**Содержание**

[Введение 5](#_Toc198763515)

[1 Постановка задачи и обзор аналогичных решений 6](#_Toc198763516)

[1.1 Постановка задачи 6](#_Toc198763517)

[1.2 Обзор аналогов 6](#_Toc198763518)

[1.2.1 Аналог «Udemy» 6](#_Toc198763519)

[1.2.2 Аналог «Stepik» 7](#_Toc198763520)

[1.2.3 Аналог «GeekBrains» 8](#_Toc198763521)

[1.3 Выводы по разделу 9](#_Toc198763522)

[2 Проектирование web-приложения 10](#_Toc198763523)

[2.1 Функциональные возможности web-приложения 10](#_Toc198763524)

[2.2 Проектирование базы данных 12](#_Toc198763525)

[2.3 Архитектура web-приложения 16](#_Toc198763526)

[2.4 Выводы по разделу 16](#_Toc198763527)

[3 Реализация web-приложения 17](#_Toc198763528)

[3.1 Программная платформа NodeJS 17](#_Toc198763529)

[3.2 Система управления базами данных PostgreSQL 17](#_Toc198763530)

[3.3 Реализация базы данных 17](#_Toc198763531)

[3.4 Программные библиотеки 18](#_Toc198763532)

[3.5 Структура серверной части 20](#_Toc198763533)

[3.6 Реализация функций пользователя с ролью «гость» 23](#_Toc198763534)

[3.6.1 Регистрация 23](#_Toc198763535)

[3.6.2 Аутентификация 24](#_Toc198763536)

[3.7 Реализация функций пользователя с ролью «ученик» 25](#_Toc198763537)

[3.7.1 Просмотр доступных курсов 25](#_Toc198763538)

[3.7.2 Подписка и прохождение курсов 27](#_Toc198763539)

[3.7.3 Отслеживание текущего и завершенного обучения 28](#_Toc198763540)

[3.7.4 Управление отзывами и расчет среднего рейтинга курса 29](#_Toc198763541)

[3.7.5 Персонализация и удаление профиля пользователя 30](#_Toc198763542)

[3.8 Реализация функций пользователя с ролью «учитель» 31](#_Toc198763543)

[3.8.1 Добавление, редактирование, удаление курсов 31](#_Toc198763544)

[3.8.2 Добавление, изменение, удаление разделов курса 35](#_Toc198763545)

[3.8.3 Управление статусом курсов и их разделов 37](#_Toc198763546)

[3.9 Реализация функций пользователя с ролью «администратор» 39](#_Toc198763547)

[3.9.1 Добавление, редактирование, удаление учетной записи учителя 39](#_Toc198763548)

[3.9.2 Удаление отзывов 40](#_Toc198763549)

[3.9.3 Блокировка курсов 40](#_Toc198763550)

[3.11 Выводы по разделу 44](#_Toc198763551)

[4 Тестирование web-приложения 45](#_Toc198763552)

[4.1 Функциональное тестирование 45](#_Toc198763553)

[4.2 Выводы по разделу 48](#_Toc198763554)

[5 Руководство пользователя 49](#_Toc198763555)

[5.1 Руководство для роли «гость» 49](#_Toc198763556)

[5.2 Руководство для роли «ученик» 49](#_Toc198763557)

[5.3 Руководство для роли «учитель» 53](#_Toc198763558)

[5.4 Руководство для роли «администратор» 55](#_Toc198763559)

[5.5 Выводы по разделу 56](#_Toc198763560)

[Заключение 57](#_Toc198763561)

[Список используемых источников 58](#_Toc198763562)

[Приложение А 59](#_Toc198763563)

[Приложение Б 60](#_Toc198763564)

[Приложение В 61](#_Toc198763565)

[Приложение Г 62](#_Toc198763566)

[Приложение Д 64](#_Toc198763567)

# Введение

Web-приложение – это клиент-серверное приложение, в котором клиент взаимодействует с сервером через протокол HTTP [1].

Обучение – это непрерывный процесс, направленный на приобретение новых знаний и навыков. Целью курсового проекта является упрощение процесса обучения с помощью web-приложения. Данное web-приложение предоставляет удобную среду для прохождения курсов, организации учебного процесса и отслеживания прогресса.

Web-приложение поддерживает несколько ролей: гость, студент, преподаватель и администратор. Основные функции включают возможность регистрации на курсы, прохождения учебных разделов, а также просмотра отзывов и рейтингов курсов.

Целевая аудитория включает студентов, стремящихся к саморазвитию, и преподавателей, создающих и управляющих курсами. Web-приложение также будет полезно организациям, желающим проводить корпоративное обучение сотрудников.

Для достижения указанной цели поставлены следующие задачи.

1. Проанализировать аналоги (раздел 1).
2. Спроектировать архитектуру приложения, обеспечивающую поддержку всех заявленных функций (раздел 2).
3. Спроектировать базу данных (раздел 3).
4. Разработать приложение по спроектированной архитектуре (раздел 3).
5. Организовать тестирование и отладку приложения для обеспечения его стабильной работы (раздел 4).

В качестве программной платформы выбрана Node.js [2], что обеспечивает высокую производительность, асинхронную обработку запросов и удобство в разработке web-приложений. Благодаря своей масштабируемости и широкому выбору библиотек, Node.js позволяет эффективно реализовать функциональность обучающей платформы. В качестве базы данных будет использоваться **PostgreSQL [3]**, что обеспечит гибкость хранения данных.

# Постановка задачи и обзор аналогичных решений

## **Постановка задачи**

Web-приложение для образовательной платформы должно обеспечивать эффективное взаимодействие между студентами, преподавателями и администраторами, автоматизируя процессы обучения. Студенты начинают с регистрации и авторизации, после чего могут выбирать курсы, записываться на них и проходить учебные разделы. Данные о прохождении сохраняются, а прогресс можно отслеживать.

Преподаватели создают, управляют курсами и их частями. Администраторы контролируют процесс обучения, управляют пользователями и следят за качеством контента.

После завершения курса студенты могут оставлять отзывы с комментариями и рейтингом учебному курсу. Вся история обучения сохраняется, что позволяет студентам возвращаться к материалам и улучшать свои знания.

# 1.2 Обзор аналогов

# 1.2.1 Аналог «Udemy»

Одним из аналогов разрабатываемого web-приложения является «Udemy» [4] – популярная международная платформа онлайн-обучения, предоставляющая доступ к тысячам курсов по различным темам (рисунок 1.1).

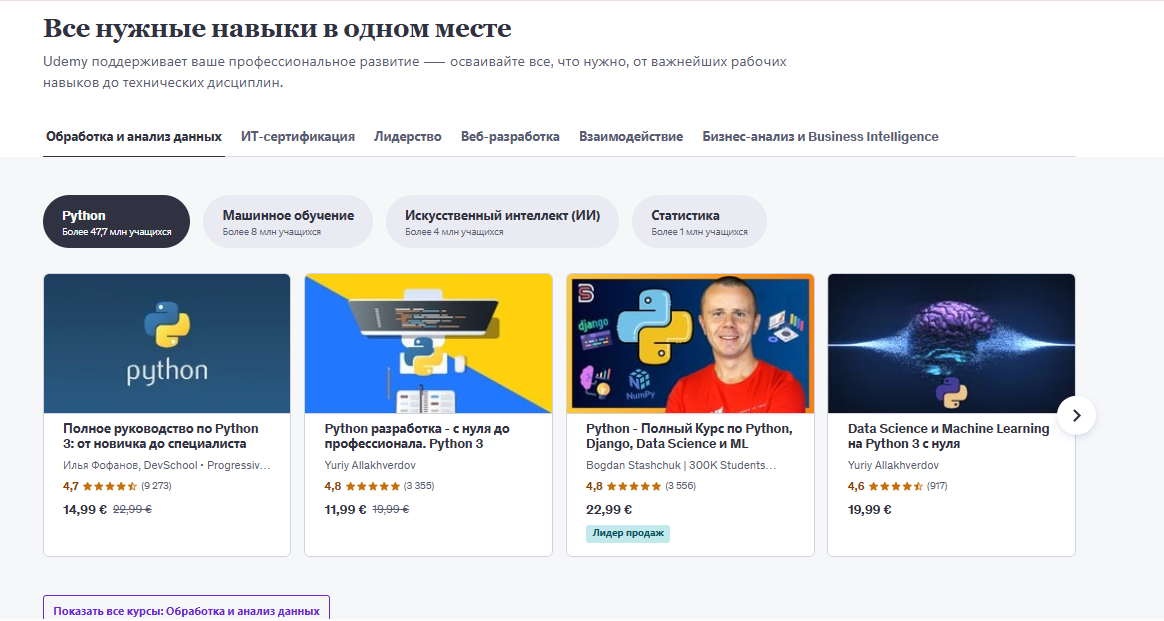


Рисунок 1.1 – Сайт «Udemy»

Данный сервис ориентирован как на студентов, стремящихся к саморазвитию, так и на преподавателей, создающих и продающих свои образовательные материалы. Платформа предоставляет возможность регистрации и авторизации пользователей, после чего студенты получают доступ к персональному кабинету, а преподаватели могут разрабатывать и управлять своими курсами. В каталоге «Udemy» представлены как платные, так и бесплатные курсы, включающие видео, текстовые материалы, тесты и практические задания.

Пользователи могут оставлять отзывы и оценки курсов, что помогает другим студентам ориентироваться в выборе. Обучение сопровождается системой отслеживания прогресса, позволяющей студентам видеть выполненные задания и продолжать обучение с места остановки.

Среди преимуществ «Udemy» можно выделить широкий выбор курсов, удобный интерфейс и систему скидок, позволяющую студентам приобретать курсы по сниженной цене. Однако у платформы есть и недостатки, такие как отсутствие четкой структуры образовательных программ, разное качество контента из-за возможности создания курсов любым пользователем, а также ограниченная интерактивность, так как обучение в основном строится на просмотре видео. Также недостатком является загруженный, не интуитивно понятный интерфейс, тем самым ограничивает целевую аудиторию, так как многим пользователям интерфейс может показаться трудым.

В целом, «Udemy» представляет собой мощную платформу для онлайн-обучения, предоставляя огромное количество курсов и гибкость как для студентов, так и для преподавателей.

# Аналог «Stepik»

«Stepik» [5] – это популярная российская платформа для онлайн-обучения, ориентированная на создание и прохождение образовательных курсов (рисунок 1.2).

