ message: "Ошибка при добавлении отзыва" });
  }
};

exports.getReviews = async (req, res) => {
  try {
    const courseId = req.params.courseId;
    const reviews = await ReviewService.listByCourse(courseId);
    res.json(reviews);
  } catch (err) {
    console.error("Ошибка получения отзывов:", err);
    res.status(500).json({ message: "Ошибка при получении отзывов" });
  }
};

server/controllers/course.controller.js:

const CourseService = require("../services/course.service");
const db = require("../config/db");

exports.createCourse = async (req, res) => {
  try {
    const { title, description, difficulty } = req.body;
    const teacher_id = req.user.id;

    const course = await CourseService.create({ title, description,
difficulty, teacher_id });
    res.status(201).json(course);
  } catch (err) {

```

```

        console.error("Ошибка создания курса:", err);
        res.status(500).json({ message: "Ошибка сервера" });
    }
};

exports.getAllCourses = async (req, res) => {
    try {
        const { category, difficulty } = req.query;

        let query = `
            SELECT c.*, cat.title AS category_title
            FROM courses c
            LEFT JOIN categories cat ON c.category_id = cat.id
            WHERE 1=1
        `;
        const values = [];

        if (category) {
            values.push(category);
            query += ` AND c.category_id = ${values.length}`;
        }

        if (difficulty) {
            values.push(difficulty);
            query += ` AND c.difficulty = ${values.length}`;
        }

        query += ' ORDER BY c.created_at DESC';

        const result = await db.query(query, values);
        res.json(result.rows);
    } catch (err) {
        console.error("Ошибка получения курсов:", err);
        res.status(500).json({ message: "Ошибка сервера" });
    }
};

exports.getCourseById = async (req, res) => {
    try {
        const course = await CourseService.getOne(req.params.id);
        if (!course) return res.status(404).json({ message: "Курс не найден" });

        res.json(course);
    } catch (err) {
        res.status(500).json({ message: "Ошибка сервера" });
    }
};

exports.updateCourse = async (req, res) => {
    try {
        const course = await CourseService.update(req.params.id, req.body);
        res.json(course);
    } catch (err) {
        res.status(500).json({ message: "Ошибка обновления курса" });
    }
};

exports.deleteCourse = async (req, res) => {

```

```

    try {
      await CourseService.remove(req.params.id);
      res.status(204).end();
    } catch (err) {
      res.status(500).json({ message: "Ошибка удаления курса" });
    }
  };

exports.getEnrolledStudents = async (req, res) => {
  const courseId = req.params.id;
  const userId = req.user.id;

  // Проверка: пользователь - автор курса?
  const check = await db.query(
    `SELECT * FROM courses WHERE id = $1 AND teacher_id = $2`,
    [courseId, userId]
  );

  if (check.rowCount === 0) {
    return res.status(403).json({ message: "Вы не автор этого курса" });
  }

  // Получаем студентов
  const result = await db.query(`
    SELECT
      u.id,
      u.username,
      e.is_completed,
      e.enrolled_at,
      e.completed_at
    FROM enrollments e
    JOIN users u ON e.user_id = u.id
    WHERE e.course_id = $1
    ORDER BY e.enrolled_at DESC
  `, [courseId]);

  res.json(result.rows);
};

exports.getMyCourses = async (req, res) => {
  try {
    const teacherId = req.user.id;

    const result = await db.query(
      `SELECT id, title, description, difficulty, created_at
      FROM courses
      WHERE teacher_id = $1
      ORDER BY created_at DESC`,
      [teacherId]
    );

    res.json(result.rows); // список курсов
  } catch (err) {
    console.error("Ошибка получения моих курсов:", err);
    res.status(500).json({ message: "Ошибка сервера при получении курсов"
  });
}
};

```

```

exports.uploadMaterial = async (req, res) => {
  const userId = req.user.id;
  const courseId = req.params.id;

  try {
    const file = req.file;
    if (!file) return res.status(400).json({ message: 'Файл не загружен'
  });

  // проверим, что он автор курса
  const course = await db.query(`SELECT * FROM courses WHERE id=$1 AND
teacher_id=$2`, [courseId, userId]);

  if (course.rowCount === 0) {
    return res.status(403).json({ message: 'Нет доступа' });
  }

  await db.query(`
    INSERT INTO course_materials (course_id, filename, original_name)
    VALUES ($1, $2, $3)
  `, [courseId, file.filename, file.originalname]);

  res.json({ message: 'Файл успешно загружен', url:
`/uploads/${file.filename}` });
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Ошибка загрузки' });
  }
};

exports.getCourseMaterials = async (req, res) => {
  const courseId = req.params.id;
  const result = await db.query(`
    SELECT * FROM course_materials WHERE course_id = $1 ORDER BY uploaded_at
DESC
  `, [courseId]);

  res.json(result.rows);
};

server/controllers/recommendation.controller.js:

const db = require("../config/db");

exports.getRecommendedVacancies = async (req, res) => {
  const studentId = req.params.id;

  try {
    const result = await db.query(
      `
      SELECT v.*, COUNT(*) AS match_count
      FROM vacancies v
      JOIN vacancy_skills vs ON v.id = vs.vacancy_id
      JOIN student_skills ss ON vs.skill_id = ss.skill_id
      WHERE ss.student_id = $1 AND v.is_active = true
      GROUP BY v.id
    `
    );
  }
};

```

```

        ORDER BY match_count DESC
      `,
      [studentId]
    );

    res.json(result.rows);
  } catch (err) {
    console.error("Ошибка при подборе вакансий:", err);
    res.status(500).json({ message: "Ошибка сервера" });
  }
};

exports.getTrajectoryMatches = async (req, res) => {
  const studentId = req.params.id;

  try {
    const result = await db.query(
      `
      SELECT
        rt.id,
        rt.title,
        rt.description,
        COUNT(DISTINCT ec.course_id) as completed_courses,
        COUNT(DISTINCT tc.course_id) as total_courses,
        ROUND(100.0 * COUNT(DISTINCT ec.course_id) /
GREATEST(COUNT(DISTINCT tc.course_id),1)) as progress_percent
      FROM role_trajectories rt
      JOIN trajectory_courses tc ON rt.id = tc.trajectory_id
      LEFT JOIN enrollments ec ON ec.course_id = tc.course_id AND ec.user_id
= $1 AND ec.is_completed = true
      GROUP BY rt.id
      ORDER BY progress_percent DESC
      `,
      [studentId]
    );

    res.json(result.rows);
  } catch (err) {
    console.error("Ошибка при подборе траекторий:", err);
    res.status(500).json({ message: "Ошибка сервера" });
  }
};

server/controllers/user.controller.js:

const db = require("../config/db");

exports.getMe = async (req, res) => {
  try {
    const userId = req.user.id;

    const userRes = await db.query(
      `SELECT id, username, email FROM users WHERE id = $1`,
      [userId]
    );

    const user = userRes.rows[0];

```

```

        if (!user) return res.status(404).json({ message: "Пользователь не найден" });

        // Проверка ролей (можно сделать через JOIN)
        const roles = [];

        const checkRole = async (table, roleName) => {
            const result = await db.query(`SELECT 1 FROM ${table} WHERE user_id = $1`, [userId]);
            if (result.rowCount > 0) roles.push(roleName);
        };

        await checkRole("admins", "admin");
        await checkRole("students", "student");
        await checkRole("teachers", "teacher");
        await checkRole("employers", "employer");

        res.json({ ...user, roles });
    } catch (err) {
        console.error("Ошибка получения профиля:", err);
        res.status(500).json({ message: "Внутренняя ошибка сервера" });
    }
};

server/controllers/vacancy.controller.js:

const db = require("../config/db");

exports.createVacancy = async (req, res) => {
    const userId = req.user.id;

    try {
        const employer = await db.query(
            `SELECT * FROM employers WHERE user_id = $1`,
            [userId]
        );

        if (employer.rowCount === 0) {
            return res.status(403).json({ message: "Пользователь не работодатель"
});
        }

        const {
            title,
            description,
            salary_range,
            employment_type,
            remote_available,
            location,
            skill_ids // 📁 массив ID навыков из формы
        } = req.body;

        const result = await db.query(
            `INSERT INTO vacancies (employer_id, title, description,
            salary_range, employment_type, remote_available, location,
            created_at, is_active)
            VALUES ($1, $2, $3, $4, $5, $6, $7, NOW(), true)

```

```

        RETURNING id`,
        [userId, title, description, salary_range, employment_type,
remote_available, location]
    );

    const vacancyId = result.rows[0].id;

    if (skill_ids && skill_ids.length > 0) {
        for (let skill_id of skill_ids) {
            await db.query(
                `INSERT INTO vacancy_skills (vacancy_id, skill_id) VALUES ($1,
$2)`,
                [vacancyId, skill_id]
            );
        }
    }

    res.status(201).json({ message: "Вакансия создана", id: vacancyId });
} catch (err) {
    console.error("Ошибка создания вакансии:", err);
    res.status(500).json({ message: "Ошибка сервера" });
}
};

exports.applyToVacancy = async (req, res) => {
    const studentId = req.user.id;
    const { vacancy_id } = req.body;

    try {
        // Проверка: студент существует?
        const check = await db.query(
            `SELECT * FROM students WHERE user_id = $1`,
            [studentId]
        );

        if (check.rowCount === 0) {
            return res.status(403).json({ message: "Вы не студент" });
        }

        // Уже откликнулся?
        const exists = await db.query(
            `SELECT * FROM applications WHERE vacancy_id = $1 AND student_id =
$2`,
            [vacancy_id, studentId]
        );

        if (exists.rowCount > 0) {
            return res.status(400).json({ message: "Вы уже откликнулись на эту
вакансию" });
        }

        // Отклик
        await db.query(
            `INSERT INTO applications (vacancy_id, student_id, status,
applied_at)
            VALUES ($1, $2, 'pending', NOW())`,
            [vacancy_id, studentId]
        );
    }
};

```

```

    res.status(201).json({ message: "📧 Вы успешно откликнулись!" });
  } catch (err) {
    console.error("Отклик ошибка:", err);
    res.status(500).json({ message: "Отклик не отправлен" });
  }
};

exports.getRecommendedCourses = async (req, res) => {
  const { vacancyId, studentId } = req.params;

  try {
    const result = await db.query(
      `
      SELECT DISTINCT c.*
      FROM courses c
      JOIN course_skills cs ON cs.course_id = c.id
      JOIN vacancy_skills vs ON cs.skill_id = vs.skill_id
      WHERE vs.vacancy_id = $1
      AND cs.skill_id NOT IN (
        SELECT skill_id FROM student_skills
        WHERE student_id = $2
      )
      LIMIT 10
      `,
      [vacancyId, studentId]
    );

    res.json(result.rows);
  } catch (err) {
    console.error("Ошибка получения рекомендованных курсов:", err);
    res.status(500).json({ message: "Ошибка сервера" });
  }
};

exports.getApplicationsByVacancy = async (req, res) => {
  const vacancyId = req.params.id;
  const userId = req.user.id;

  try {
    // проверяем, что эта вакансия принадлежит работодателю
    const verify = await db.query(
      `SELECT * FROM vacancies WHERE id = $1 AND employer_id = $2`,
      [vacancyId, userId]
    );

    if (verify.rowCount === 0) {
      return res.status(403).json({ message: "Вы не владелец этой вакансии" });
    }

    const result = await db.query(
      `
      SELECT a.*, u.username, u.email
      FROM applications a
      JOIN users u ON a.student_id = u.id
      WHERE a.vacancy_id = $1
      ORDER BY a.applied_at DESC
      `,
      [vacancyId]
    );
  }
};

```