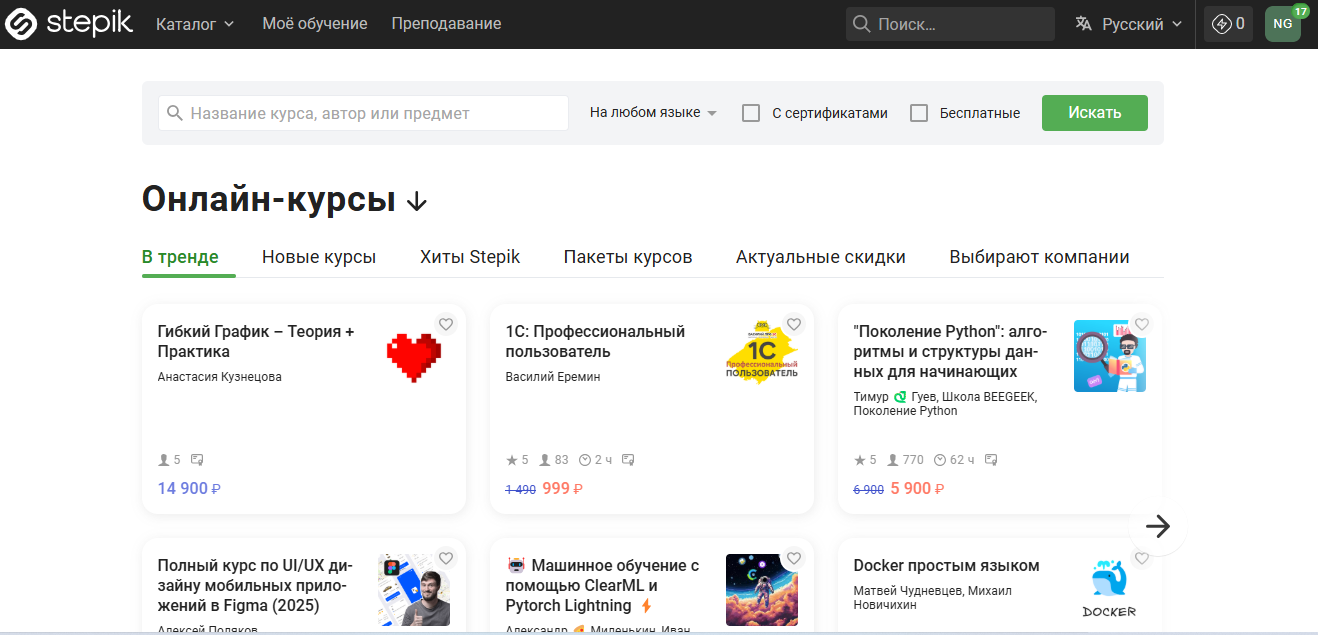


Рисунок 1.2 – Сайт «Stepik»

Один из ключевых аспектов платформы – интерактивные задания, такие как задания с автоматической проверкой, которые позволяют студентам не только изучать теорию, но и практиковать навыки в реальном времени. Платформа поддерживает создание курсов с использованием различных типов контента, включая видео, текст, изображения и вопросы с развернутыми ответами.

«Stepik» имеет большую вариацию фильтрации курсов по разным аспектам, а также поиск по курсам, который производится не только по названию курса, но и по ключевым словам, которые присутствуют в курсе. Также присутствует удобный личный кабинет, который позволяет не только отслеживать прогресс по курсу, но и активность пользователя за определенный период. Пользователь за прохождение разделов курсов получает баллы, что дает ему стимул ещё больше развиваться.

Большим недостатком является то, что для некоторых преподавателей процесс создания курса может быть сложным и требовать дополнительных усилий для интеграции мультимедийных материалов или создания нестандартных заданий. Хотя «Stepik» предоставляет инструменты для создания курсов, они не всегда интуитивно понятны, особенно для пользователей с ограниченным техническим опытом.

# Аналог «GeekBrains»

**«GeekBrains»** [6] – это российская образовательная онлайн-платформа, которая предлагает курсы и программы по программированию, дизайну, маркетингу и другим востребованным профессиям в области IT.

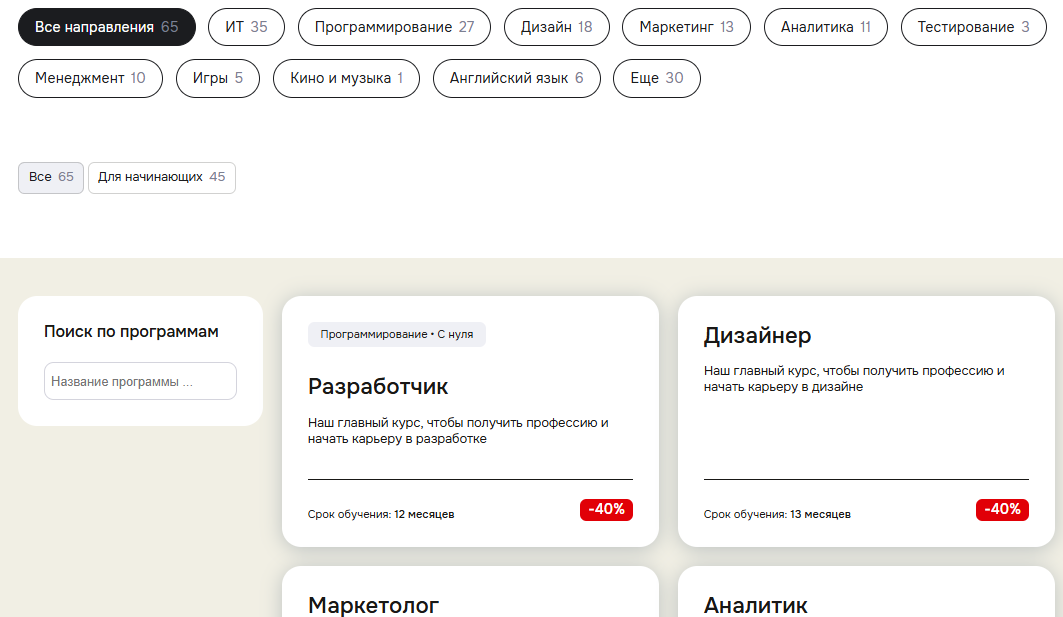


Рисунок 1.3 – Сайт «GeekBrains»

Одна из ключевых особенностей платформы – это возможность получения диплома или сертификата по завершении курса, что может служить доказательством навыков для работодателей. Также есть поддержка карьерных консультаций, где специалисты помогают студентам с трудоустройством и карьерным ростом в выбранной области. Платформа активно сотрудничает с крупными компаниями, что дает студентам возможность проходить практику и стажировки, а также участвовать в реальных проектах.

Платформа предлагает разнообразные форматы обучения, включая виде-уроки, практические задания и поддержку наставников, что позволяет создать гибкую и удобную среду для студентов.

Кроме того, платформа «GeekBrains» предоставляет удобный личный кабинет, в котором студент может отслеживать прогресс по курсам, управлять расписанием занятий и получать персональные рекомендации. Программа обучения часто включает в себя итоговые проекты, выполнение которых позволяет закрепить полученные знания на практике.

Одним из недостатков является то, что некоторые курсы ориентированы исключительно на IT-специалистов, и меньше внимания уделяется более гуманитарным или творческим дисциплинам. Для студентов, интересующихся другими областями, платформа может не предложить подходящего контента, что урезает целевую аудиторию.

# Выводы по разделу

После анализа трех популярных образовательных платформ можно выделить несколько ключевых особенностей. Необходимо уделить особое внимание разработке **хорошей системы поиска и фильтрации курсов**,чтобы пользователи могли легко находить нужные материалы среди большого количества курсов.

Также важным аспектом является **удобное отслеживание прогресса** студентов, что позволит им видеть достижения и мотивировать их к дальнейшему обучению. Не менее важным элементом является создание удобной системы для **преподавателей**, которая позволяла бы им легко создавать и управлять курсами.

Также немаловажным фактором является наличие системы отзывов и рейтингов, которая позволяет ученикам выражать своё мнение о курсе, а преподавателям – получать ценную обратную связь. Это способствует улучшению качества образовательного контента и повышает доверие к платформе.

# Проектирование web-приложения

# 2.1 Функциональные возможности web-приложения

Функциональные возможности web-приложения представлены в диаграмме вариантов использования, представленной в Приложении А. Данная диаграмма вариантов использования наглядно демонстрирует функциональность web-приложения для различных ролей пользователей. Перечень ролей и их назначение приведены в таблице 2.1.

Таблица 2.1 – Назначение ролей пользователей в web-приложении

|  |  |
| --- | --- |
| Роль | Назначение |
| Гость | Регистрация и аутентификация. |
| Ученик | Просмотр списка курсов, фильтрация и поиск курсов, подписка на курс, прохождение разделов курса, просмотр прогресса курса, добавления и удаление отзыва на курс, редактирование и удаление профиля. |
| Учитель | Создание, изменения, удаление курсов и разделов курсов, просмотр статистики курсов, изменение статуса курса. |
| Администратор | Создание, изменение, удаление учетный записей учителей, просмотр статистики по курсам, удаление отзыва, блокировка курса. |

Функциональные возможности пользователя с ролью «гость» представлены в таблице 2.2.

Таблица 2.2 – Функциональные возможности пользователя с ролью «гость»

|  |  |  |
| --- | --- | --- |
| № | Вариант использования | Пояснение |
| 1 | Регистрация | Гость может создать учетную запись, заполнив необходимые данные, чтобы получить доступ к полному функционалу web-приложения. |
| 2 | Аутентификация | Процесс предоставление пользователям прав на выполнение определенных действий с использованием учетных данных, таких как логин и пароль. |

Далее рассмотрим функциональные возможности для пользователя с ролью «ученик», которые представлены в таблице 2.3.

Таблица 2.3 – Функциональные возможности пользователя с ролью «ученик»

|  |  |  |
| --- | --- | --- |
| № | Вариант использования | Пояснение |
| 3 | Просмотр списка курсов | Ученик просматривает все курсы и получать их подробную информацию. |
| 4 | Фильтрация курсов | Ученик может найти курс, отфильтровав курсы по тематике. |
|  |  |  |
| Продолжение таблицы 2.3 | | |
| № | Вариант использования | Пояснение |
| 5 | Поиск курсов | Ученик может найти курс путем ввода названия или ключевого слово в поле поиска. |
| 6 | Подписка на курс | После просмотра подробной информации по курсу, ученик может на него подписать и начать проходить обучение. |
| 7 | Прохождение разделов курса | После подписки на курс, ученику открываются разделы курсов, при прохождении которых он последовательно и поэтапно получает информацию по теме. |
| 8 | Просмотр прогресса по курсам | После прохождения разделов курса, пользователь может получить информацию о том, сколько процентов курса он прошёл. |
| 9 | Создание отзыва | После получения всей информации курса, ученик может оставить отзыв, который содержит комментарий и оценку курса по пятибалльной шкале. |
| 10 | Удаление отзыва | Отзыв можно удалить в случае необходимости, это действие изменит значение рейтинга курса. |
| 11 | Редактирование профиля | Ученик может редактировать профиль, что включает в себя возможность изменить своё имя в системе, либо загрузить фотографию. |
| 12 | Удаление профиля | Ученик имеет возможность удалять свою учетную запись. |

Пользователь с ролью «учитель» имеет возможность создания, изменения и удаления курсов. Все функциональные возможности пользователя с ролью «учитель» представлены в таблице 2.4.

Таблица 2.4 – Функциональные возможности пользователя с ролью «учитель»

|  |  |  |
| --- | --- | --- |
| № | Вариант использования | Пояснение |
| 13 | Создание курса | Учитель может создать курс, заполнив все необходимые поля. |
| 14 | Изменение курса | После создания курса, учитель может изменять все данные о курсе. |
| 15 | Удаление курса | Учитель может удалить курс. |
| 16 | Создание раздела курса | Учитель может создавать разделы курса, заполняя все необходимые поля для создания. |
| 17 | Изменение раздела курса | После создания раздела курса учитель может изменить любые добавленные им ранее данные. |
| 18 | Удаление раздела курса | Учитель может удалить раздел курса. |
|  |  |  |
| Продолжение таблицы 2.4 | | |
| № | Вариант использования | Пояснение |
| 19 | Изменение статуса курса | В любой момент времени учитель может сделать курс не активным, что делает его не видимым для всех учеников. |
| 20 | Изменение статуса раздела курса | В любой момент времени учитель может сделать раздел курса не активным, что делает его не видимым для всех учеников. |
| 21 | Просмотр статистики курса | Учитель может посмотреть информацию для каждого своего созданного курса о том, сколько учеников на него подписаны. |

Функциональные возможности пользователя с ролью «администратор» представлены в таблице 2.5.

Таблица 2.5 – Функциональные возможности пользователя с ролью «администратор»

|  |  |  |
| --- | --- | --- |
| № | Вариант использования | Пояснение |
| 22 | Создание учетной записи учителя | Администратор может создать учетную запись для учителя. |
| 23 | Изменение учетной записи учителя | В случае неверного заполнения имени учителя, его можно изменить. |
| 24 | Удаление учетной записи учителя | Администратор может удалить учетную запись учителя в случае необходимости. |
| 25 | Удаление отзыва | Администратор может удалить отзывы |
| 26 | Блокировка курса | Администратор может сделать курс заблокированным, тем самым он будет не виден другим пользователям. |
| 27 | Просмотр статистики по курсам | Администратор может получать информацию о том, сколько учеников подписано на курсы |

Таким образом для каждой роли определен набор доступных действий и возможностей.

# Проектирование базы данных

Согласно диаграмме вариантов использования приложения была разработана схема базы данных, структура которой представлена в Приложении Б. База данных содержит семь таблиц, назначение таблиц базы данных представлено в таблице 2.6.

Таблица 2.6 – Назначение таблиц базы данных

|  |  |
| --- | --- |
| Таблица | Назначение |
| Users | Хранение информации о пользователях, включая их имя, ссылку на фотографию, пароль, почту и роль. |
| Courses | Хранение информации об обучающих курсах. |
| Продолжение таблицы 2.6 | |
| Таблица | Назначение |
| CourseParts | Хранение информации о каждом разделе курса. |
| Progresses | Хранение информации о прогрессе пользователей в изучении курса. |
| Reviews | Хранение отзывов пользователей для курсов. |
| Tags | Хранение ключевых слов, по которым будет осуществляться поиск курсов. |
| CourseTags | Хранение связи курсов с ключевыми словами. |

Таблица Users содержит информацию о пользователях приложения. Описание полей таблицы представлено в таблице 2.7.

Таблица 2.7 – Структура таблицы Users

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| Id | uuid | Идентификатор пользователя |
| Username | varchar | Имя пользователя |
| Email | varchar | Email пользователя |
| Password | varchar | Пароль пользователя |
| Role | varchar | Роль пользователя |
| ImageUrl | varchar | Ссылка на фотографию пользователя |
| CreatedAt | varchar | Дата создания пользователя |
| UpdatedAt | varchar | Дата редактирования данных о пользователе |

Таблица Courses содержит информацию о курсах, включая заголовок, описание, ссылку на изображение, категорию и статус курса. Описание полей приведено в таблице 2.8.

Таблица 2.8 – Структура таблицы Courses

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| Id | uuid | Идентификатор курса |
| TeacherId | uuid | Идентификатор пользователя с ролью учитель, который создал курс |
| Title | varchar | Заголовок курса |
| Description | varchar | Описание курса |
| Topic | varchar | Категория, к которой относится курс |
| Status | varchar | Статус курса, значение которого может иметь только два значения: «public» или «private», которые изменяют видимость курса |
| Published | varchar | Поле показывает, в каком состоянии находится курс: черновой вариант курса, курс опубликован или не опубликован |
| Продолжение таблицы 2.8 | | |
| Название | Тип данных | Описание |
| ImageUrl | varchar | Ссылка на фотографию курса |
| AverageRating | float | Средний рейтинг курса |
| CreatedAt | varchar | Дата создания курса |
| UpdatedAt | varchar | Дата редактирования курса |

Таблица CourseParts содержит информацию о разделах курса. Описание полей приведено в таблице 2.9.

Таблица 2.9 – Структура таблицы CourseParts

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| Id | uuid | Идентификатор раздела курса |
| CourseId | uuid | Идентификатор курса, с которым связан раздел |
| Title | varchar | Заголовок раздела курса |
| Content | varchar | Контент раздела курса |
| MediaUrl | varchar | Ссылка на изображение или видео |
| Status | varchar | Статус раздела курса, значение которого может иметь только два значения: «active» или «inactive», которые изменяют видимость раздела курса |
| Order | integer | Поле показывает порядок раздела в рамках курса |
| CreatedAt | varchar | Дата создания раздела курса |
| UpdatedAt | varchar | Дата редактирования раздела курса |

Таблица Progresses содержит информацию о прогрессе пользователя. Описание полей приведено в таблице 2.10.

Таблица 2.10 – Структура таблицы Progresses

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| Id | uuid | Идентификатор прогресса |
| CourseId | uuid | Идентификатор курса, с которым связан прогресс |
| UserId | uuid | Идентификатор пользователя, с которым связан прогресс |
| Progress | float | Значение прогресса в процентах |
| CompletedParts | array | Номера частей, пройденных пользователем в рамках курса |
| CreatedAt | varchar | Дата создания прогресса |
| UpdatedAt | varchar | Дата редактирования прогресса |

Таблица Reviews содержит комментарии, написанные пользователями к курсам. Описание полей приведено в таблице 2.11.

Таблица 2.11 – Структура таблицы Reviews

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| Id | uuid | Идентификатор отзыва |
| CourseId | uuid | Идентификатор курса, с которым связан отзыв |
| UserId | uuid | Идентификатор пользователя, который оставил отзыв |
| Rating | float | Оценка курса пользователем |
| Comment | varchar | Текст отзыва |
| CreatedAt | varchar | Дата создания отзыва |

Таблица Tags содержит теги, или, можно назвать, ключевые слова, по которым можно удобно производить поиск курса. Описание полей приведено в таблице 2.12.

Таблица 2.12 – Структура таблицы Tags

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| Id | uuid | Идентификатор тега |
| Name | varchar | Значение тега |

Таблица CourseTags содержит связь тега с курсом. Описание полей приведено в таблице 2.13.

Таблица 2.13 – Структура таблицы CourseTags

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| TagId | uuid | Идентификатор тега |
| CourseId | uuid | Идентификатор курса |

Назначение связей приведено в таблице 2.14.

Таблица 2.14 – Назначение связей между таблицами

|  |  |  |
| --- | --- | --- |
| Таблица источник | Связанная таблица | Тип связи |
| User | Course | Один ко многим |
| User | Progress | Один ко многим |
| User | Review | Один ко многим |
| Course | CoursePart | Один ко многим |
| Course | Progress | Один ко многим |
| Course | Review | Один ко многим |
| Course | CourseTag | Один ко многим |
| Tag | CourseTag | Один ко многим |
| CourseTag | Course | Многие к одному |
| CourseTag | Tag | Многие к одному |

Эти поля обеспечивают целостность данных и позволяют эффективно управлять информацией, связанной с пользователями, курсами и их прогрессом. Каждая таблица имеет четко определенные поля, отражающие определенные аспекты работы web-приложения.

## **Архитектура web-приложения**

Приложение разработано на основе клиент-серверной архитектуры, что обеспечивает четкое разделение логики и способствует масштабируемости системы.

Клиентская часть (frontend) реализована с использованием библиотеки React [7], обеспечивающей компонентный подход к построению пользовательского интерфейса. Обмен данными между клиентом и сервером осуществляется с помощью библиотеки axios [8], посредством HTTP-запросов. Полученные данные отображаются в интерфейсе в реальном времени, что делает взаимодействие с системой более отзывчивым и наглядным.

Серверная часть (backend) реализована на платформе Node.js с использованием легковесного и гибкого фреймворка Express [9], что позволило эффективно организовать маршрутизацию и обработку API-запросов. Для работы с базой данных применяется ORM Sequelize [10], обеспечивающая удобный механизм взаимодействия с реляционной СУБД и абстракцию над SQL-запросами.

В качестве системы управления базами данных используется PostgreSQL, предназначенная для хранения информации о курсах, пользователях, отзывах, прогрессе и других сущностях. Обработка операций чтения, записи, обновления и удаления данных реализована с использованием Sequelize-моделей.

Для хранения и обработки мультимедийных файлов (изображений курсов, аватаров и др.) интегрировано облачное хранилище Cloudinary [11]. Загрузка файлов на сервер осуществляется через middleware Multer [12], после чего изображения отправляются в облако, и клиент получает ссылку для дальнейшего использования.

Архитектура приложения представлена в Приложении В.

Таким образом, клиент-серверное взаимодействие осуществляется по следующей схеме: клиент инициирует запрос, сервер обрабатывает его, взаимодействует с базой данных и, при необходимости, с облачным хранилищем, формирует JSON-ответ и возвращает его на клиентскую сторону для дальнейшего отображения данных.

## **Выводы по разделу**

В рамках раздела была рассмотрена функциональность web-приложения, предназначенная для всех ролей: гостя, ученика, учителя и администратора. Общее количество функций, которые должны быть реализованы в web-приложении составляет 27. Рассмотрена логическая схема база данных web-приложения, которая включает 7 таблиц. В таблицах хранятся данные о пользователях, курсах, разделах курса, прогрессе пользователей и другое. Также рассмотрена архитектура приложения.

# Реализация web-приложения

# 3.1 Программная платформа NodeJS

Для серверной части проекта была использована платформа Node.js с использованием фреймворка Express, которая обеспечивает высокую производительность и гибкость при разработке web-приложений.

В рамках проекта Express используется для создания RESTful API, что позволяет эффективно реализовывать взаимодействие между клиентской и серверной частью приложения.

Express предоставляет удобные механизмы для маршрутизации, валидации запросов и подключения middleware-компонентов, что значительно упрощает разработку и настройку серверной части. Благодаря поддержке внедрения зависимостей через модульную структуру и интеграции с ORM Sequelize, платформа позволяет упростить работу с базой данных, обеспечить масштабируемость приложения и ускорить процесс разработки и тестирования серверной логики.

# Система управления базами данных PostgreSQL

Для работы web-приложения используется PostgreSQL – мощная, надёжная и масштабируемая система управления базами данных с открытым исходным кодом. PostgreSQL поддерживает сложные SQL-запросы, транзакции, а также работу с расширенными типами данных, включая JSON и ARRAY. Такая СУБД идеально подходит для современного web-приложения, обеспечивая высокую производительность и устойчивость к нагрузкам.

# Реализация базы данных

Для реализации базы данных в проекте применяется подход Code First с использованием ORM-библиотеки Sequelize. Это позволяет сначала определить структуру данных в виде моделей на языке JavaScript, а затем автоматически синхронизировать их с базой данных с помощью миграций или метода автоматической генерации схемы.