```

        res.json(result.rows);
    } catch (err) {
        console.error("Ошибка получения откликов:", err);
        res.status(500).json({ message: "Ошибка на сервере" });
    }
};

exports.getMyVacancies = async (req, res) => {
    const employerId = req.user.id;

    try {
        const result = await db.query(
            `SELECT * FROM vacancies WHERE employer_id = $1 ORDER BY created_at
DESC`,
            [employerId]
        );
        res.json(result.rows);
    } catch (err) {
        res.status(500).json({ message: "Ошибка получения вакансий" });
    }
};

```

server/controllers/certificate.controller.js:

```
const db = require("../config/db");
```

```

exports.getMyCertificates = async (req, res) => {
    try {
        const userId = req.user.id;

        const result = await db.query(`
            SELECT cert.*, c.title AS course_title
            FROM certificates cert
            JOIN courses c ON cert.course_id = c.id
            WHERE cert.user_id = $1
            ORDER BY cert.issue_date DESC
        `, [userId]);

        res.json(result.rows);
    } catch (err) {
        console.error("Ошибка получения сертификатов:", err);
        res.status(500).json({ message: "Ошибка сервера" });
    }
};

```

server/controllers/auth.controller.js:

```

const UserService = require("../services/user.service");
const { comparePassword } = require("../utils/password");
const { generateToken } = require("../utils/jwt");
const db = require("../config/db");

exports.register = async (req, res) => {
    try {
        const { username, email, password, role } = req.body;

```

```

    if (!['student', 'teacher', 'employer'].includes(role)) {
      return res.status(400).json({ message: "Недопустимая роль" });
    }

    const existing = await UserService.getUserByEmail(email);
    if (existing) {
      return res.status(400).json({ message: "Пользователь уже существует" });
    }

    const user = await UserService.createUser({ username, email, password });

    // Добавим в таблицу роли
    if (role === 'student') {
      await db.query(`INSERT INTO students (user_id) VALUES (${1})`,
[user.id]);
    }

    if (role === 'teacher') {
      await db.query(
`INSERT INTO teachers (user_id, department, is_verified) VALUES
($1, '', false)`,
[user.id]
);
    }

    if (role === 'employer') {
      await db.query(
`INSERT INTO employers (user_id, company_name, verified) VALUES
($1, '', false)`,
[user.id]
);
    }

    // 📌 также добавляй в teachers, чтобы соблюсти FK
    await db.query(
`INSERT INTO teachers (user_id, department, academic_degree,
office_number, is_verified)
VALUES (${1}, '', '', '', false)`,
[user.id]
);

    const token = generateToken(user);

    res.status(201).json({ user, token });
  } catch (err) {
    console.error("Registration error:", err);
    res.status(500).json({ message: "Ошибка регистрации" });
  }
};

exports.login = async (req, res) => {
  try {
    const { email, password } = req.body;

    const user = await UserService.getUserByEmail(email);

```

```

    if (!user) {
      return res.status(404).json({ message: "Пользователь не найден" });
    }

    // ☒ Добавляем проверку блокировки
    if (user.is_blocked) {
      return res.status(403).json({ message: "Аккаунт заблокирован" });
    }

    const match = await comparePassword(password, user.password_hash);

    if (!match) {
      return res.status(401).json({ message: "Неверный пароль" });
    }

    const token = generateToken(user);
    const { password_hash, ...userData } = user;

    res.json({ user: userData, token });

  } catch (err) {
    console.error("Login error:", err);
    res.status(500).json({ message: "Ошибка авторизации" });
  }
};

server/controllers/category.controller.js:

const db = require('../config/db');

exports.getAllCategories = async (req, res) => {
  try {
    const result = await db.query(`SELECT * FROM categories ORDER BY
title`);
    res.json(result.rows);
  } catch (err) {
    console.error("Ошибка загрузки категорий:", err);
    res.status(500).json({ message: "Ошибка сервера" });
  }
};

exports.getAllSkills = async (req, res) => {
  try {
    const result = await db.query(`SELECT id, name FROM skills ORDER BY
name`);
    res.json(result.rows);
  } catch (err) {
    console.error("Ошибка загрузки навыков:", err);
    res.status(500).json({ message: "Ошибка сервера" });
  }
};

server/controllers/submission.controller.js:

const db = require("../config/db");

```

```

exports.getSubmissionsByCourse = async (req, res) => {
  const courseId = req.params.courseId;
  const userId = req.user.id;

  // Проверка: только автор курса может просматривать
  const check = await db.query(
    `SELECT * FROM courses WHERE id = $1 AND teacher_id = $2`,
    [courseId, userId]
  );

  if (check.rowCount === 0) {
    return res.status(403).json({ message: "Нет доступа: вы не автор курса"
  });
  }

  const submissions = await db.query(
    `SELECT s.*, u.username FROM student_submissions s
    JOIN users u ON s.student_id = u.id
    WHERE s.course_id = $1
    ORDER BY submitted_at DESC`,
    [courseId]
  );

  res.json(submissions.rows);
};

exports.approveSubmission = async (req, res) => {
  const submissionId = req.params.id;
  const userId = req.user.id;

  try {
    // Проверка авторства (безопасность)
    const submission = await db.query(
      `SELECT s.*, c.teacher_id
      FROM student_submissions s
      JOIN courses c ON c.id = s.course_id
      WHERE s.id = $1
      `, [submissionId]);

    if (submission.rowCount === 0) {
      return res.status(404).json({ message: "Работа не найдена" });
    }

    const row = submission.rows[0];
    if (row.teacher_id !== userId) {
      return res.status(403).json({ message: "Доступ запрещен. Вы не автор
курса" });
    }

    await db.query(
      `UPDATE student_submissions SET is_approved = true
      WHERE id = $1
      `, [submissionId]);

    res.json({ message: "☑ Работа одобрена" });
  } catch (err) {
    console.error("Ошибка approve:", err);
  }
};

```

```

        res.status(500).json({ message: "Ошибка при одобрении" });
    }
};

server/controllers/enrollment.controller.js:

const db = require("../config/db");

exports.enroll = async (req, res) => {
    try {
        const user = req.user;
        const { course_id } = req.body;

        const enrollment = await
require("../models/enrollment.model").enrollInCourse(user.id, course_id);
        res.status(201).json(enrollment);
    } catch (err) {
        if (err.code === "23505") {
            return res.status(400).json({ message: "Вы уже записаны на этот курс"
});
        }
        res.status(500).json({ message: "Ошибка записи на курс" });
    }
};

exports.list = async (req, res) => {
    try {
        const user = req.user;
        const enrollments = await
require("../models/enrollment.model").getUserEnrollments(user.id);
        res.json(enrollments);
    } catch (err) {
        res.status(500).json({ message: "Ошибка получения записей" });
    }
};

exports.updateProgress = async (req, res) => {
    try {
        const { is_completed } = req.body;
        const { id } = req.params;

        const enrollment = await
require("../models/enrollment.model").updateProgress(id, is_completed);
        res.json(enrollment);
    } catch (err) {
        res.status(400).json({ message: err.message || "Ошибка обновления
прогресса" });
    }
};

// ☒ Загрузка итоговой работы
exports.submitWork = async (req, res) => {
    try {
        const studentId = req.user.id;
        const courseId = req.params.courseId;
        const file = req.file;

```

```

    if (!file) {
      return res.status(400).json({ message: "Файл не загружен" });
    }

    await db.query(
      `INSERT INTO student_submissions (student_id, course_id, file_url,
original_name)
      VALUES ($1, $2, $3, $4)
      ON CONFLICT (student_id, course_id)
      DO UPDATE SET file_url = $3, original_name = $4, submitted_at =
NOW(), is_approved = false`,
      [studentId, courseId, file.filename, file.originalname]
    );

    res.json({ message: "✅ Работа отправлена на проверку!" });
  } catch (err) {
    console.error("Ошибка отправки работы:", err);
    res.status(500).json({ message: "Ошибка при загрузке" });
  }
};

// ✅ Отписка от курса (если не завершён)
exports.unenroll = async (req, res) => {
  try {
    const userId = req.user.id;
    const courseId = req.params.courseId;

    const check = await db.query(
      `SELECT is_completed FROM enrollments
      WHERE user_id = $1 AND course_id = $2`,
      [userId, courseId]
    );

    if (check.rows[0]?.is_completed) {
      return res.status(400).json({ message: "Нельзя отписаться - курс уже
завершён" });
    }

    await db.query(
      `DELETE FROM enrollments WHERE user_id = $1 AND course_id = $2`,
      [userId, courseId]
    );

    res.json({ message: "🚫 Вы успешно отписались от курса" });
  } catch (err) {
    console.error("Ошибка отписки:", err);
    res.status(500).json({ message: "Ошибка при отписке" });
  }
};

server/controllers/admin.controller.js:

const AdminService = require("../services/admin.service");

exports.getPlatformStats = async (req, res) => {
  try {
    const stats = await AdminService.getStats();

```

```

        res.json(stats);
    } catch (err) {
        res.status(500).json({ message: "Ошибка получения статистики" });
    }
};

exports.getAllUsers = async (req, res) => {
    try {
        const users = await AdminService.getAllUsers();
        res.json(users);
    } catch (err) {
        res.status(500).json({ message: "Ошибка получения пользователей" });
    }
};

exports.toggleBlock = async (req, res) => {
    try {
        const user = await AdminService.toggleBlock(req.params.id);
        res.json(user);
    } catch (err) {
        res.status(500).json({ message: "Ошибка блокировки" });
    }
};

exports.changeRole = async (req, res) => {
    try {
        const { role } = req.body;
        const result = await AdminService.changeRole(req.params.id, role);

        if (!result.success) {
            return res.status(400).json({ message: result.message });
        }

        res.json({ message: "Роль обновлена" });
    } catch (err) {
        console.error(err);
        res.status(500).json({ message: "Ошибка смены роли" });
    }
};

```

client/package.json:

```

{
  "name": "client",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/dom": "^10.4.0",
    "@testing-library/jest-dom": "^6.6.3",
    "@testing-library/react": "^16.3.0",
    "@testing-library/user-event": "^13.5.0",
    "axios": "^1.9.0",
    "bootstrap": "^5.3.6",
    "react": "^19.1.0",
    "react-dom": "^19.1.0",
    "react-router-dom": "^7.6.2",
    "react-scripts": "5.0.1",

```

```

    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}

```

client/public/manifest.json:

```

{
  "short_name": "React App",
  "name": "Create React App Sample",
  "icons": [
    {
      "src": "favicon.ico",
      "sizes": "64x64 32x32 24x24 16x16",
      "type": "image/x-icon"
    },
    {
      "src": "logo192.png",
      "type": "image/png",
      "sizes": "192x192"
    },
    {
      "src": "logo512.png",
      "type": "image/png",
      "sizes": "512x512"
    }
  ],
  "start_url": ".",
  "display": "standalone",
  "theme_color": "#000000",
  "background_color": "#ffffff"
}

```

```

client/public/index.html:

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed
on a      user's      mobile      device      or      desktop.      See
https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the
build.      Only files inside the `public` folder can be referenced from the HTML.