Конфигурация подключения к базе данных вынесена в отдельный модуль, где задаются параметры подключения (имя базы данных, имя пользователя, пароль, хост и порт). Для повышения гибкости и безопасности эти параметры загружаются из файла окружения .env с помощью библиотеки dotenv. Код подключения к базе данных приведён в листинге 3.1

|  |
| --- |
| import dotenv from 'dotenv';  import { Sequelize } from 'sequelize';  dotenv.config();  const sequelize = new Sequelize(process.env.DB\_NAME, process.env.DB\_USER, process.env.DB\_PASSWORD, {  host: process.env.DB\_HOST,  dialect: 'postgres',  logging: console.log,  });  export default sequelize; |

Листинг 3.1 – Код подключения базы данных

Модели базы данных описываются в отдельных файлах. Каждая модель соответствует таблице и определяет поля, типы данных, связи (например, «один ко многим», «многие ко многим») и ограничения. Примеры моделей: User, Course, Review, Progress, CoursePart и другие.

Модели регистрируются в основном файле и связываются друг с другом с помощью методов hasMany, belongsTo, hasOne, belongsToMany и т. д.

Инициализация моделей и их связей вынесена в отдельный модуль и происходит при запуске приложения. Это позволяет централизованно управлять схемой базы данных и использовать модели во всех частях серверного кода для выполнения операций CRUD (создание, чтение, обновление, удаление).

Для автоматического создания таблиц в базе данных на основе описанных моделей используется встроенный метод sequelize.sync(). Он выполняет синхронизацию структуры моделей с физической схемой базы данных – то есть создаёт все необходимые таблицы и связи, если они ещё не существуют.

Процесс синхронизации реализован в виде асинхронной функции syncDatabase, вызываемой при запуске приложения. Код функции представлен в листинге 3.2

|  |
| --- |
| const syncDatabase = async () => {  try {  await sequelize.sync();  console.log('База данных синхронизирована!');  } catch (error) {  console.error('Ошибка при синхронизации базы данных:', error);  }  }; |

Листинг 3.2 – Код подключения базы данных

Этот механизм особенно удобен при использовании подхода Code First, так как позволяет избежать написания SQL-скриптов вручную. После завершения синхронизации база данных готова к использованию, и запускается сервер приложения.

# Программные библиотеки

В процессе разработки серверной и клиентской частей приложения использовались сторонние программные библиотеки, позволяющие ускорить реализацию функциональности, упростить работу с базой данных, реализацию аутентификации, загрузку файлов, маршрутизацию, взаимодействие с API и управление состоянием. Библиотеки, используемые на стороне сервера, представлены в таблице 3.1.

Таблица 3.1 – Программные библиотеки серверной части

|  |  |  |
| --- | --- | --- |
| Библиотека | Версия | Назначение |
| Express | 4.21.2 | Фреймворк для создания HTTP-сервера и маршрутизации. |
| Sequelize | 6.37.5 | ORM-библиотека для взаимодействия с базой данных PostgreSQL. |
| Jsonwebtoken [13] | 9.0.2 | Библиотека для генерации и валидации JWT-токенов (аутентификация). |
| Bcrypt [14] | 5.1.1 | Хеширование паролей пользователей. |
| **Multer** | 1.4.5 | Обработка multipart/form-data при загрузке изображений. |
| **Multer-storage-cloudinary** | 4.0.0 | Хранение изображений на облачном сервисе Cloudinary. |
| **Cloudinary** | 2.6.0 | Работа с API Cloudinary. |
| **Express-validator** | 7.2.1 | Валидация и проверка данных на уровне маршрутов. |
| **Dotenv [15]** | 16.4.7 | Загрузка конфигурации из .env-файла. |
| Cors [16] | 2.8.5 | Обеспечение поддержки CORS для работы с клиентской части. |
| Uuid | 11.1.0 | Генерация уникальных идентификаторов. |

В процессе разработки клиентской части web-приложения были задействованы программные библиотеки, представленные в таблице 3.2.

Таблица 3.2 – Программные библиотеки клиентской части

|  |  |  |
| --- | --- | --- |
| Библиотека | Версия | Назначение |
| react | 19.0.0 | Основная библиотека для построения интерфейса. |
| react-router [17] | 7.2.0 | Реализация маршрутизации в приложении. |
| axios | 1.8.1 | Отправка HTTP-запросов к серверу. |
| @reduxjs/toolkit [18] | 2.6.0 | Инструменты для организации глобального состояния с использованием Redux. |
| react-redux [19] | 9.2.0 | Связывает Redux с компонентами React. |
| react-markdown [20] | 8.0.7 | Отображение разметки Markdown в компонентах React. |
| recharts | 2.15.3 | Построение графиков и визуализация статистики. |

Эти библиотеки обеспечивают устойчивую архитектуру REST API, безопасность при работе с данными и простоту поддержки проекта.

Использование перечисленных библиотек позволяет сформировать отзывчивый и функциональный пользовательский интерфейс, реализовать навигацию по страницам, визуализацию данных и обеспечить стабильное взаимодействие с серверной частью приложения.

# Структура серверной части

Структура серверной части приложения, реализованная на платформе Node.js с использованием фреймворка Express, включает в себя основные компоненты, необходимые для обработки HTTP-запросов, маршрутизации, работы с базой данных и обеспечения расширяемости архитектуры.

В таблице 3.3 приведён список директорий проекта разработки web-приложения и назначение файлов, хранящихся в этих директориях.

Таблица 3.3 – Директории серверной части проекта

|  |  |
| --- | --- |
| Директория | Назначение |
| Config | Содержит конфигурационные файлы приложения, такие как настройки базы данных и подключение к облачному хранилищу. |
| Controllers | Содержит контроллеры, обрабатывающие HTTP-запросы. Контроллеры вызывают соответствующие сервисы и формируют HTTP-ответы. |
| Middlewares | Содержит промежуточные обработчики для обработки ошибок, проверки JWT-токенов, валидации входных данных и других задач. |
| Models | Включает определения моделей ORM Sequelize, отражающих таблицы базы данных и их связи. |
| Routes | Определяет маршруты API и связывает их с соответствующими методами контроллеров. |
| Services | Реализует бизнес-логику приложения: обработку данных, вызовы к базе данных через модели, вспомогательные вычисления. |
| Uploads | Содержит файлы для правильной обработки загружаемых данных. |
| Utils | Содержит вспомогательные функции для валидации данных. |

На основе диаграммы вариантов использования были определены основные маршруты серверной части, соответствующие ключевым действиям пользователей. Контроллер UserController обрабатывает запросы, проводит валидацию данных, взаимодействует с моделью пользователя и возвращает соответствующие ответы в формате JSON. Таблица 3.4 содержит соответствие маршрутов, HTTP-методов и методов контроллера UserController, который реализует операции над данными таблицы User, включая регистрацию, аутентификацию, обновление профиля, а также функции, связанные с управлением учётными записями преподавателей.

Таблица 3.4 – Контроллер и функции маршрутов

|  |  |  |  |
| --- | --- | --- | --- |
| Метод | Маршрут | Контроллер | Метод контроллера |
| POST | /auth/register | UserController | register |
| POST | /auth/login | UserController | login |
| GET | /auth/me | UserController | getMe |
| DELETE | /auth/me | UserController | deleteUser |
| PATCH | /auth/me/username | UserController | updateUsername |
| PATCH | /auth/me/image | UserController | updateProfileImage |
| DELETE | /auth/me/image | UserController | deleteProfileImage |
| GET | /auth/admin-mode/teachers | UserController | getAllTeachers |
| POST | /auth/admin-mode/register-teacher | UserController | createTeacher |
| PATCH | /auth/admin-mode/teachername | UserController | updateTeachername |
| DELETE | /auth/admin-mode/teacher/:teacherId | UserController | deleteTeacherAdmin |

Контроллер CourseController отвечает за управление сущностью курса (course) в системе. Он реализует маршруты, обеспечивающие доступ к информации о курсах, а также позволяет преподавателям и администраторам выполнять действия по созданию, редактированию, удалению, публикации курсов, загрузке изображений и получению статистики. В таблице 3.5 представлено соответствие маршрутов, HTTP-методов и функций данного контроллера.

Таблица 3.5 – Контроллер и функции маршрутов

|  |  |  |  |
| --- | --- | --- | --- |
| Метод | Маршрут | Контроллер | Метод контроллера |
| GET | /courses/ | CourseController | getAllCourses |
| GET | /courses/search-filter | CourseController | getCoursesWithSearchFilter |
| GET | /courses/all-course/:id | CourseController | getCourseById |
| GET | /courses/details | CourseController | getCourseDetails |
| GET | /courses/inprogress | CourseController | getInProgressCourses |
| GET | /courses/completed | CourseController | getCompletedCourses |
| GET | /courses/teacher-mode | CourseController | getCoursesByTeacher |
| POST | /courses/teacher-mode | CourseController | createTeacherCourse |
| DELETE | /courses/teacher-mode/course/:id | CourseController | deleteCourse |
| PATCH | /courses/teacher-mode/text-update/:id | CourseController | updateTextFields |
| POST | /courses/teacher-mode/tags/:id | CourseController | linkTagsToCourse |
| PATCH | /courses/teacher-mode/published/:id | CourseController | updateCoursePublishedStatus |
| PATCH | /courses/teacher-mode/status/:id | CourseController | updateCourseStatus |
| Продолжение таблицы 3.5 | | | |
| Метод | Маршрут | Контроллер | Метод контроллера |
| PATCH | /courses/teacher-mode/imageUpload | CourseController | updateCourseImageUpload |
| PATCH | /courses/teacher-mode/image | CourseController | updateCourseImage |
| DELETE | /courses/teacher-mode/image | CourseController | deleteCourseImageUrl |
| GET | /courses/teacher-mode/analytics | CourseController | getTeacherCourseStatistics |
| GET | /courses/admin-mode/courses | CourseController | getAllCoursesForAdmin |
| PATCH | /courses/admin-mode/blocked/:id | CourseController | blockedCourse |

Контроллер CoursePartController отвечает за управление разделами курсов – добавление, удаление, изменение, сортировку и загрузку медиафайлов. В таблице 3.6 представлено соответствие маршрутов, HTTP-методов и функций данного контроллера.

Таблица 3.6 – Контроллер и функции маршрутов

|  |  |  |  |
| --- | --- | --- | --- |
| Метод | Маршрут | Контроллер | Метод контроллера |
| GET | /course-parts/ | CoursePartController | getCourseParts |
| POST | /course-parts/ | CoursePartController | createTeacherCoursePart |
| DELETE | /course-parts/:id | CoursePartController | deleteCoursePart |
| PATCH | /course-parts/:courseId/parts/order | CoursePartController | updateCoursePartsOrder |
| PATCH | /course-parts/teacher-mode/text-update/:courseId/:id | CoursePartController | updateTextFields |
| PATCH | /course-parts/teacher-mode/imageUpload | CoursePartController | updateCoursePartImageUpload |
| PATCH | /course-parts/teacher-mode/status/:id | CoursePartController | updateCoursePartStatus |
| PATCH | /course-parts/teacher-mode/image | CoursePartController | updateCoursePartImage |
| DELETE | /course-parts/teacher-mode/media | CoursePartController | deleteCoursePartMediaUrl |

Контроллеры ProgressController, ReviewController и TagsController обеспечивают работу с прогрессом пользователей по курсам, отзывами и тегами соответственно. ProgressController реализует логику отслеживания прогресса пользователя по курсам, подписки на курсы, обновления и получения статуса прохождения. ReviewController отвечает за добавление, получение и удаление отзывов к курсам как со стороны пользователя, так и администратора. TagsController обрабатывает создание тегов, которые могут использоваться для категоризации курсов.