      Unlike `"/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico"
will      work correctly both with client-side routing and a non-root public
URL.      Learn how to configure a non-root public URL by running `npm run
build`.
    -->
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run `npm start` or `yarn start`.
      To create a production bundle, use `npm run build` or `yarn build`.
    -->
  </body>
</html>

client/src/index.js:

```

```

import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';

import 'bootstrap/dist/css/bootstrap.min.css'; // если используешь
bootstrap

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);

client/src/index.css:

client/src/api.js:

import axios from 'axios';

const API = axios.create({
  baseURL: 'http://localhost:5000/api',
});

// авторизация автоматическая (если токен есть)
API.interceptors.request.use((req) => {
  const token = localStorage.getItem("token");
  if (token) req.headers.Authorization = `Bearer ${token}`;
  return req;
});

export const FILE_BASE = 'http://localhost:5000';
export default API;

client/src/App.js:

import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import Navbar from './components/Navbar';
import HomePage from './pages/HomePage';
import LoginPage from './pages/LoginPage';
import RegisterPage from './pages/RegisterPage';
import CoursesPage from './pages/CoursesPage';
import DashboardPage from './pages/DashboardPage';
import PrivateRoute from './components/PrivateRoute';
import CreateCoursePage from './pages/CreateCoursePage';
import MyCertificatesPage from './pages/MyCertificatesPage';
import AdminPanel from './pages/AdminPanel';
import CourseDetailPage from './pages/CourseDetailPage';
import RecommendationsPage from './pages/RecommendationsPage';
import CreateVacancyPage from './pages/CreateVacancyPage';
import VacanciesPage from './pages/VacanciesPage';

function App() {
  return (
    <Router>
      <Navbar />

```

```

<Routes>
  <Route path="/" element={<HomePage />} />
  <Route path="/courses" element={<CoursesPage />} />
  <Route path="/login" element={<LoginPage />} />
  <Route path="/register" element={<RegisterPage />} />
  <Route
    path="/dashboard"
    element={
      <PrivateRoute>
        <DashboardPage />
      </PrivateRoute>
    }
  />
  <Route
    path="/create-course"
    element={
      <PrivateRoute>
        <CreateCoursePage />
      </PrivateRoute>
    }
  />
  <Route
    path="/certificates"
    element={
      <PrivateRoute>
        <MyCertificatesPage />
      </PrivateRoute>
    }
  />
  <Route path="/admin" element={
    <PrivateRoute>
      <AdminPanel />
    </PrivateRoute>
  }
  />
  <Route path="/courses/:id" element={
    <CourseDetailPage />
  }
  />
  <Route path="/recommendations" element={
    <PrivateRoute>
      <RecommendationsPage />
    </PrivateRoute>
  }
  />
  <Route path="/create-vacancy" element={
    <PrivateRoute>
      <CreateVacancyPage />
    </PrivateRoute>
  }
  />
  <Route path="/vacancies" element={
    <PrivateRoute>
      <VacanciesPage />
    </PrivateRoute>
  }
  />
</Routes>

```

```

    </Router>
  );
}

```

```
export default App;
```

```
client/src/auth.js:
```

```

export const setToken = (token) => {
  localStorage.setItem("token", token);
};

export const getToken = () => {
  return localStorage.getItem("token");
};

export const removeToken = () => {
  localStorage.removeItem("token");
};

export const isAuthenticated = () => {
  return !!getToken();
};

export const getRoles = () => {
  try {
    return JSON.parse(localStorage.getItem("roles")) || [];
  } catch {
    return [];
  }
};

export const hasRole = (role) => {
  const roles = getRoles();
  return roles.includes(role);
};

```

```
client/src/pages/AdminPanel.js:
```

```

import React, { useEffect, useState } from "react";
import API from "../api";
import { hasRole } from "../auth";

const AdminPanel = () => {
  const [users, setUsers] = useState([]);
  const [stats, setStats] = useState(null);
  const [message, setMessage] = useState("");

  useEffect(() => {
    if (!hasRole("admin")) return;

    const fetchData = async () => {
      try {
        const usersRes = await API.get("/admin/users");
        const statsRes = await API.get("/admin/stats");
        setUsers(usersRes.data);

```

```

        setStats(statsRes.data);
      } catch (err) {
        console.error(err);
        setMessage("❌ Ошибка загрузки данных администратора");
      }
    };

    fetchData();
  }, []);

  const toggleBlock = async (id) => {
    try {
      const res = await API.put(`/admin/users/${id}/block`);
      setMessage(`Пользователь ${res.data.username} ${res.data.is_blocked ?
"заблокирован" : "разблокирован"}`);
      setUsers((prev) =>
        prev.map((u) => (u.id === id ? { ...u, is_blocked: !u.is_blocked }
: u))
      );
    } catch (err) {
      console.error(err);
      setMessage("❌ Не удалось изменить статус");
    }
  };

  const changeRole = async (id, role) => {
    try {
      await API.put(`/admin/users/${id}/role`, { role });
      setMessage(`🔄 Роль пользователя обновлена на "${role}"`);
    } catch (err) {
      console.error(err);
      setMessage("❌ Ошибка при смене роли");
    }
  };

  if (!hasRole("admin")) {
    return (
      <div className="container mt-4">
        <div className="alert alert-danger">🚫 Доступ только для
администраторов</div>
      </div>
    );
  }

  return (
    <div className="container mt-4">
      <h2 className="mb-4">⚙️ Панель администратора</h2>

      {message && (
        <div
          className={`alert ${
            message.includes("Ошибка") || message.includes("❌")
              ? "alert-danger"
              : "alert-info"
          }`}
        >
          {message}
        </div>
      )}
    </div>
  );

```

```

    </div>
  })

  { /* --- Статистика --- */
  {stats && (
    <div className="mb-4">
      <h4><img alt="Bar chart icon" data-bbox="345 193 365 208"/> Статистика платформы</h4>
      <div className="row">
        <div className="col-sm-6 col-md-3">
          <div className="card text-bg-light mb-3">
            <div className="card-header"><img alt="User icon" data-bbox="658 250 678 265"/> Пользователи</div>
            <div className="card-body">
              <h5 className="card-title">{stats.users}</h5>
            </div>
          </div>
        </div>
        <div className="col-sm-6 col-md-3">
          <div className="card text-bg-light mb-3">
            <div className="card-header"><img alt="Graduation cap icon" data-bbox="658 360 678 375"/> Студенты</div>
            <div className="card-body">
              <h5 className="card-title">{stats.students}</h5>
            </div>
          </div>
        </div>
        <div className="col-sm-6 col-md-3">
          <div className="card text-bg-light mb-3">
            <div className="card-header"><img alt="Book icon" data-bbox="658 470 678 485"/> Курсы</div>
            <div className="card-body">
              <h5 className="card-title">{stats.courses}</h5>
            </div>
          </div>
        </div>
        <div className="col-sm-6 col-md-3">
          <div className="card text-bg-light mb-3">
            <div className="card-header"><img alt="Checkmark icon" data-bbox="658 580 678 595"/> Завершено курсов</div>
            <div className="card-body">
              <h5 className="card-title">{stats.completedCourses}</h5>
            </div>
          </div>
        </div>
      </div>
    </div>
  })

  { /* --- Таблица по пользователям --- */
  <div className="mb-4">
    <h4><img alt="User icon" data-bbox="325 745 345 760"/> Управление пользователями</h4>
    {users.length === 0 ? (
      <div className="alert alert-info">Пользователи не найдены</div>
    ) : (
      <div className="table-responsive">
        <table className="table table-bordered table-hover align-
middle">
          <thead className="table-dark">
            <tr>
              <th>ID</th>
              <th>Имя</th>

```

```

        <th>Email</th>
        <th>Статус</th>
        <th>🔒 Блокировка</th>
        <th>🔄 Сменить роль</th>
      </tr>
    </thead>
    <tbody>
      {users.map((u) => (
        <tr key={u.id}>
          <td>{u.id}</td>
          <td>{u.username}</td>
          <td>{u.email}</td>
          <td>
            {u.is_blocked ? (
              <span      className="badge      text-bg-danger">🛑
Заблокирован</span>
            ) : (
              <span      className="badge      text-bg-success">✅
Активен</span>
            )}
          </td>
          <td>
            <button
              className={`btn btn-sm ${
                u.is_blocked ? "btn-success" : "btn-danger"
              }`}
              onClick={() => toggleBlock(u.id)}
            >
              {u.is_blocked ? "Разблокировать" : "Заблокировать"}
            </button>
          </td>
          <td>
            <select
              className="form-select form-select-sm"
              defaultValue=""
              onChange={(e) => changeRole(u.id, e.target.value)}
            >
              <option value="">-- выбрать --</option>
              <option value="student">Студент</option>
              <option value="teacher">Преподаватель</option>
              <option value="employer">Работодатель</option>
            </select>
          </td>
        </tr>
      )})
    </tbody>
  </table>
</div>
)}
</div>
</div>
);
};

export default AdminPanel;

```

```

client/src/pages/MyCertificatesPage.js:

import React, { useEffect, useState } from 'react';
import API from '../api';
import { hasRole } from '../auth';

const MyCertificatesPage = () => {
  const [certs, setCerts] = useState([]);

  useEffect(() => {
    const fetch = async () => {
      const res = await API.get('/certificates');
      setCerts(res.data);
    };

    fetch();
  }, []);

  if (!hasRole('student')) {
    return <div className="container"><p className="alert alert-danger mt-4">🚫 Только студенты могут просматривать сертификаты</p></div>;
  }

  return (
    <div className="container mt-4">
      <h2>🎓 Мои сертификаты</h2>

      {certs.length === 0 ? (
        <div className="alert alert-info mt-3">Сертификатов пока нет.</div>
      ) : (
        <table className="table table-striped mt-3">
          <thead className="table-dark">
            <tr>
              <th>📖 Курс</th>
              <th>📅 Дата</th>
              <th>🆔 Код</th>
              <th>🔗 Ссылка</th>
            </tr>
          </thead>
          <tbody>
            {certs.map(cert => (
              <tr key={cert.id}>
                <td>{cert.course_title}</td>
                <td>{new Date(cert.issue_date).toLocaleDateString()}</td>
                <td><code>{cert.verification_code}</code></td>
                <td>
                  <a href={cert.certificate_url} target="_blank"
rel="noreferrer">
                    Открыть
                  </a>
                </td>
              </tr>
            ))}
          </tbody>
        </table>
      )}
    </div>
  )
}

```

```

    );
  };

export default MyCertificatesPage;

client/src/pages/CreateVacancyPage.js:

import React, { useState, useEffect } from 'react';
import API from '../api';
import { hasRole } from '../auth';
import { useNavigate } from 'react-router-dom';

const CreateVacancyPage = () => {
  const navigate = useNavigate();

  const [form, setForm] = useState({
    title: '',
    description: '',
    salary_range: '',
    employment_type: 'full-time',
    remote_available: false,
    location: '',
  });

  const [skills, setSkills] = useState([]); // Все навыки
  const [selectedSkills, setSelectedSkills] = useState([]); // Выбранные ID
  НАВЫКОВ
  const [message, setMessage] = useState('');

  // Загрузим список навыков
  useEffect(() => {
    const fetchSkills = async () => {
      try {
        const res = await API.get('/categories/skills'); // предполагаемый
        маршрут
        setSkills(res.data || []);
      } catch (err) {
        console.error('Ошибка загрузки навыков', err);
      }
    };

    fetchSkills();
  }, []);

  const handleChange = (e) => {
    const { name, value, type, checked } = e.target;
    setForm({
      ...form,
      [name]: type === 'checkbox' ? checked : value,
    });
  };

  const handleSkillsChange = (e) => {
    const options = [...e.target.selectedOptions];
    const values = options.map(o => parseInt(o.value));
    setSelectedSkills(values);
  };

```

```

const handleSubmit = async (e) => {
  e.preventDefault();
  setMessage('');

  const payload = {
    ...form,
    skill_ids: selectedSkills,
  };

  try {
    await API.post('/vacancies', payload);
    setMessage('✅ Вакансия успешно создана!');
    setTimeout(() => navigate('/dashboard'), 2000);
  } catch (err) {
    console.error(err);
    setMessage('❌ Ошибка при создании вакансии');
  }
};

if (!hasRole('employer')) {
  return <p className="text-danger">⛔ Доступ только для
работодателя</p>;
}

return (
  <div className="container mt-4">
    <h2 className="mb-4">📄 Создание вакансии</h2>

    {message && (
      <div
        className={`alert ${message.startsWith('✅') ? 'alert-success' :
'alert-danger'}`}`
      >
        {message}
      </div>
    )}

    <form onSubmit={handleSubmit}>
      <div className="mb-3">
        <label>Название должности:</label>
        <input
          type="text"
          className="form-control"
          name="title"
          value={form.title}
          onChange={handleChange}
          required
        />
      </div>

      <div className="mb-3">
        <label>Описание работы:</label>
        <textarea
          className="form-control"
          name="description"
          rows="4"

```

```

        value={form.description}
        onChange={handleChange}
        required
      />
    </div>

    <div className="mb-3">
      <label>Уровень зарплаты:</label>
      <input
        type="text"
        className="form-control"
        name="salary_range"
        value={form.salary_range}
        onChange={handleChange}
        placeholder="например: 80k-120k"
        required
      />
    </div>

    <div className="mb-3">
      <label>Тип занятости:</label>
      <select
        className="form-select"
        name="employment_type"
        value={form.employment_type}
        onChange={handleChange}
      >
        <option value="full-time">Полная занятость</option>
        <option value="part-time">Частичная</option>
        <option value="contract">Контракт</option>
        <option value="internship">Стажировка</option>
      </select>
    </div>

    <div className="form-check mb-3">
      <input
        className="form-check-input"
        type="checkbox"
        name="remote_available"
        id="remoteAvailable"
        checked={form.remote_available}
        onChange={handleChange}
      />
      <label className="form-check-label" htmlFor="remoteAvailable">
        Удалённая работа возможна
      </label>
    </div>

    <div className="mb-3">
      <label>Локация / город:</label>
      <input
        type="text"
        className="form-control"
        name="location"
        value={form.location}
        onChange={handleChange}
        required
      />

```

```

    </div>

    <div className="mb-3">
      <label>Необходимые навыки:</label>
      <select
        className="form-select"
        multiple
        value={selectedSkills}
        onChange={handleSkillsChange}
        size="5"
      >
        {skills.map((skill) => (
          <option key={skill.id} value={skill.id}>
            {skill.name}
          </option>
        ))}
      </select>
      <div className="form-text">Удерживайте Ctrl (или Cmd) для выбора
нескольких.</div>
    </div>

    <button type="submit" className="btn btn-primary">
       Создать вакансию
    </button>
  </form>
</div>
);
};

export default CreateVacancyPage;

client/src/pages/CreateCoursePage.js:

import React, { useState } from 'react';
import { hasRole } from '../auth';
import API from '../api';
import { useNavigate } from 'react-router-dom';

const CreateCoursePage = () => {
  const navigate = useNavigate();

  const [form, setForm] = useState({
    title: '',
    description: '',
    difficulty: 'beginner'
  });

  const [message, setMessage] = useState('');

  const hasAccess = hasRole('teacher') || hasRole('employer');

  if (!hasAccess) {
    return <div className="container mt-4"><p className="alert alert-
danger">🚫 У вас нет доступа к созданию курсов</p></div>;
  }

  const handleChange = (e) => {

```

```

    setForm({ ...form, [e.target.name]: e.target.value });
  };

const handleSubmit = async (e) => {
  e.preventDefault();
  setMessage('');

  try {
    await API.post('/courses', form);
    setMessage('✅ Курс успешно создан!');
    setForm({ title: '', description: '', difficulty: 'beginner' });

    setTimeout(() => navigate('/courses'), 1500);
  } catch (err) {
    console.error(err);
    setMessage('❌ Ошибка при создании курса');
  }
};

return (
  <div className="container my-4">
    <h2>📚 Создание нового курса</h2>

    {message && (
      <div className={`alert ${message.startsWith("✅") ? "alert-success"
: "alert-danger"}`}>
        {message}
      </div>
    )}

    <form onSubmit={handleSubmit}>
      {/* Название */}
      <div className="mb-3">
        <label htmlFor="title" className="form-label">Название
курса:</label>
        <input
          type="text"
          className="form-control"
          id="title"
          name="title"
          value={form.title}
          onChange={handleChange}
          required
        />
      </div>

      {/* Описание */}
      <div className="mb-3">
        <label htmlFor="description" className="form-
label">Описание:</label>
        <textarea
          className="form-control"
          id="description"
          name="description"
          rows="5"
          value={form.description}
          onChange={handleChange}

```

```

        required
      />
    </div>

    {/** Сложность */}
    <div className="mb-4">
      <label      htmlFor="difficulty"      className="form-label">Уровень
сложности:</label>
      <select
        className="form-select"
        id="difficulty"
        name="difficulty"
        value={form.difficulty}
        onChange={handleChange}
      >
        <option value="beginner">🧐 Начальный</option>
        <option value="intermediate">😬 Средний</option>
        <option value="advanced">🧠 Продвинутый</option>
      </select>
    </div>

    <button type="submit" className="btn btn-primary">📄 Создать
курс</button>
  </form>
</div>
);
};

export default CreateCoursePage;

client/src/pages/HomePage.js:

import React from 'react';
import { Link } from 'react-router-dom';
import { isAuthenticated } from '../auth';

const HomePage = () => {
  const startLink = isAuthenticated() ? "/dashboard" : "/register";

  return (
    <div className="container py-5">
      {/** Hero Section */}
      <div className="text-center mb-5">
        <h1 className="display-4 fw-bold">
          🏠 Добро пожаловать на <span      className="text-
primary">Образовательный Портал</span>
        </h1>
        <p className="lead text-muted mx-auto" style={{ maxWidth: '800px'
}}>
          Интерактивная платформа, где <strong>студенты</strong>,
<strong>преподаватели</strong> и <strong>работодатели</strong> объединяются через
совместное обучение, развитие навыков и карьерные возможности.
        </p>

        {/** CTA Buttons */}
        <div className="mt-4">

```

```

        <Link to={startLink} className="btn btn-success btn-lg me-3">
            🚀 Начать обучение
        </Link>
        <Link to="/courses" className="btn btn-outline-primary btn-lg">
            📖 Просмотреть курсы
        </Link>
    </div>
</div>

{/* Features */}
<div className="row text-center g-4">
    <div className="col-md-4">
        <div className="card border-0 shadow-sm h-100">
            <div className="card-body">
                <h4 className="card-title">👤 Гибкое обучение</h4>
                <p className="card-text text-muted">
                    Записывайся на курсы в любое время, занимайся в удобном
                    темпе и получай сертификаты.
                </p>
            </div>
        </div>
    </div>
    <div className="col-md-4">
        <div className="card border-0 shadow-sm h-100">
            <div className="card-body">
                <h4 className="card-title">👛 Курсы от работодателей</h4>
                <p className="card-text text-muted">
                    Осваивай реальные технологии и навыки, нужные в компаниях и
                    на рынке труда прямо сейчас.
                </p>
            </div>
        </div>
    </div>
    <div className="col-md-4">
        <div className="card border-0 shadow-sm h-100">
            <div className="card-body">
                <h4 className="card-title">✅ Персональные рекомендации</h4>
                <p className="card-text text-muted">
                    Мы анализируем твои успехи, чтобы предложить подходящие
                    курсы, вакансии и карьерные траектории.
                </p>
            </div>
        </div>
    </div>
</div>

{/* Footer-style block */}
<div className="mt-5 text-center">
    <p className="text-muted">
        🌐 Доступно для студентов, преподавателей, работодателей и
        администраторов любого уровня.
    </p>
    <small className="text-secondary">© {new Date().getFullYear()}
    Образовательный Портал. Все права защищены.</small>
</div>
</div>
);

```

```

};

export default HomePage;

client/src/pages/DashboardPage.js:

import React, { useEffect, useState } from 'react';
import { useNavigate, Link } from 'react-router-dom';
import API from '../api';
import { hasRole } from '../auth';

const DashboardPage = () => {
  const [profile, setProfile] = useState(null);
  const [enrollments, setEnrollments] = useState([]);
  const [myCourses, setMyCourses] = useState([]);
  const [myVacancies, setMyVacancies] = useState([]);
  const [applications, setApplications] = useState({});
  const navigate = useNavigate();

  useEffect(() => {
    const getProfile = async () => {
      const res = await API.get('/users/me');
      setProfile(res.data);
      localStorage.setItem('roles', JSON.stringify(res.data.roles || []));
    };
    getProfile();
  }, []);

  useEffect(() => {
    if (hasRole('student')) {
      API.get('/enrollments').then(res => setEnrollments(res.data));
    }

    if (hasRole('teacher') || hasRole('employer')) {
      API.get('/courses/my').then(res => setMyCourses(res.data));
    }

    if (hasRole('employer')) {
      const fetchVacancies = async () => {
        const res = await API.get('/vacancies/my');
        setMyVacancies(res.data);

        for (let v of res.data) {
          const appRes = await API.get(`/vacancies/${v.id}/applications`);
          setApplications(prev => ({
            ...prev,
            [v.id]: appRes.data,
          }));
        }
      };
      fetchVacancies();
    }
  }, []);

  const handleComplete = async (id) => {
    try {
      await API.put(`/enrollments/${id}/progress`, { is_completed: true });
    }
  }
}

```