Таблица 3.7 – Контроллеры и функции маршрутов

|  |  |  |  |
| --- | --- | --- | --- |
| Метод | Маршрут | Контроллер | Метод контроллера |
| GET | /progress/subscribe | ProgressController | subscribeToCourse |
| PATCH | /progress/ | ProgressController | updateProgress |
| GET | /progress/total-progress | ProgressController | getTotalProgress |
| GET | /progress/course-status | ProgressController | getCourseStatus |
| GET | /progress/ | ProgressController | getProgressForCourses |
| GET | /review/course | ReviewController | getReviewsByCourseId |
| GET | /review/user | ReviewController | getReviewsByUserIdAndCourseId |
| POST | /review/course | ReviewController | createReview |
| DELETE | /review/admin-mode/:reviewId | ReviewController | deleteReviewAdmin |
| DELETE | /review/:reviewId | ReviewController | deleteReview |
| POST | /tags/ | TagsController | createTag |

При передаче данных между клиентом и сервером используется формат JSON (JavaScript Object Notation). В контроллерах будет осуществляться взаимодействие с сервисами, которые инкапсулируют бизнес-логику приложения.

# Реализация функций пользователя с ролью «гость»

# Регистрация

Для гостя web-приложения доступна возможность регистрации. Данная функция позволяет пользователю создать учётную запись в системе и получить доступ к функционалу, требующему авторизации. Процесс регистрации реализован в методе register контроллера UserController. Реализация данного метода приведена в листинге 3.3.

|  |
| --- |
| export const register = async (req, res) => {  try {  const user = await User.findOne({ where: { email: req.body.email } });  if (user) {  return res.status(400).json({  message: 'Такой пользователь уже существует.',});}  const password = req.body.password;  const salt = await bcrypt.genSalt(10);  const hashedPassword = await bcrypt.hash(password, salt);  const newUser = await User.create({  username: req.body.username,  email: req.body.email,  password: hashedPassword,  role: req.body.role || 'student',  imageUrl: req.body.image\_url || '/img/default-user.svg',  });  const token = jwt.sign({ \_id: newUser.id, \_role: newUser.role }, process.env.JWT\_SECRET, {  expiresIn: '30d',  });  res.status(201).json({ message: 'Пользователь успешно зарегистрирован', userId: newUser.id, token: token, role: newUser.role,});  } catch (err) {  res.status(500).json({  message: 'Ошибка при регистрации пользователя',  });}}; |

Листинг 3.3 – Реализация метода Register

Перед выполнением основной логики регистрации HTTP-запрос проходит через middleware-функцию handleValidations, которая проверяет входные данные на соответствие установленным требованиям: формат электронной почты, наличие имени, корректность пароля и т.д. В случае успешного прохождения проверки, управление передаётся методу register.

Метод register выполняет проверку на наличие пользователя с указанным адресом электронной почты. Если пользователь уже существует, возвращается соответствующее сообщение об ошибке. В противном случае, осуществляется хэширование пароля с использованием библиотеки bcrypt, создаётся новая запись в базе данных, содержащая информацию о пользователе, и генерируется JWT-токен. После регистрации у пользователя будет роль «ученик».

Сформированный токен возвращается клиенту вместе с идентификатором пользователя и его ролью. Это позволяет автоматически авторизовать пользователя сразу после регистрации без необходимости повторного входа в систему.

# Аутентификация

Гость web-приложения имеет возможность пройти процедуру аутентификации, получив тем самым права на выполнение действий, соответствующих определённой роли в системе. Процесс аутентификации реализован в методе login контроллера UserController. Реализация данного метода представлена в листинге 3.4.

|  |
| --- |
| export const login = async (req, res) => {  const { email, password } = req.body;  try {  const user = await User.findOne({ where: { email } });  if (!user) return res.status(401).json({ message: 'Неверные учетные данные' });  const isMatch = await bcrypt.compare(password, user.password);  if (!isMatch) return res.status(401).json({ message: 'Неверные учетные данные' });  const token = jwt.sign({ \_id: user.id, \_role: user.role }, process.env.JWT\_SECRET, {  expiresIn: '30d',  });  res.status(200).json({  user: { id: user.id, email: user.email, role: user.role },  token: token,  message: 'Успешный вход',  });  } catch (err) {  res.status(500).json({ message: 'Ошибка при авторизации пользователя' });}}; |

Листинг 3.4 – Реализация метода Login

Перед выполнением основной логики аутентификации, данные, передаваемые в теле HTTP-запроса, проходят предварительную валидацию с помощью middleware-функции handleValidations. Этот промежуточный обработчик проверяет корректность введённых пользователем данных.

Далее осуществляется идентификация пользователя в системе: происходит поиск по адресу электронной почты, указанному в запросе. Если пользователь найден, хэш пароля, хранящийся в базе данных, сравнивается с паролем, переданным в запросе, с использованием библиотеки bcrypt. В случае совпадения генерируется JWT-токен, содержащий идентификатор и роль пользователя. Этот токен возвращается клиенту и используется для последующей авторизации при обращении к защищённым маршрутам приложения.

В дальнейшем, при попытке выполнения операций, требующих авторизации, все запросы проходят через middleware checkAuth, который извлекает и проверяет токен. При успешной проверке токена, в объект запроса добавляется информация о пользователе (в частности userId), что позволяет контроллерам выполнять действия от имени конкретного пользователя и проверять его права доступа.

# Реализация функций пользователя с ролью «ученик»

# Просмотр доступных курсов

Основной функциональной задачей клиентской части приложения является предоставление пользователю (ученику) доступа к учебным курсам. Для реализации данной функциональности на сервере был разработан метод getAllCourses контроллера CourseController, представленный в листинге 3.5.

|  |
| --- |
| export const getAllCourses = async (req, res) => {  const page = parseInt(req.query.page) || 1;  const limit = 8;  const offset = (page - 1) \* limit;  try {  const filteredCourses = await Course.findAll({  where: { status: 'public' },  include: [  {  model: CoursePart,  where: { status: 'active' }  },  ],  });  const courses = await courseService.getAllCourses( limit, offset);  const totalCourses = filteredCourses.length;  const totalPages = Math.ceil(totalCourses / limit);  res.status(200).json({  currentPage: page,  totalPages,  courses: courses,  });  } catch (err) {  console.error('Ошибка при получении данных о курсах: ', err);  res.status(500).json({ message: 'Внутренняя ошибка сервера' });  }  }; |

Листинг 3.5 – Реализация метода getAllCourses

Метод обеспечивает получение списка курсов, доступных для просмотра, с учётом условий фильтрации и пагинации. В частности, из базы данных извлекаются только те курсы, у которых поле status имеет значение «public», то есть они опубликованы и доступны для учеников. Также в выборку включаются активные разделы курсов (CoursePart), у которых поле status установлено в «active».

Метод поддерживает пагинацию – количество курсов на одной странице ограничено 8 элементами. Номер текущей страницы передаётся через параметр page запроса, а смещение (offset) и лимит (limit) рассчитываются автоматически. Общая численность курсов определяется отдельно, на её основе рассчитывается количество страниц (totalPages).

Помимо базового получения курсов, пользователю предоставлена возможность выполнять поиск и фильтрацию по определённым критериям, а также сортировку результатов. Для этого реализован дополнительный метод getCoursesWithSearchFilter, который расширяет функциональность базового метода getAllCourses, представлен в Приложении Г.

Поиск может выполняться по названию курса либо по тегам, связанным с курсом. Поисковый запрос поступает в виде параметра query. При фильтрации учитывается также тема курса (topic), если она указана. Запросы обрабатываются с помощью конструкций Sequelize.where, Op.or и подзапросов через Sequelize.literal для фильтрации по тегам. Все запросы чувствительны к регистру, что обеспечивается использованием функции LOWER.

Дополнительно, в выборке используются модели Tag, User (для отображения автора) и Progress (чтобы при необходимости отобразить прогресс текущего пользователя). Курсы сортируются по убыванию среднего рейтинга (averageRating) и по дате создания (createdAt), что позволяет выводить сначала самые популярные и новые курсы.

# Подписка и прохождение курсов

Пользовательская часть приложения предусматривает не только просмотр курсов, но и полноценное прохождение учебного материала. Для этого реализован функционал подписки на курс и отслеживания прогресса его прохождения.

Перед тем как начать обучение, пользователь должен подписаться на выбранный курс. Это реализовано через метод subscribeToCourse, представленный в листинге 3.6.

|  |
| --- |
| export const subscribeToCourse = async (req, res) => {  const userId = req.userId;  const courseId = req.query.courseId;  try {  const existingProgress = await Progress.findOne({  where: { userId, courseId },  });  if (existingProgress) {  return res.status(400).json({ message: 'Вы уже подписаны на этот курс.' });  }  const progress = await Progress.create({  userId,  courseId,  progress: 0,  completedParts: [],  });  return res.status(201).json(progress);  } catch (error) {  console.error('Ошибка при подписке на курс:', error);  return res.status(500).json({ message: 'Ошибка сервера' });  }  }; |

Листинг 3.6 – Реализация метода subscribeToCourse

Данный метод сохраняет информацию о новой записи в таблице прогресса (Progress) с привязкой к конкретному пользователю и курсу. Также устанавливаются начальные параметры: прогресс равен 0%, список завершённых разделов пустой. Повторная подписка на уже активный курс запрещена, что проверяется через поиск существующей записи.

Это позволяет точно отслеживать прогресс пользователя и динамически обновлять его состояние в базе данных при прохождении курса. Таким образом обеспечивается индивидуальный контроль успеваемости, что важно для формирования статистики и выдачи итоговых результатов обучения. После подписки пользователь получает возможность проходить отдельные разделы курса. По мере завершения разделов вызывается метод updateProgress, реализация которого представлена в листинге 3.7.

|  |
| --- |
| export const updateProgress = async (req, res) => {  const userId = req.userId;  const { courseId, currentPartOrder } = req.body;  try {  const existingProgress = await Progress.findOne({  where: { userId, courseId },  });  if (!existingProgress) {  return res.status(404).json({ message: 'Запись о прогрессе не найдена.' });}  if (!existingProgress.completedParts) {  existingProgress.completedParts = [];  }  if(!existingProgress.completedParts.includes(currentPartOrder)){  existingProgress.completedParts = [...existingProgress.completedParts, currentPartOrder];}  const totalParts = await CoursePart.count({ where: { courseId, status: 'active' } });  const completedPartsCount = existingProgress.completedParts.length;  existingProgress.progress = Math.round((completedPartsCount / totalParts) \* 100);  await existingProgress.save();  return res.status(200).json(existingProgress);  } catch (error) {  console.error('Ошибка при обновлении прогресса:', error);  return res.status(500).json({ message: 'Ошибка сервера' });  }  }; |

Листинг 3.7 – Реализация метода updateProgress

UpdateProgress обновляет данные прогресса. Метод добавляет номер пройденного раздела в массив completedParts (если он ещё не был добавлен ранее), и пересчитывает общий процент прохождения курса.

Процент прогресса рассчитывается на основе количества завершённых разделов относительно общего количества активных разделов курса. Обновлённое значение сохраняется в базе данных и использовано для отображения прогресса на клиентской стороне.

# Отслеживание текущего и завершенного обучения

После подписки и начала прохождения курса пользователю предоставляется возможность в любой момент получить список курсов, на которые он подписан, с учётом прогресса.