```

        alert("✅ Курс завершён!");
        window.location.reload();
    } catch (err) {
        console.error(err);
        alert("❌ Ошибка завершения");
    }
};

if (!profile) return <div className="text-center mt-5">Загрузка
профиля...</div>;

return (
    <div className="container my-4">
        <h2>👤 Личный кабинет</h2>
        <div className="mb-4">
            <p>
                <strong>{profile.username}</strong> - Роли:&nbsp;
                <span className="badge bg-info text-dark">{profile.roles.join(',
' )}</span>
            </p>
        </div>

        { (hasRole('teacher') || hasRole('employer')) && (
            <>
                <div className="mb-3">
                    <button className="btn btn-primary" onClick={() =>
navigate('/create-course')}>
                        + Создать курс
                    </button>
                </div>

                <h4>📚 Мои курсы</h4>
                {myCourses.length === 0 ? (
                    <div className="alert alert-warning">У вас пока нет созданных
курсов.</div>
                ) : (
                    <ul className="list-group mb-4">
                        {myCourses.map(c => (
                            <li key={c.id} className="list-group-item d-flex justify-
content-between align-items-center">
                                <Link to={` /courses/${c.id}`}>{c.title}</Link>
                                <span
                                    className="badge
secondary">{c.difficulty}</span>
                            </li>
                        ))}
                    </ul>
                )}
            </>
        )}

        {hasRole('student') && (
            <>
                <div className="mb-3">
                    <button className="btn btn-outline-success" onClick={() =>
navigate('/certificates')}>
                        🏠 Посмотреть сертификаты
                    </button>

```

```

    </div>
    <h4>📚 Курсы студента</h4>
    {enrollments.length === 0 ? (
      <div className="alert alert-warning">Вы пока не записаны ни на
    один курс.</div>
    ) : (
      <ul className="list-group mb-4">
        {enrollments.map(e => (
          <li key={e.id} className="list-group-item">
            <div className="d-flex justify-content-between align-items-center">
              <Link to={` /courses/${e.course_id}`} className="fw-bold">{e.title}</Link>
              <div>
                {e.is_completed ? (
                  <span className="badge bg-success">✅
                Завершен</span>
                ) : (
                  <span className="badge bg-warning text-dark me-2">⌚ В процессе</span>
                  <button
                    onClick={() => handleComplete(e.id)}
                    className="btn btn-sm btn-outline-success"
                  >
                    🚩 Завершить
                  </button>
                </>
              </div>
            </div>
          </li>
        )
        )
        )
      </ul>
    )
  }
}

{hasRole('employer') && (
  <h4>👛 Мои вакансии и отклики</h4>
  {myVacancies.length === 0 ? (
    <div className="alert alert-warning">Нет созданных
    вакансий</div>
  ) : (
    myVacancies.map(v => (
      <div key={v.id} className="card mb-4 border-secondary">
        <div className="card-body">
          <h5 className="card-title">{v.title}</h5>
          <p className="card-text">{v.description}</p>

          {(applications[v.id] || []).length === 0 ? (
            <div className="alert alert-light">Нет откликов</div>
          ) : (
            <div className="table-responsive mt-3">
              <table className="table table-bordered table-sm">
                <thead className="table-secondary">

```

```

        <tr>
          <th>👤 Имя</th>
          <th>✉ Email</th>
          <th>📅 Дата</th>
          <th>Статус</th>
        </tr>
      </thead>
      <tbody>
        {applications[v.id].map(app => (
          <tr key={app.id}>
            <td>{app.username}</td>
            <td>{app.email}</td>
            <td>{new
Date(app.applied_at).toLocaleDateString()}</td>
            <td>{app.status}</td>
          </tr>
        ))}
      </tbody>
    </table>
  </div>
)}
</div>
</div>
))
)}
</>
)}
</div>
);
};

```

```
export default DashboardPage;
```

```
client/src/pages/RecommendationsPage.js:
```

```

import React, { useEffect, useState } from "react";
import API from "../api";

const RecommendationsPage = () => {
  const [vacancies, setVacancies] = useState([]);
  const [trajectories, setTrajectories] = useState([]);
  const [error, setError] = useState("");

  useEffect(() => {
    const fetch = async () => {
      try {
        const userRes = await API.get("/users/me");
        const userId = userRes.data.id;

        const [vacRes, trajRes] = await Promise.all([
          API.get(`/students/${userId}/recommendations/vacancies`),
          API.get(`/students/${userId}/recommendations/trajectories`)
        ]);

        setVacancies(vacRes.data);
        setTrajectories(trajRes.data);
      } catch (err) {

```

```

        console.error(err);
        setError("Ошибка получения рекомендаций");
    }
};

fetch();
}, []);

return (
    <div className="container mt-4">
        <h2>🔗 Персональные рекомендации</h2>
        {error && <div className="alert alert-danger">{error}</div>}

        <div className="mt-4">
            <h4>📅 Подходящие вакансии</h4>
            {vacancies.length === 0 ? (
                <div className="alert alert-secondary">Нет подходящих
вакансий</div>
            ) : (
                vacancies.map((v) => (
                    <div className="card mb-3" key={v.id}>
                        <div className="card-body">
                            <h5 className="card-title">{v.title}</h5>
                            <p>{v.description}</p>
                            <span className="badge bg-success mb-2">Совпадений по
навыкам: {v.match_count}</span>
                        </div>
                    </div>
                ))
            )}
        </div>

        <div className="mt-5">
            <h4>📈 Карьерные траектории</h4>
            {trajectories.length === 0 ? (
                <div className="alert alert-secondary">Недостаточно данных по
траекториям</div>
            ) : (
                trajectories.map((t) => (
                    <div key={t.id} className="card mb-3">
                        <div className="card-body">
                            <h5>{t.title}</h5>
                            <p>{t.description}</p>
                            <div className="progress">
                                <div className="progress-bar bg-info" role="progressbar"
style={{ width: `${t.progress_percent}%` }}>
                                    {t.progress_percent}%
                                </div>
                            </div>
                        </div>
                    </div>
                ))
            )}
        </div>
    </div>
);
};

```

```

export default RecommendationsPage;

client/src/pages/LoginPage.js:

import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import API from '../api';
import { setToken } from '../auth';

const LoginPage = () => {
  const [form, setForm] = useState({ email: '', password: '' });
  const [error, setError] = useState('');
  const navigate = useNavigate();

  const handleChange = e => {
    setForm({ ...form, [e.target.name]: e.target.value });
    setError('');
  };

  const handleSubmit = async e => {
    e.preventDefault();

    try {
      const res = await API.post('/auth/login', form);
      setToken(res.data.token);
      localStorage.setItem("roles", JSON.stringify(res.data.user.roles ||
[]));
      navigate('/dashboard');
    } catch (err) {
      setError(err.response?.data?.message || 'Ошибка входа');
    }
  };

  return (
    <div className="container mt-4">
      <h2>🔒 Вход в систему</h2>

      {error && <div className="alert alert-danger">{error}</div>}

      <form onSubmit={handleSubmit} className="w-50 mx-auto">
        <div className="mb-3">
          <label>Email</label>
          <input
            name="email"
            type="email"
            className="form-control"
            value={form.email}
            onChange={handleChange}
            required
          />
        </div>

        <div className="mb-3">
          <label>Пароль</label>
          <input
            name="password"
            type="password"

```

```

        className="form-control"
        value={form.password}
        onChange={handleChange}
        required
      />
    </div>

    <button className="btn btn-primary w-100">Войти</button>
  </form>
</div>
);
};

export default LoginPage;

client/src/pages/CoursesPage.js:

import React, { useEffect, useState } from 'react';
import API from '../api';
import { isAuthenticated, hasRole } from '../auth';
import { Link } from 'react-router-dom';

const CoursesPage = () => {
  const [courses, setCourses] = useState([]);
  const [categories, setCategories] = useState([]);
  const [filter, setFilter] = useState({ category: '', difficulty: '' });
  const [message, setMessage] = useState('');

  const fetchCourses = async () => {
    try {
      const res = await API.get('/courses', {
        params: {
          category: filter.category,
          difficulty: filter.difficulty
        }
      });
      setCourses(res.data);
    } catch (err) {
      console.error(err);
    }
  };

  const fetchCategories = async () => {
    try {
      const res = await API.get('/categories');
      setCategories(res.data);
    } catch (err) {
      console.error(err);
    }
  };

  const handleEnroll = async (course_id) => {
    try {
      await API.post('/enrollments', { course_id });
      setMessage('✅ Запись успешна!');
      fetchCourses(); // обновить список (чтобы убрать кнопку "Записаться",
      // если уже записан)
    } catch (err) {
      console.error(err);
    }
  };
};

```

```

    } catch (err) {
      setMessage(err.response?.data?.message || '❌ Ошибка записи');
    }
  };

  useEffect(() => {
    fetchCategories();
  }, []);

  useEffect(() => {
    fetchCourses();
  }, [filter]);

  const canEnroll = isAuthenticated() && hasRole('student');

  return (
    <div className="container my-4">
      <h2>📖 Каталог курсов</h2>

      { /* Сообщения об ошибке или успехе */ }
      {message && (
        <div
          className={`alert ${
            message.startsWith("✅")
              ? "alert-success"
              : "alert-danger"
          }`}
        >
          {message}
        </div>
      )}

      { /* Фильтры */ }
      <div className="row mb-4">
        <div className="col-md-6">
          <label>Категория:</label>
          <select
            className="form-select"
            value={filter.category}
            onChange={(e) => setFilter({ ...filter, category:
e.target.value })}
          >
            <option value="">Все категории</option>
            {categories.map((cat) => (
              <option key={cat.id} value={cat.id}>
                {cat.title}
              </option>
            ))}
          </select>
        </div>
        <div className="col-md-6">
          <label>Сложность:</label>
          <select
            className="form-select"
            value={filter.difficulty}
            onChange={(e) => setFilter({ ...filter, difficulty:
e.target.value })}
          >

```

```

        <option value="">Любая</option>
        <option value="beginner">Начальный</option>
        <option value="intermediate">Средний</option>
        <option value="advanced">Продвинутый</option>
      </select>
    </div>
  </div>