Данный функционал реализован в виде двух отдельных маршрутов: получения незавершённых курсов (прогресс от 0% до 99%) и завершённых курсов (прогресс 100%). Это позволяет гибко отображать информацию на клиентской стороне, разделяя текущее обучение и уже пройденные курсы.

Метод getInProgressCourses возвращает список курсов, на которые пользователь подписан, но ещё не завершил. Метод представлен в Приложении Д. В выборку попадают только те курсы, у которых значение прогресса меньше 100%. Также добавлена пагинация по 4 элемента на страницу, чтобы оптимизировать загрузку и отображение данных.

Метод getCompletedCourses аналогично возвращает все курсы, которые пользователь завершил, то есть прогресс которых равен 100%. Это позволяет пользователю отслеживать свои достижения и при необходимости пересматривать пройденный материал.

Таким образом, пользователь всегда имеет доступ к актуальной информации о своих учебных активностях и может быстро вернуться к любому курсу в процессе обучения или ознакомиться с ранее завершёнными материалами.

# Управление отзывами и расчет среднего рейтинга курса

По завершении курса пользователь может оставить отзыв, который включает в себя числовую оценку (от 0 до 5 баллов) и текстовый комментарий. Это позволяет другим пользователям ориентироваться на мнения студентов, а преподавателям – получать обратную связь. Каждый пользователь может создать отзыв на курс и удалить ранее оставленный отзыв.

Средняя оценка курса автоматически пересчитывается при добавлении или удалении отзыва. Для того чтобы рейтинг курса автоматически пересчитывался при добавлении нового отзыва, в модели Review реализован хук Sequelize afterCreate, реализация которого представлена в листинге 3.8.

|  |
| --- |
| Review.addHook('afterCreate', async (review, options) => {  const courseId = review.courseId;  const reviews = await Review.findAll({  where: { courseId },  attributes: ['rating'],  });  const totalRating = reviews.reduce((sum, r) => sum + r.rating, 0);  const averageRating = totalRating / reviews.length;  await Course.update({ averageRating }, { where: { id: courseId } });}); |

Листинг 3.8 – Реализация хука afterCreate

Внутри хука выполняется запрос к базе данных, который извлекает все отзывы, связанные с соответствующим курсом. Из каждого отзыва берётся только числовая оценка (rating), после чего с помощью метода reduce вычисляется суммарное значение всех оценок. Далее это значение делится на общее количество отзывов, что позволяет получить актуальное среднее значение рейтинга для курса. Полученное значение сохраняется в поле averageRating модели Course, что обеспечивает постоянную актуальность информации о качестве курса на основе пользовательских отзывов.

Для обеспечения актуальности средней оценки курса после удаления реализован Sequelize-хук afterDestroy, его реализация приведена в листинге 3.9.

|  |
| --- |
| Review.addHook('afterDestroy', async (review, options) => {  const courseId = review.courseId;  const remainingReviews = await Review.findAll({  where: { courseId },attributes: ['rating'],  });  const totalRating = remainingReviews.reduce((sum, r) => sum + r.rating, 0);  const averageRating = remainingReviews.length > 0 ? totalRating / remainingReviews.length : 0;  await Course.update({ averageRating }, { where: { id: courseId } });}); |

Листинг 3.9 – Реализация хука afterDestroy

Данный хук срабатывает сразу после удаления отзыва. Его основная задача – пересчитать средний рейтинг курса на основе оставшихся отзывов и обновить соответствующее поле в модели курса.

# Персонализация и удаление профиля пользователя

В разработанном приложении предусмотрена функциональность персонализации учетной записи пользователя, позволяющая учащемуся редактировать личные данные, а также при необходимости полностью удалить свой аккаунт.

Одним из ключевых элементов персонализации является возможность изменения имени пользователя (username). Эта операция сопровождается валидацией нового имени на соответствие установленным правилам (например, минимальная длина). При успешной проверке обновлённое имя сохраняется в базе данных, что обеспечивает пользователю гибкость в управлении своим профилем без необходимости создания новой учётной записи.

Дополнительно реализована возможность изменения фотографии профиля. Для загрузки и хранения изображений используется интеграция с облачным сервисом Cloudinary. Загрузка изображений осуществляется при помощи middleware multer в связке с CloudinaryStorage, который конфигурируется на использование специальной папки user\_images и поддерживает наиболее распространённые форматы изображений: .jpg, .jpeg, .png, .gif. Такая архитектура обеспечивает быструю и безопасную загрузку изображений, а также надёжное хранение с возможностью последующего доступа по URL.

Подключение облачного хранилища Cloudinary реализовано с использованием переменных окружения и представлено в листинге 3.10.

|  |
| --- |
| import cloudinary from 'cloudinary';  import dotenv from 'dotenv';  dotenv.config();  cloudinary.v2.config({  cloudinary\_url: process.env.CLOUDINARY\_URL,  });  export default cloudinary.v2; |

Листинг 3.10 – Подключение Cloudinary

Загрузка изображений реализована с использованием middleware multer, который обрабатывает входящие файлы. В связке с модулем multer-storage-cloudinary настраивается адаптированное хранилище, где каждый файл автоматически загружается в облачную папку user\_images. Код для настройки хранилища изображений пользователя представлен в листинге 3.11.

|  |
| --- |
| import multer from 'multer';  import { CloudinaryStorage } from 'multer-storage-cloudinary';  import cloudinary from '../config/cloudinary.js';  const storage = new CloudinaryStorage({  cloudinary: cloudinary,  params: {  folder: 'user\_images',  allowed\_formats: ['jpg', 'jpeg', 'png', 'gif'],  },  });  const upload = multer({ storage });  export default upload; |

Листинг 3.11 – Настройка хранилища изображений

Удаление аккаунта пользователя также реализовано в приложении и производится по инициативе самого пользователя. После подтверждения операции все данные об учетной записи, включая личную информацию, историю активности, удаляются из базы данных, тем самым обеспечивая соответствие принципам конфиденциальности и права на удаление данных.

# Реализация функций пользователя с ролью «учитель»

# Добавление, редактирование, удаление курсов

Web-приложение предоставляет пользователям с ролью «учитель» удобный и интуитивно понятный интерфейс для управления своими учебными курсами. Основной функционал включает создание, редактирование и удаление курсов, настройку их описания, категорий, тегов, а также загрузку изображений. Вся логика реализована стороне сервера с использованием асинхронных методов и обязательной проверкой прав доступа.

Добавление курса – это стартовый и обязательный этап в процессе создания образовательного контента. Для снижения порога входа и упрощения первого шага при работе с системой преподавателю предоставляется минимально необходимый функционал: возможность создать курс, указав лишь его название. Остальные сведения – такие как описание, категория, теги, изображение и структура материалов –должны быть добавлены позже через специализированный учебный редактор.

На серверной стороне за создание курса отвечает маршрут POST /courses/teacher-mode/, обработка которого осуществляется функцией createTeacherCourse, представленной в листинге 3.12.

|  |
| --- |
| export const createTeacherCourse = async (req, res) => {  const teacherId = req.userId;  const { title } = req.body;  if (!teacherId || !title) {  return res.status(400).json({ message: 'teacherId и title обязательны' });  }  try {  const newCourse = await courseService.createTeacherCourse(teacherId, title);  res.status(201).json(newCourse);  } catch (error) {  console.error('Ошибка при создании курса:', error);  res.status(500).json({ message: 'Ошибка при создании курса' });  }  }; |

Листинг 3.12 – Реализация метода createTeacherCourse

В начале метода осуществляется валидация входных данных. В случае отсутствия обязательных параметров (teacherId или title) возвращается HTTP-ответ с кодом 400 (Bad Request) и соответствующим сообщением об ошибке.

При успешной валидации вызывается метод createTeacherCourse из бизнес-логики (courseService), который выполняет непосредственное взаимодействие с базой данных – создаёт новую запись о курсе и связывает её с преподавателем.

В случае успешного создания курса клиент получает статус 201 (Created) и объект курса в формате JSON. При возникновении ошибок на сервере возвращается статус 500 (Internal Server Error).

Данный подход, при котором курс изначально создаётся с минимальным набором данных, обеспечивает гибкость и поэтапность наполнения.

Это позволяет преподавателю сперва сформировать общую структуру курса, а затем, по мере необходимости, дополнять его содержанием через учебный редактор.

Дополнительно такой подход упрощает обработку возможных ошибок и снижает риск потери данных, поскольку преподаватель может сохранять промежуточные версии курса на различных этапах создания. Это особенно актуально при работе с объемными курсами, состоящими из множества разделов и материалов.

После создания курса преподаватель может изменять ключевые текстовые поля курса: название, описание и тематику (категорию). Для этого реализован маршрут PATCH /courses/teacher-mode/text-update/:id, обрабатываемый функцией updateTextFields, представленной в листинге 3.13.

|  |
| --- |
| export const updateTextFields = async (req, res) => {  const teacherId = req.userId;  const { id } = req.params;  const { title, description, topic } = req.body;  try {  const updatedCourse = await courseService.updateTextFields(id, teacherId, {  title,  description,  topic,  });  if (!updatedCourse) {  return res.status(404).json({ message: 'Курс не найден' });  }  const course = await courseService.getCoursesByTeacherAfterUpdate(teacherId, id);  if (!course) {  return res.status(404).json({ message: 'Курс не найден' });  }  return res.status(200).json(course);  } catch (error) {  console.error('Ошибка при обновлении курса:', error);  return res.status(500).json({ message: 'Ошибка при обновлении курса' });  }  }; |

Листинг 3.13 – Реализация метода updateTextFields

Метод позволяет преподавателю гибко управлять содержимым курса. В теле запроса можно передавать одно или несколько из следующих полей: title – новое название курса; description – подробное описание содержания и целей курса; topic – тематика (категория), к которой относится курс.

Каждое поле является необязательным, что позволяет обновлять только те данные, которые нуждаются в изменении.

Для повышения визуальной привлекательности курсов и лучшего восприятия пользователями web-приложение предоставляет возможность прикрепления изображения к каждому курсу. Изображение может быть добавлено двумя способами: путём вставки прямой ссылки (URL) на изображение из интернета, либо путём загрузки файла с компьютера, который автоматически сохраняется в облачное хранилище Cloudinary, а затем сохраняется ссылка на это изображение в базе данных. Загрузка файла реализуется с помощью middleware Multer, а дальнейшая передача файла в облачное хранилище Cloudinary описана в разделе ранее. После успешной загрузки возвращается URL изображения, сохранённый в базе данных.

Для удобного поиска курсов учениками в платформе реализована возможность привязки тегов к каждому курсу. Теги представляют собой ключевые слова или фразы, которые описывают содержание курса, его тему или специфику. Это позволяет учащимся быстрее находить нужный курс. Метод linkTagsToCourse позволяет преподавателю указать, какие теги должны быть связаны с определённым курсом. Реализация данного метода представлена в листинге 3.14.

|  |
| --- |
| export const linkTagsToCourse = async (req, res) => {  const { id } = req.params;  const teacherId = req.userId;  const { tagIds } = req.body;  if (!id || !Array.isArray(tagIds)) {  return res.status(400).json({ message: 'Необходимы courseId и массив tagIds' });  }  try {  const updatedCourse = await courseService.linkTagsToCourse(id, teacherId, tagIds);  const course = await courseService.getCoursesByTeacherAfterUpdate(teacherId, id);  if (!course) {  return res.status(404).json({ message: 'Курс не найден' });  }  return res.status(200).json(course);  } catch (error) {  console.error('Ошибка при связывании тегов с курсом:', error);  res.status(500).json({ message: 'Ошибка при связывании тегов с курсом' });  }  }; |

Листинг 3.14 – Реализация метода linkTagsToCourse

Метод получает идентификатор курса и массив tagIds, содержащий идентификаторы существующих тегов. Затем он обновляет связи между курсом и соответствующими тегами в базе данных.

Перед привязкой тега к курсу преподаватель имеет возможность добавить новый тег. При добавлении тега система сначала выполняет проверку на наличие тега с таким именем в базе данных. Если тег уже существует, то из базы извлекается его идентификатор (ID), который впоследствии используется для привязки к курсу. Если же тег отсутствует, система создаёт новую запись в базе данных и возвращает ID вновь созданного тега. Таким образом, обеспечивается предотвращение дублирования тегов и сохранение целостности данных.

Удаление курса влечёт за собой каскадное удаление всех связанных данных, таких как разделы, прогресс учеников и отзывы, что предотвращает появление "висячих" записей в базе данных. Такой подход способствует поддержанию чистоты и согласованности данных в системе.

Преподаватель имеет возможность удалить созданный курс, если он больше не актуален или был создан ошибочно. Для этого реализован метод deleteCourse, представленный в листинге 3.15.

|  |
| --- |
| export const deleteCourse = async (req, res) => {  const { id } = req.params;  const userId = req.userId;  try {  const course = await courseService.findCourseById(id);  if (!course) {  return res.status(404).json({ message: 'Курс не найден' });  }  if (course.teacherId !== userId) {  return res.status(403).json({ message: 'У вас нет прав для удаления этого курса' });  }  await courseService.deleteCourseAndRelatedData(id);  return res.status(204).send();  } catch (error) {  console.error('Ошибка при удалении курса:', error);  return res.status(500).json({ message: 'Ошибка при удалении курса' });  }  }; |

Листинг 3.15 – Реализация метода deleteCourse

Метод принимает идентификатор курса (id) и идентификатор пользователя (userId), полученный из аутентификационного токена. Сначала происходит проверка существования курса в базе данных. Если курс не найден, сервер возвращает ответ с кодом 404. Далее осуществляется проверка прав доступа: курс может удалить только тот преподаватель, который его создал. При попытке удаления курса другим пользователем возвращается ошибка с кодом 403 (Forbidden). Если все условия выполнены, вызывается вспомогательная функция deleteCourseAndRelatedData, в которой последовательно удаляются все связанные с курсом данные: части курса (CoursePart), прогресс учащихся (Progress), связи с тегами (CourseTag), отзывы (Review), а затем и сам курс (Course).

# Добавление, изменение, удаление разделов курса

Разделы курса (части курса) являются структурными единицами, с помощью которых преподаватель может организовать содержание образовательного материала в логические блоки. В приложении предусмотрен функционал для добавления новых разделов курса, а также их удаления. Все действия доступны только автору курса (преподавателю) при условии прохождения аутентификации.

Для добавления нового раздела реализован метод createTeacherCoursePart, реализация которого представлена в листинге 3.16.

|  |
| --- |
| export const createTeacherCoursePart = async (req, res) => {  const teacherId = req.userId;  const { title } = req.body;  const { id } = req.query;  if (!teacherId || !title || !id) {  return res.status(400).json({ message: 'teacherId, courseId и title обязательны' });  }  try {  await coursePartService.createTeacherCoursePart(id, title, teacherId);  const course = await courseService.getCoursesByTeacherAfterUpdate(teacherId, id);  if (!course) {  return res.status(404).json({ message: 'Курс не найден' });  }  return res.status(200).json(course);  } catch (error) {  res.status(500).json({ message: 'Ошибка при создании раздела курса' });  }}; |

Листинг 3.16 – Реализация метода createTeacherCoursePart

Перед созданием раздела осуществляется базовая валидация. Далее вызывается метод сервиса createTeacherCoursePart, который добавляет запись в таблицу CoursePart, связывая её с соответствующим курсом. После успешного добавления, клиенту возвращается обновлённая информация о курсе, включая все его текущие разделы.

После создания раздела он изначально содержит только название – другие данные, такие как описание, фотография или видео на этапе создания не указываются. Такой подход обеспечивает модульную структуру, аналогичную работе с самим курсом: сначала создаётся основа (пустой контейнер), а затем через отдельные запросы добавляются или изменяются конкретные поля.

Таким образом, заполнение содержимого раздела осуществляется постепенно: каждое поле редактируется по мере необходимости, что позволяет гибко настраивать образовательный материал, не перегружая интерфейс создания избыточными параметрами. Это также упрощает управление контентом и позволяет динамически обновлять отдельные элементы раздела без необходимости пересоздания всей структуры.

Помимо создания и редактирования, в системе предусмотрена возможность удаления раздела курса. Данная операция доступна только автору курса (учителю), что проверяется через идентификатор пользователя. Удаление раздела выполняется на основании его идентификатора, после чего из базы данных удаляется соответствующая запись. Для повышения надёжности и предотвращения случайных удалений реализована дополнительная проверка прав доступа: перед выполнением операции система удостоверяется, что пользователь действительно является автором курса.

Особенностью реализации является то, что удаление раздела не ограничивается только удалением его метаданных. В случае, если курс уже начали проходить ученики, система автоматически пересчитывает прогресса для каждого из них.

Это необходимо, так как структура курса напрямую влияет на вычисление процента его прохождения. Если один или несколько разделов были удалены, общий объём доступных для изучения частей уменьшается, и, соответственно, прогресс учеников должен быть откорректирован, чтобы отразить актуальное состояние курса.

# Управление статусом курсов и их разделов

При создании нового курса он по умолчанию не отображается ученикам и находится в черновом режиме. Это отражается в его параметрах: значение поля published устанавливается как draft, а поле status принимает значение private, что означает приватный курс в стадии разработки.

Для того чтобы курс стал доступным для учеников, преподаватель должен заполнить все обязательные поля: название курса, его описание, категорию, а также прикрепить как минимум один активный раздел. Раздел считается активным, если заполнены его основные поля – название и описание. На этапе первоначального создания раздел имеет статус «неактивный», что позволяет преподавателю подготовить его содержание без немедленного отображения для пользователей.

После того как все необходимые данные внесены, статус поля published изменяется с draft на unpublished, что обозначает курс как завершённый, но ещё не опубликованный. На этом этапе преподаватель может провести финальную проверку и, при необходимости, внести корректировки.

Для изменения статуса курса реализованы два отдельных метода: updateCoursePublishedStatus и updateCourseStatus, которые позволяют преподавателю управлять доступностью курса. Первый метод отвечает за управление состоянием публикации (draft, unpublished, published), его реализация приведена в листинг 3.17.

|  |
| --- |
| export const updateCoursePublishedStatus = async (req, res) => {  const teacherId = req.userId;  const { id } = req.params;  const { published } = req.body;  try {  const course = await courseService.findCourseById(id);  if (!course) {  return res.status(404).json({ message: 'Курс не найден' });  }  if (course.teacherId !== teacherId) {  return res.status(403).json({ message: 'У вас нет прав изменять этот курс' });  }  await courseService.updatePublishedStatus(course, published);  const courseUpdated = await courseService.getCoursesByTeacherAfterUpdate(teacherId, id);  return res.status(200).json(courseUpdated);  } catch (err) {  res.status(500).json({ message: 'Ошибка при обновлении статуса опубликованности курса' });  }}; |

Листинг 3.17 – Реализация метода updateCoursePublishedStatus

Публикация курса переводит его в открытый доступ для учеников. Это сопровождается изменением значений полей: published становится published, а status –public. После этого курс становится видимым в пользовательском интерфейсе студентов и доступен для прохождения.

Второй метод отвечает за установку приватности (private, public). Данный метод представлен в листинге 3.18.

|  |
| --- |
| export const updateCourseStatus = async (req, res) => {  const teacherId = req.userId;  const { id } = req.params;  const { status } = req.body;  try {  const course = await courseService.findCourseById(id);  if (!course) {  return res.status(404).json({ message: 'Курс не найден' });  }  if (course.teacherId !== teacherId) {  return res.status(403).json({ message: 'У вас нет прав изменять этот курс' });  }  await courseService.updateStatus(course, status);  const courseUpdated = await courseService.getCoursesByTeacherAfterUpdate(teacherId, id);  return res.status(200).json(courseUpdated);  } catch (err) {  res.status(500).json({ message: 'Ошибка при обновлении статуса курса' });  }}; |

Листинг 3.18 – Реализация метода updateCourseStatus

При необходимости преподаватель может в любой момент скрыть курс от учеников, изменив его статус обратно на private. Это позволяет гибко управлять доступностью курсов и разделов в зависимости от текущих целей и готовности учебного материала.

Такая реализация обеспечивает проверку прав доступа, исключая возможность изменения статуса курса другими пользователями. Кроме того, обновлённая информация немедленно возвращается на клиентскую часть, позволяя мгновенно отобразить изменения в интерфейсе преподавателя. Это улучшает взаимодействие с системой и снижает вероятность ошибок при управлении курсами.

# Реализация функций пользователя с ролью «администратор»

# 3.9.1 Добавление, редактирование, удаление учетной записи учителя

Пользователь с ролью администратора имеет доступ к функциям управления учетными записями преподавателей. Для него реализованы три основные операции: создание, редактирование имени и удаление учетной записи преподавателя.

Для регистрации нового преподавателя используется метод createTeacher, представлен в листинге 3.19.

|  |
| --- |
| export const createTeacher = async (req, res) => {  try {  const user = await User.findOne({ where: { email: req.body.email } });  if (user) {  return res.status(400).json({  message: 'Такой пользователь уже существует.',  });  }  const password = req.body.password;  const salt = await bcrypt.genSalt(10);  const hashedPassword = await bcrypt.hash(password, salt);  const newUser = await User.create({  username: req.body.username,  email: req.body.email,  password: hashedPassword,  role: req.body.role || 'teacher',  imageUrl: req.body.image\_url || '/img/default-user.svg',  });  res.status(201).json({  message: 'Пользователь успешно зарегистрирован',  userId: newUser.id,  });  } catch (err) {  res.status(500).json({  message: 'Ошибка при регистрации пользователя',  });  }  }; |

Листинг 3.19 – Реализация метода createTeacher

Перед созданием происходит проверка на существование пользователя с таким же адресом электронной почты. Если пользователь с указанным email уже существует, система возвращает сообщение об ошибке. В случае отсутствия дубликатов, пароль преподавателя хешируется с использованием библиотеки bcrypt, после чего создается новая запись в базе данных. Учетной записи по умолчанию присваивается роль teacher, а также изображение пользователя (если не указано – используется стандартное изображение по умолчанию).

Если в учетной записи преподавателя была допущена ошибка при указании имени, администратор может изменить его с помощью функции updateTeachername. Метод получает идентификатор пользователя и новое имя, находит соответствующую запись в базе данных, обновляет поле username и сохраняет изменения.

Для полного удаления преподавателя реализована функция deleteTeacherAdmin. Она выполняет каскадное удаление всех данных, связанных с данным преподавателем. Вначале выбираются все курсы, созданные учителем. Для каждого курса удаляются связанные сущности: части курса (CoursePart), прогресс прохождения (Progress), отзывы (Review) и теги (CourseTag). Затем удаляются сами курсы, после чего удаляется и сам пользователь.