  {courses.length === 0 ? (
    <div className="alert alert-warning">Курсы не найдены.</div>
  ) : (
    <div className="row">
      {courses.map((course) => (
        <div className="col-md-6 col-lg-4 mb-4" key={course.id}>
          <div className="card h-100 shadow-sm border-primary">
            <div className="card-body d-flex flex-column">
              <h5 className="card-title">{course.title}</h5>
              <p className="card-text">{course.description}</p>
              <p className="text-muted">
                Сложность: <strong>{course.difficulty}</strong>
                {course.category_title && (
                  <> | Категория: {course.category_title}</>
                )}
              </p>
              <div className="mt-auto d-flex justify-content-between
align-items-center">
                <Link
                  to={`/${courses}/${course.id}`}
                  className="btn btn-outline-primary btn-sm"
                >
                  Подробнее
                </Link>
                {canEnroll && (
                  <button
                    onClick={() => handleEnroll(course.id)}
                    className="btn btn-success btn-sm"
                  >
                     Записаться
                  </button>
                )}
              </div>
            </div>
          </div>
        </div>
      )})}
    </div>
  )}
);
};

export default CoursesPage;

client/src/pages/RegisterPage.js:

import React, { useState } from 'react';
import API from '../api';

```

```

import { setToken } from '../auth';
import { useNavigate } from 'react-router-dom';

const RegisterPage = () => {
  const [form, setForm] = useState({
    username: '',
    email: '',
    password: '',
    role: 'student',
  });

  const navigate = useNavigate();

  const handleChange = e => {
    setForm({ ...form, [e.target.name]: e.target.value });
  };

  const handleSubmit = async e => {
    e.preventDefault();

    try {
      const res = await API.post('/auth/register', form);
      setToken(res.data.token);
      localStorage.setItem("roles", JSON.stringify(res.data.user.roles ||
[]));
      navigate('/dashboard');
    } catch {
      alert("❌ Ошибка регистрации");
    }
  };

  return (
    <div className="container mt-4">
      <h2>📝 Регистрация</h2>

      <form onSubmit={handleSubmit} className="w-50 mx-auto">
        <div className="mb-3">
          <label>Имя пользователя</label>
          <input
            name="username"
            className="form-control"
            value={form.username}
            onChange={handleChange}
            required
          />
        </div>

        <div className="mb-3">
          <label>Email</label>
          <input
            name="email"
            type="email"
            className="form-control"
            value={form.email}
            onChange={handleChange}
            required
          />
        </div>
      </form>
    </div>
  );
};

```

```

    <div className="mb-3">
      <label>Пароль</label>
      <input
        name="password"
        type="password"
        className="form-control"
        value={form.password}
        onChange={handleChange}
        required
      />
    </div>

    <div className="mb-4">
      <label>Роль</label>
      <select name="role" className="form-select" value={form.role}
onChange={handleChange}>
        <option value="student">Студент</option>
        <option value="teacher">Преподаватель</option>
        <option value="employer">Работодатель</option>
      </select>
    </div>

    <button className="btn btn-success w-
100">Зарегистрироваться</button>
  </form>
</div>
);
};

export default RegisterPage;

client/src/pages/CourseDetailPage.js:

import React, { useEffect, useState } from 'react';
import { useParams } from 'react-router-dom';
import API, { FILE_BASE } from '../api';
import { hasRole, isAuthenticated } from '../auth';
import CourseReviews from '../CourseReviews';

const CourseDetailPage = () => {
  const { id } = useParams();
  const [course, setCourse] = useState(null);
  const [isEnrolled, setIsEnrolled] = useState(false);
  const [message, setMessage] = useState('');
  const [students, setStudents] = useState([]);
  const [isAuthor, setIsAuthor] = useState(false);
  const [submissions, setSubmissions] = useState([]);
  const [materials, setMaterials] = useState([]);

  const [materialFile, setMaterialFile] = useState(null);
  const [submissionFile, setSubmissionFile] = useState(null);

  // Загрузка базы
  useEffect(() => {
    const fetchData = async () => {
      try {

```

```

        const res = await API.get(`/courses/${id}`);
        setCourse(res.data);
      } catch {
        setMessage("✗ Курс не найден");
      }

      if (hasRole("student")) {
        try {
          const res = await API.get("/enrollments");
          const found = res.data.find((e) => e.course_id === Number(id));
          setIsEnrolled(!found);
        } catch {}
      }

      try {
        const res = await API.get(`/courses/${id}/students`);
        setStudents(res.data);
        setIsAuthor(true);
      } catch {
        setIsAuthor(false);
      }
    };

    fetchData();
  }, [id]);

  useEffect(() => {
    if (isAuthor) {
      API.get(`/submissions/course/${id}`).then(res =>
setSubmissions(res.data));
    }
    API.get(`/courses/${id}/material`).then(res =>
setMaterials(res.data));
  }, [isAuthor, id]);

  const handleEnroll = async () => {
    try {
      await API.post("/enrollments", { course_id: id });
      setIsEnrolled(true);
      setMessage("☑ Вы записались на курс");
    } catch (err) {
      setMessage(err.response?.data?.message || "✗ Ошибка записи");
    }
  };

  const handleUnenroll = async () => {
    if (!window.confirm("Вы уверены?")) return;
    try {
      await API.delete(`/enrollments/${id}`);
      setIsEnrolled(false);
      setMessage("🚫 Вы отписались от курса");
    } catch (err) {
      setMessage(err.response?.data?.message || "✗ Ошибка при отписке");
    }
  };

  const handleUploadMaterial = async () => {

```

```

    if (!materialFile) return alert("Выберите файл");
    const formData = new FormData();
    formData.append("file", materialFile);
    try {
      await API.post(`/courses/${id}/material`, formData);
      setMaterialFile(null);
      const res = await API.get(`/courses/${id}/material`);
      setMaterials(res.data);
    } catch {
      alert("❌ Ошибка загрузки");
    }
  };

const handleSubmitWork = async () => {
  if (!submissionFile) return alert("Выберите файл");
  const formData = new FormData();
  formData.append("file", submissionFile);
  try {
    await API.post(`/enrollments/${id}/submit`, formData);
    setSubmissionFile(null);
    alert("📁 Работа отправлена на проверку");
  } catch {
    alert("❌ Ошибка отправки");
  }
};

const handleApprove = async (submissionId) => {
  try {
    await API.post(`/submissions/${submissionId}/approve`);
    setSubmissions((prev) =>
      prev.map((s) =>
        s.id === submissionId ? { ...s, is_approved: true } : s
      )
    );
    alert("✅ Работа одобрена");
  } catch {
    alert("❌ Ошибка при одобрении");
  }
};

if (!course) return <div className="container mt-5">Загрузка...</div>;

return (
  <div className="container my-4">
    <h2>📁 {course.title}</h2>
    <p>{course.description}</p>
    <p
      className="text-muted"><strong>Сложность:</strong>
    {course.difficulty}</p>

    {message && (
      <div className={`alert ${message.includes("✅") ? "alert-success"
: "alert-danger"}`}>
        {message}
      </div>
    )}

    { /* STUDENT */ }
  </div>
);

```

```

{isAuthenticated() && hasRole("student") && (
  <div className="my-4">
    {isEnrolled ? (
      <>
        <div className="alert alert-success">✔ Вы записаны на
курс</div>
        <button className="btn btn-outline-danger mb-3"
onClick={handleUnenroll}>
          ❌ Отписаться от курса
        </button>
        <div>
          <h5>📁 Итоговая работа</h5>
          <input type="file" onChange={ (e) =>
setSubmissionFile(e.target.files[0])} className="form-control mb-2" />
          <button className="btn btn-success"
onClick={handleSubmitWork}>📤 Отправить работу</button>
        </div>
      </>
    ) : (
      <button className="btn btn-primary" onClick={handleEnroll}>📖
Записаться</button>
    )}
  </div>
)}

{/* TEACHER: загрузка материала */}
{isAuthor && (
  <div className="my-4">
    <h5>📁 Загрузить учебный материал</h5>
    <input type="file" onChange={ (e) =>
setMaterialFile(e.target.files[0])} className="form-control mb-2" />
    <button className="btn btn-secondary"
onClick={handleUploadMaterial}>📤 Загрузить</button>
  </div>
)}

<div className="my-4">
  <h5>📁 Материалы курса</h5>
  {materials.length === 0 ? (
    <div className="alert alert-info">Материалов пока нет</div>
  ) : (
    <ul className="list-group">
      {materials.map((m) => (
        <li key={m.id} className="list-group-item">
          <a href={`${FILE_BASE}/uploads/${m.filename}`}
target="_blank" rel="noreferrer">
            {m.original_name}
          </a>
        </li>
      ))}
    </ul>
  )}
</div>

{isAuthor && submissions.length > 0 && (
  <div className="my-4">

```

```

<h5>📁 Итоговые работы студентов</h5>
<table className="table table-bordered">
  <thead>
    <tr>
      <th>Студент</th>
      <th>Файл</th>
      <th>Дата</th>
      <th>Статус</th>
      <th>Действие</th>
    </tr>
  </thead>
  <tbody>
    {submissions.map((s) => (
      <tr key={s.id}>
        <td>{s.username}</td>
        <td>
          <a
            href={`/${FILE_BASE}/uploads/${s.file_url}`}
            target="_blank" rel="noreferrer">
              {s.original_name}
            </a>
          </td>
        <td>{new Date(s.submitted_at).toLocaleDateString()}</td>
        <td className={s.is_approved ? "text-success" : "text-
muted"}>
          {s.is_approved ? "✅ Одобрено" : "⌚ Ожидает"}
        </td>
        <td>
          {!s.is_approved && (
            <button className="btn btn-sm btn-outline-success"
              onClick={() => handleApprove(s.id)}>
                ✅ Одобрить
              </button>
          )}
        </td>
      </tr>
    )})}
  </tbody>
</table>
</div>
))}

{/* ОТЗЫВЫ */}
<div className="my-5">
  <CourseReviews />
</div>
</div>
);
};

```

```
export default CourseDetailPage;
```

```
client/src/pages/VacanciesPage.js:
```

```

import React, { useEffect, useState } from "react";
import API from "../api";
import { hasRole } from "../auth";
import { Link } from "react-router-dom";

```

```

const VacanciesPage = () => {
  const [vacancies, setVacancies] = useState([]);
  const [user, setUser] = useState(null);
  const [recommendedCourses, setRecommendedCourses] = useState({});
  const [messages, setMessages] = useState({});

  useEffect(() => {
    const fetchData = async () => {
      const res = await API.get("/vacancies");
      setVacancies(res.data || []);

      if (hasRole("student")) {
        try {
          const userRes = await API.get("/users/me");
          setUser(userRes.data);

          for (let v of res.data) {
            const rec = await
API.get(`/vacancies/${v.id}/recommended/${userRes.data.id}`);
            setRecommendedCourses(prev => ({ ...prev, [v.id]: rec.data }));
          }
        } catch (err) {
          console.error("Ошибка загрузки данных студента");
        }
      }
    };
    fetchData();
  }, []);

  const handleApply = async (vacancyId) => {
    try {
      await API.post("/vacancies/apply", { vacancy_id: vacancyId });
      setMessages(prev => ({ ...prev, [vacancyId]: "✔ Вы успешно
откликнулись" }));
    } catch (err) {
      const message = err.response?.data?.message || "✘ Ошибка при
отклике";
      setMessages(prev => ({ ...prev, [vacancyId]: message }));
    }
  };

  return (
    <div className="container mt-4">
      <h2>👛 Вакансии</h2>
      {vacancies.length === 0 ? (
        <div className="alert alert-warning">Нет вакансий</div>
      ) : (
        <div className="row">
          {vacancies.map((v) => (
            <div className="col-md-6 mb-4" key={v.id}>
              <div className="card h-100 border-primary shadow">
                <div className="card-body">
                  <h5 className="card-title">{v.title}</h5>
                  <p>{v.description}</p>
                  <p
className="mb-1"><strong>Зарплата:</strong>
{v.salary_range}</p>

```

```

        <p                                className="mb-1"><strong>Тип:</strong>
{v.employment_type}, <strong>Город:</strong> {v.location}</p>
        <p><strong>Удалёнка:</strong> {v.remote_available ? "✔
Да" : "✗ Нет"}</p>

        {hasRole("student") && (
            <>
                <button            className="btn            btn-outline-primary"
onClick={ () => handleApply(v.id) }>
                     Откликнуться
                </button>
                {messages[v.id]            &&            <p            className="mt-
2">{messages[v.id]}</p>}
            </>
        )}

        {(recommendedCourses[v.id] || []).length > 0 && (
            <div className="mt-3">
                <h6> Рекомендации:</h6>
                <ul className="list-unstyled">
                    {recommendedCourses[v.id].map((c) => (
                        <li key={c.id}>
                            
                            <Link
to={` /courses/${c.id}`}>{c.title}</Link>
                            <span
muted">{c.difficulty}</span>
                            </li>
                        )))
                </ul>
            </div>
        )}
    </div>
</div>
</div>
</div>
    )}
</div>
);
};

```

```
export default VacanciesPage;
```

```
client/src/pages/CourseReviews.js:
```

```

import React, { useEffect, useState } from 'react';
import API from '../api';
import { useParams } from 'react-router-dom';
import { hasRole } from '../auth';

const CourseReviews = () => {
    const { id: courseId } = useParams();
    const [reviews, setReviews] = useState([]);
    const [form, setForm] = useState({ rating: 5, text: '' });
    const [message, setMessage] = useState('');

    useEffect(() => {

```

```

const fetchReviews = async () => {
  const res = await API.get(`/courses/${courseId}/reviews`);
  setReviews(res.data);
};
fetchReviews();
}, [courseId]);

const handleSubmit = async (e) => {
  e.preventDefault();
  try {
    await API.post(`/courses/${courseId}/reviews`, form);
    setMessage("✅ Отзыв отправлен!");
    setForm({ rating: 5, text: "" });
    const res = await API.get(`/courses/${courseId}/reviews`);
    setReviews(res.data);
  } catch (err) {
    setMessage("❌ Ошибка при отправке отзыва");
  }
};

return (
  <div>
    <h3>Отзывы о курсе</h3>

    {hasRole("student") && (
      <form onSubmit={handleSubmit} style={{ marginBottom: "20px" }}>
        <label>Оценка (1-5):</label>
        <select
          value={form.rating}
          onChange={(e) => setForm({ ...form, rating: +e.target.value })}
        >
          {[1, 2, 3, 4, 5].map((n) => (
            <option key={n} value={n}>
              {n}
            </option>
          ))}
        </select>
        <br />
        <textarea
          placeholder="Ваш комментарий"
          value={form.text}
          onChange={(e) => setForm({ ...form, text: e.target.value })}
          required
        />
        <br />
        <button type="submit">💬 Оставить отзыв</button>
        {message && <p>{message}</p>}
      </form>
    )}

    <ul>
      {reviews.map((r) => (
        <li key={r.id}>
          <strong>{r.username}</strong>: ☆ {r.rating}
          <p>{r.text}</p>
        </li>
      ))}
    </ul>
  </div>
)

```

```

        </ul>
      </div>
    );
  };
};

export default CourseReviews;

```

client/src/components/Navbar.js:

```

import React from 'react';
import { Link, useNavigate } from 'react-router-dom';
import { isAuthenticated, hasRole, removeToken } from '../auth';

const Navbar = () => {
  const navigate = useNavigate();

  const handleLogout = () => {
    removeToken();
    navigate('/login');
  };

  return (
    <nav style={{ background: '#eee', padding: '10px' }}>
      <Link to="/">🏠 Home</Link> | <Link to="/courses">📖 Курсы</Link>

      {isAuthenticated() && (
        <>
          {' | '}
          <Link to="/dashboard">👤 Кабинет</Link>
        </>
      )}

      {/* + Только для teacher/employer */}
      {isAuthenticated() && (hasRole('teacher') || hasRole('employer')) && (
        <>
          {' | '}
          <Link to="/create-course">+ Создать курс</Link>
        </>
      )}

      {/* ✅ Только для студентов: сертификаты */}
      {isAuthenticated() && hasRole('student') && (
        <>
          {' | '}
          <Link to="/certificates">📜 Сертификаты</Link>
        </>
      )}

      {isAuthenticated() && hasRole('admin') && (
        <>
          {' | '}
          <Link to="/admin">⚙️ Админка</Link>
        </>
      )}
    </nav>
  );
};

```

```

    {hasRole('student') && (
      <>
        {" | "}
        <Link to="/recommendations">🔗 Рекомендации</Link>
      </>
    )}

    {' | '}
    <Link to="/vacancies">📋 Вакансии</Link>

    {hasRole('employer') && (
      <>
        {' | '}
        <Link to="/create-vacancy">📄 Создать вакансию</Link>
      </>
    )}

    {!isAuthenticated() ? (
      <>
        {' | '}
        <Link to="/login">🔑 Вход</Link> | <Link to="/register">📄
Регистрация</Link>
      </>
    ) : (
      <>
        {' | '}
        <button
          onClick={handleLogout}
          style={{ background: 'none', border: 'none', color: 'blue',
cursor: 'pointer' }}
        >
          🚪 Выйти
        </button>
      </>
    )}
  </nav>
);
};

export default Navbar;

```

client/src/components/PrivateRoute.js:

```

import { Navigate } from 'react-router-dom';
import { isAuthenticated } from '../auth';

const PrivateRoute = ({ children }) => {
  return isAuthenticated() ? children : <Navigate to="/login" />;
};

export default PrivateRoute;

```

Приложение Б
(рекомендуемое)

Таблица Б.1 – Табличное описание сущностей для концептуальной модели базы данных

Сущность	Описание	Атрибуты
Users	Зарегистрированные пользователи системы	id, username, email, password_hash, is_blocked, created_at
Admins	Администраторы системы	user_id, department, access_level, last_login
Students	Студенты, обучающиеся на курсах	user_id, enrollment_year, university, bio, gpa, linkedin_url
Teachers	Преподаватели, создающие курсы	user_id, department, academic_degree, office_number, is_verified
Employers	Работодатели, предлагающие курсы и вакансии	user_id, company_name, company_description, website_url, verified, company_logo_url
Levels	Уровни сложности курсов	id, title, description, sequence
Courses	Курсы, предлагаемые на платформе	id, title, description, teacher_id, created_at, thumbnail_url, difficulty
Modules	Модули, входящие в курсы	id, level_id, course_id, title, is_required, sequence
Lessons	Уроки, входящие в модули	id, module_id, title, content, sequence, duration_minutes, video_url
Enrollments	Записи студентов на курсы	id, user_id, course_id, is_completed, enrolled_at, completed_at

Продолжение таблицы Б.1

Сущность	Описание	Атрибуты
Certificates	Сертификаты, выданные студентам	id, user_id, course_id, issue_date, certificate_url, verification_code
Skills	Профессиональные навыки	id, name, description, category
Course Skills	Навыки, связанные с курсами	course_id, skill_id, level_gain
Student Skills	Навыки, освоенные студентами	student_id, skill_id, achieved_level, last_updated
Role Trajectories	Карьерные предлагаемые пути, работодателями	id, employer_id, title, description, industry, avg_salary
Trajectory Courses	Курсы, входящие в траектории	trajectory_id, course_id
Student Trajectories	Участие студентов в траекториях	student_id, trajectory_id, started_at, completed_at
Vacancies	Вакансии, предлагаемые работодателями	id, employer_id, title, description, salary_range, employment_type, remote_available, is_active, created_at, expires_at, location
Vacancy Requirements	Требования к вакансиям	id, vacancy_id, requirement_text, is_mandatory, requirement_type
Vacancy Skills	Навыки, требуемые для вакансий	vacancy_id, skill_id, required_level

Окончание таблицы Б.1

Сущность	Описание	Атрибуты
Application Documents	Документы, приложенные к заявкам	id, application_id, document_type, content, file_url
Student Achievements	Достижения студентов	id, student_id, achievement_type, title, description, achieved_at, image_url
Applications	Заявки студентов на вакансии	id, vacancy_id, student_id, status, applied_at, reviewed_at, feedback

Таблица Б.2 – Спецификации связей между сущностями для концептуальной модели базы данных

Связь	Сущность 1	Сущность 2	Тип связи	Описание связи
Users-Admins	Users	Admins	Один-к-одному	Пользователь может быть администратором
Users-Students	Users	Students	Один-к-одному	Пользователь может быть студентом
Users-Teachers	Users	Teachers	Один-к-одному	Пользователь может быть преподавателем
Users-Employers	Users	Employers	Один-к-одному	Пользователь может быть работодателем
Teachers-Courses	Teachers	Courses	Один-ко-многим	Преподаватель может создавать несколько курсов

Продолжение таблицы Б.2

Связь	Сущность 1	Сущность 2	Тип связи	Описание связи
Modules-Lessons	Modules	Lessons	Один-ко-многим	Модуль может содержать несколько уроков
Students-Enrollments	Students	Enrollments	Один-ко-многим	Студент может записаться на несколько курсов
Courses-Enrollments	Courses	Enrollments	Один-ко-многим	На курс может записаться несколько студентов
Students-Certificates	Students	Certificates	Один-ко-многим	Студент может получить несколько сертификатов
Courses-Certificates	Courses	Certificates	Один-ко-многим	Курс может быть связан с несколькими сертификатами
Courses-Course Skills	Courses	Course Skills	Один-ко-многим	Курс может быть связан с несколькими навыками
Skills-Course Skills	Skills	Course Skills	Один-ко-многим	Навык может быть связан с несколькими курсами
Students-Student Skills	Students	Student Skills	Один-ко-многим	Студент может освоить несколько навыков
Skills-Student Skills	Skills	Student Skills	Один-ко-многим	Навык может быть освоен несколькими студентами
Employers-Role Trajectories	Employers	Role Trajectories	Один-ко-многим	Работодатель может предлагать несколько траекторий

Продолжение таблицы Б.2

Связь	Сущность 1	Сущность 2	Тип связи	Описание связи
Courses-Trajectory Courses	Courses	Trajectory Courses	Один-ко-многим	Курс может входить в несколько траекторий
Students-Student Trajectories	Students	Student Trajectories	Один-ко-многим	Студент может участвовать в нескольких траекториях
Role Trajectories-Student Trajectories	Role Trajectories	Student Trajectories	Один-ко-многим	Траектория может быть выбрана несколькими студентами
Employers-Vacancies	Employers	Vacancies	Один-ко-многим	Работодатель может предлагать несколько вакансий
Vacancies-Vacancy Requirements	Vacancies	Vacancy Requirements	Один-ко-многим	Вакансия может иметь несколько требований
Vacancies-Vacancy Skills	Vacancies	Vacancy Skills	Один-ко-многим	Вакансия может требовать несколько навыков
Skills-Vacancy Skills	Skills	Vacancy Skills	Один-ко-многим	Навык может быть требованием для нескольких вакансий
Vacancies-Applications	Vacancies	Applications	Один-ко-многим	На вакансию может быть подано несколько заявок
Students-Applications	Students	Applications	Один-ко-многим	Студент может подать несколько заявок
Applications-Application Documents	Applications	Application Documents	Один-ко-многим	Заявка может включать несколько документов

Окончание таблицы Б.2

Role Trajectories- Trajectory Courses	Role Trajectories	Trajectory Courses	Один-ко- многим	Траектория может включать несколько курсов
Courses-Modules	Courses	Modules	Один-ко- многим	Курс может содержать несколько модулей

Таблица Б.3 – Спецификация отношений логической модели базы данных

Таблица	Связанная таблица	Тип связи	Внешний ключ	Описание связи
admins	users	Один-к- одному	user_id	Ссылка на пользователя, являющегося администрато ром
students	users	Один-к- одному	user_id	Ссылка на пользователя, являющегося студентом
teachers	users	Один-к- одному	user_id	Ссылка на пользователя, являющегося преподавателе м
employers	users	Один-к- одному	user_id	Ссылка на пользователя, являющегося работодателе м
courses	teachers	Многие-к- одному	teacher_id	Ссылка на преподавателя , создавшего курс
modules	levels	Многие-к- одному	level_id	Ссылка на уровень сложности модуля

Продолжение таблицы Б.3

Таблица	Связанная таблица	Тип связи	Внешний ключ	Описание связи
enrollments	students	Многие-к-одному	user_id	Ссылка на студента, записавшегося
enrollments	courses	Многие-к-одному	course_id	Ссылка на курс, на который записан студент
certificates	students	Многие-к-одному	user_id	Ссылка на студента, получившего сертификат
certificates	courses	Многие-к-одному	course_id	Ссылка на курс, по которому выдан сертификат
course_skills	courses	Многие-к-одному	course_id	Ссылка на курс, связанный с навыком
course_skills	skills	Многие-к-одному	skill_id	Ссылка на навык, связанный с курсом
student_skills	students	Многие-к-одному	student_id	Ссылка на студента, освоившего навык
student_skills	skills	Многие-к-одному	skill_id	Ссылка на навык, освоенный студентом

Продолжение таблицы Б.3

Таблица	Связанная таблица	Тип связи	Внешний ключ	Описание связи
trajectory_courses	role_trajectories	Многие-к-одному	trajectory_id	Ссылка на траекторию
trajectory_courses	courses	Многие-к-одному	course_id	Ссылка на курс, входящий в траекторию
student_trajectories	students	Многие-к-одному	student_id	Ссылка на студента, участвующего в траектории
student_trajectories	role_trajectories	Многие-к-одному	trajectory_id	Ссылка на траекторию, выбранную студентом
vacancies	employers	Многие-к-одному	employer_id	Ссылка на работодателя, предлагающего вакансию
vacancy_requirements	vacancies	Многие-к-одному	vacancy_id	Ссылка на вакансию, имеющую требование
vacancy_skills	vacancies	Многие-к-одному	vacancy_id	Ссылка на вакансию, требующую навык
vacancy_skills	skills	Многие-к-одному	skill_id	Ссылка на навык, требуемый для вакансии
applications	vacancies	Многие-к-одному	vacancy_id	Ссылка на вакансию, на которую подана заявка

Окончание таблицы Б.3

Таблица	Связанная таблица	Тип связи	Внешний ключ	Описание связи
application_documents	applications	Многие-к-одному	application_id	Ссылка на заявку
student_achievements	students	Многие-к-одному	student_id	Ссылка на студента
modules	courses	Многие-к-одному	course_id	Ссылка на курс, содержащий модуль
lessons	modules	Многие-к-одному	module_id	Ссылка на модуль, содержащий урок
role_trajectories	employers	Многие-к-одному	employer_id	Ссылка на работодателя

Таблица Б.4 – Описание сценария варианта использования «Посмотреть каталог курсов»

Параметр	Описание
Наименование	Посмотреть каталог курсов
ID	1
Краткое описание	Пользователь просматривает список доступных курсов.
Действующие лица	Студент, гость
Предусловие	Пользователь находится на главной странице платформы.
Основной поток	1. Пользователь открывает каталог курсов. 2. Пользователь просматривает список курсов.
Постусловие	Пользователь видит список доступных курсов.

Таблица Б.5 – Описание сценария варианта использования «Использовать параметры поиска курсов»

Параметр	Описание
Наименование	Использовать параметры поиска курсов
ID	2
Краткое описание	Пользователь использует фильтры для поиска курсов.
Действующие лица	Студент, гость
Предусловие	Пользователь находится в каталоге курсов.
Основной поток	1. Пользователь выбирает фильтр. 2. Система отображает релевантные курсы.
Постусловие	Пользователь видит результаты поиска в каталоге

Таблица Б.6 – Описание сценария варианта использования «Посмотреть информацию о курсе»

Параметр	Описание
Наименование	Посмотреть информацию о курсе
ID	3
Краткое описание	Пользователь открывает подробности о выбранном курсе.
Действующие лица	Студент, гость
Предусловие	Пользователь выбрал курс из каталога.
Основной поток	1. Пользователь кликает на курс. 2. Система отображает подробности о курсе.
Постусловие	Пользователь получает информацию о курсе.

Таблица Б.7 – Описание сценария варианта использования «Регистрация в системе»

Параметр	Описание
Наименование	Регистрация в системе
ID	4
Краткое описание	Гость регистрируется для получения доступа к дополнительным функциям платформы.
Действующие лица	Гость
Предусловие	Гость перешел на страницу регистрации.
Основной поток	1. Гость вводит регистрационные данные. 2. Система проверяет данные. 3. Пользователь успешно регистрируется.
Постусловие	Пользователь зарегистрирован в системе.

Таблица Б.8 – Описание сценария варианта использования «Авторизация в системе»

Параметр	Описание
Наименование	Авторизация в системе
ID	5
Краткое описание	Пользователь входит в систему с помощью логина и пароля.
Действующие лица	Студент, преподаватель, администратор
Предусловие	Пользователь имеет действующую учетную запись.
Основной поток	1. Пользователь вводит логин и пароль. 2. Система проверяет данные. 3. Система предоставляет доступ.
Постусловие	Пользователь авторизован в системе.

Таблица Б.9 – Описание сценария варианта использования «Записаться на курс»

Параметр	Описание
Наименование	Записаться на курс
ID	6
Краткое описание	Студент записывается на курс для его прохождения.
Действующие лица	Студент
Предусловие	Пользователь авторизован и выбрал курс.
Основной поток	1. Пользователь нажимает «Записаться». 2. Система регистрирует пользователя на курс.
Постусловие	Пользователь зачислен на курс.

Таблица Б.10 – Описание сценария варианта использования «Просмотреть прогресс по курсам»

Параметр	Описание
Наименование	Просмотреть прогресс по курсам
ID	7
Краткое описание	Студент просматривает список своих курсов и прогресс по ним.
Действующие лица	Студент
Предусловие	Пользователь авторизован в системе.
Основной поток	1. Пользователь открывает личный кабинет. 2. Переходит в раздел «Мои курсы».
Постусловие	Пользователь видит список курсов и текущий прогресс.

Таблица Б.11 – Описание сценария варианта использования «Проходить уроки»

Параметр	Описание
Наименование	Проходить уроки
ID	8
Краткое описание	Пользователь изучает материалы, выполняет задания и тесты в рамках курса.
Действующие лица	Студент
Предусловие	Пользователь записан на курс.

Продолжение таблицы Б.11

Основной поток	1. Пользователь выбирает урок. 2. Изучает содержание. 3. Выполняет задания или тесты.
Постусловие	Урок завершен, результаты сохранены.

Таблица Б.12 – Описание сценария варианта использования «Получить сертификат»

Параметр	Описание
Наименование	Получить сертификат
ID	9
Краткое описание	После завершения курса студент получает электронный сертификат.
Действующие лица	Студент
Предусловие	Студент завершил все модули и задания курса.
Основной поток	1. Система проверяет выполнение всех условий. 2. Генерирует сертификат.
Постусловие	Сертификат становится доступным для загрузки.

Таблица Б.13 – Описание сценария варианта использования «Оставить отзыв и оценку курсу»

Параметр	Описание
Наименование	Оставить отзыв и оценку курсу
ID	10
Краткое описание	Студент оценивает курс и оставляет комментарий.
Действующие лица	Студент
Предусловие	Курс завершен или просмотрен.
Основной поток	1. Студент открывает курс. 2. Пишет отзыв и выставляет оценку.
Постусловие	Отзыв отображается на странице курса.

Таблица Б.14 – Описание сценария варианта использования «Общение с другими пользователями»

Параметр	Описание
Наименование	Общение с другими пользователями

Продолжение таблицы Б.14

ID	11
Краткое описание	Пользователь взаимодействует с другими участниками платформы.
Действующие лица	Студент, преподаватель
Предусловие	Пользователь авторизован.
Основной поток	1. Пользователь заходит в чат/форум. 2. Пишет и читает сообщения.
Постусловие	Пользователь общается через интерфейс платформы.

Таблица Б.15 – Описание сценария варианта использования «Управление курсами преподавателем»

Параметр	Описание
Наименование	Управление курсами преподавателем
ID	12
Краткое описание	Преподаватель может создавать, редактировать и структурировать свои курсы.
Действующие лица	Преподаватель
Предусловие	Преподаватель авторизован.
Основной поток	1. Открытие панели курсов. 2. Создание или редактирование курса.
Постусловие	Курс создан или обновлен и доступен студентам.

Таблица Б.16 – Описание сценария варианта использования «Проверка заданий студентов»

Параметр	Описание
Наименование	Проверка заданий студентов
ID	13
Краткое описание	Преподаватель проверяет и оценивает работы студентов.
Действующие лица	Преподаватель
Предусловие	Студенты выполнили и отправили задания.
Основной поток	1. Преподаватель открывает список заданий. 2. Просматривает и оценивает их.
Постусловие	Оценки сохранены и доступны студентам.