Таким образом, администратор может управлять учетными записями преподавателей.

# Удаление отзывов

В рамках административных функций предусмотрена возможность удаления комментариев (отзывов) пользователей. Это необходимо для поддержания качественной и безопасной среды платформы, особенно в случаях, когда комментарии содержат: ненормативную лексику, оскорбительные выражения, спам или рекламу.

Администратор, при просмотре списка комментариев, оставленных к курсам, и в случае выявления нарушений – удалить любой из них. При этом система автоматически пересчитывает среднюю оценку курса, исключая удалённый отзыв. Это позволяет сохранять актуальность рейтинга курсов и предотвращать его искажение.

Функциональность реализуется через вызов соответствующего метода на сервере, который удаляет отзыв из базы данных, а затем выполняет перерасчёт средней оценки на основе оставшихся отзывов. Такой подход обеспечивает как модерацию контента, так и корректное представление рейтинга курсов на платформе.

# Блокировка курсов

В рамках административных возможностей реализована функция блокировки курсов, нарушающих правила платформы. Эта функция предназначена для контроля качества и соответствия контента внутренним требованиям или стандартам образовательного ресурса.

Блокировка курса может быть выполнена администратором в случае выявления следующих нарушений: размещение нецензурного, оскорбительного или неприемлемого контента, нарушение авторских прав, предоставление недостоверной информации, если администратор его таковым посчитает.

Функция реализована в методе blockedCourse, он представлен в листинге 3.20.

|  |
| --- |
| export const blockedCourse = async (req, res) => {  try {  const userId = req.userId;  const { id } = req.params;  const user = await User.findOne({ where: { id: userId } });  if (!user || user.role !== 'admin') {  return res.status(403).json({ message: 'Доступ запрещен.' });  }  const course = await Course.findOne({ where: { id: id } });  if (!course) {  return res.status(404).json({ message: 'Course not found' });  }  course.published = course.published === 'blocked' ? 'published' : 'blocked';  course.status = course.published === 'blocked' ? 'private' : 'public';  await course.save();  res.json({ message: 'Course status updated', course });  } catch (error) {  console.error('Error toggling course status:', error);  res.status(500).json({ message: 'Server error' });  }  }; |

Листинг 3.20 – Реализация метода blockedCourse

При вызове метода выполняется проверка, имеет ли пользователь роль администратора. Если проверка проходит успешно, осуществляется переключение состояния курса: если курс был опубликован (published), он переводится в статус "заблокирован" (blocked), и его общий статус (status) меняется на private, что делает курс недоступным для обычных пользователей; если курс уже был заблокирован, администратор может разблокировать его, изменив значение обратно на published и статус на public.

Таким образом, только администратор может как заблокировать, так и разблокировать курс. Разблокировка возможна только в том случае, если курс был отредактирован и приведён в соответствие с требованиями платформы. Это позволяет эффективно управлять качеством контента и обеспечивает контроль доступа к образовательным материалам.

**3.10** **Структура клиентской части**

Клиентская часть приложения реализована с использованием компонентного подхода библиотеки React.

Основная логика и элементы пользовательского интерфейса размещены в директории src. Каждая директория содержит компоненты, отвечающие за отдельные разделы функциональности, такие как авторизация, просмотр курсов, управление профилем и т.д. Такой подход способствует удобной навигации по проекту и упрощает процесс поддержки и масштабирования приложения. Компоненты повторного использования оформлены в виде отдельных модулей, что позволяет минимизировать дублирование кода. Директории представлены в таблице 3.8.

Таблица 3.8 – Основные директории проекта в папке src и их назначение

|  |  |
| --- | --- |
| Директория | Назначение |
| Api | Включает в себя переиспользуемые React-компоненты, предназначенные для создания элементов пользовательского интерфейса web-приложения. |
| Components | Включает в себя переиспользуемые React-компоненты, из которых формируется пользовательский интерфейс. |
| Pages | Содержит компоненты, представляющие собой отдельные страницы приложения (например, главная страница, страница с деталями курса и т.д.). |
| Redux | Содержит файлы для управления глобальным состоянием приложения с использованием Redux: редьюсеры, экшены и конфигурация хранилища. |
| Scss | Содержит стили, написанные с использованием препроцессора SCSS. Используется для стилизации компонентов и страниц. |
| Utils | Включает вспомогательные функции и утилиты, которые используются в различных частях приложения для упрощения повторяющихся операций. |

Таблица соответствия маршрутов и компонентов страниц представлена в таблице 3.9.

Таблица 3.9 – Маршруты и компоненты страниц

|  |  |  |  |
| --- | --- | --- | --- |
| Компонент страницы | Маршрут | Роль | Назначение компонента |
| AllCoursePage | / | Ученик, учитель, администратор | Отображает список всех доступных курсов и служит начальной точкой для навигации по другим разделам системы. |
| AuthPage | /auth | Гость | Страница аутентификации и регистрации пользователей. |
| CoursePreviewPage | /course/:id | Ученик, учитель, администратор | Страница предварительного просмотра курса. Отображает дополнительную информацию о курсе и предоставляет возможность подписки на него. |
| SettingsPage | /settings/general | Ученик, учитель, администратор | Позволяет пользователю редактировать личные данные или удалить свою учетную запись. |
| Продолжение таблицы 3.9 | | | |
| Компонент страницы | Маршрут | Роль | Назначение компонента |
| CoursePartsPage | /course/:id/parts | Ученик, учитель, админи-стратор | Страница предоставляет навигацию по разделам определенного курса. |
| MyProgressPage | /my-progress | Ученик, учи-тель, админи-стратор | Страница отслеживания прогресса курсов, на которые пользователь подписан. |
| TeacherPage | /teacher-mode | Учитель | Страница преподавателя. Предоставляет возможность для создания и управления собственными курсами. |
| AdminPage | /admin-mode | Администратор | Страница администратора. Предоставляет возможность управлять преподавателями и курсами. |

В таблице 3.10 представлено описание основных компонентов приложения и их назначение.

Таблица 3.10 – Описание компонентов

|  |  |
| --- | --- |
| Компонент | Назначение |
| CourseCard | Отображает карточку курса с краткой, но ключевой информацией: названием, кратким описанием и изображением. |
| Header | Верхний колонтитул страницы. Содержит логотип приложения, строку поиска курсов и кнопки для навигации по основным разделам. |
| CoursePreview | Предоставляет расширенную информацию о выбранном курсе перед его подпиской. Служит для предварительного ознакомления. |
| AuthLogin | Отображает форму аутентификации пользователя с полями ввода электронной почты и пароля. |
| AuthRegister | Отображает форму регистрации нового пользователя с необходимыми полями ввода. |
| Navbar | Боковое навигационное меню для перехода между страницами приложения. |
| Settings | Обеспечивает функциональность управления учётной записью пользователя: редактирование данных и удаление аккаунта. |
| CoursePart | Отображает информацию об отдельном разделе курса: заголовок, содержимое и кнопку завершения раздела. |
| Review | Предоставляет форму для добавления и отправки отзывов о курсе, включает поле для текста и систему оценки. |

Помимо маршрутов и страниц, приложение включает множество компонентов, которые обеспечивают функциональность и удобство использования клиентской части. Компоненты и страницы реализуют все необходимые функции для различных ролей.

## **3.11 Выводы по разделу**

В web-приложении использовалась платформа Node.js с применением фреймворка Express, что позволило создать надёжный и масштабируемый сервер с удобной маршрутизацией и эффективной обработкой HTTP-запросов.

Для хранения данных применялась СУБД PostgreSQL, обеспечивающая надёжное и масштабируемое управление данными. Взаимодействие с базой данных осуществлялось с помощью ORM Sequelize, что упростило создание моделей и выполнение операций с данными. Также в приложении реализована система аутентификации и авторизации пользователей с использованием токенов доступа (JWT), что обеспечивает безопасное управление доступом к различным разделам платформы в зависимости от роли.

Структура приложения построена на модульном подходе, с разделением клиентской и серверной частей.

Кроме того, в процессе разработки особое внимание уделялось удобству взаимодействия пользователя с интерфейсом. Использование библиотеки React в сочетании с SCSS позволило создать современный и интуитивно понятный интерфейс, соответствующий актуальным требованиям UX/UI-дизайна.

Были разработаны и внедрены механизмы валидации данных как на клиенте, так и на сервере, что позволило повысить надёжность работы приложения и минимизировать количество ошибок, связанных с пользовательским вводом. Также предусмотрена обработка различных исключений и ошибок с отображением понятных уведомлений для пользователя.

Были реализованы все функции для различных ролей пользователей web-приложения: гостя, ученика, учителя и администратора. Общее количество реализованных функций составило 27, что охватывает полный функционал системы.

# Тестирование web-приложения

# 4.1 Функциональное тестирование

Для проверки корректности работы всех функций разработанного web-приложения было проведено ручное тестирование, описание и итоги которого представлены в таблице 4.1.

Таблица 4.1 – Описание тестирования функций web-приложения

|  |  |  |  |
| --- | --- | --- | --- |
| № | Функция | Описание тестирования | Ожидаемый результат |
| 1 | Регистрация | Под ролью «Гость» необходимо перейти на страницу регистрации (/auth/register), заполнить форму, указав имя пользователя (username), которое должно содержать от 3 до 12 символов, адрес электронной почты (email) и пароль (password), длиной от 5 до 18 символов. Затем следует нажать на кнопку «Зарегистрироваться». | В базу данных добавляется новый пользователь с ролью «Ученик». Пользователь перенаправляется на главную страницу (/). |
| 2 | Аутентификация | Под ролью «Гость» необходимо перейти на страницу входа (/auth/login), заполнить форму, указав адрес электронной почты (email) и пароль (password) существующего пользователя, затем нажать на кнопку «Войти». | Пользователь успешно проходит аутентификацию и перенаправляется на главную страницу (/). |
| 3 | Изменение имени пользователя | Под ролью «Ученик» необходимо перейти на страницу настроек (/settings/general), в соответствующем поле ввести новое корректное имя пользователя и нажать на кнопку «Сохранить». | Пользователь остаётся на странице настроек (/settings/general), имя пользователя обновляется как в базе данных, так и в пользовательском интерфейсе. |
| 4 | Удаление пользователя | Под ролью «Ученик» необходимо перейти на страницу настроек (/settings/general), нажать на кнопку «Удалить профиль» и подтвердить действие в модальном окне, нажав кнопку «Да, подтверждаю». | Пользователь получает роль «Гость», его данные удаляются из базы данных, и он перенаправляется на страницу входа (/auth/login). |
| 5 | Создание учителя | Под ролью «Администратор» необходимо перейти на страницу (/admin-mode), заполнить форму создания учителя, указав имя пользователя, email и пароль, после чего нажать на кнопку «Создать». | Пользователь остаётся на текущей странице, в списке отображается новая запись, в базу данных добавляется пользователь с ролью «Учитель». |

Продолжение таблицы 4.1

|  |  |  |  |
| --- | --- | --- | --- |
| № | Функция | Описание тестирования | Ожидаемый результат |
| 6 | Удаление учителя | Под ролью «Администратор» перейти на страницу (/admin-mode), в списке учителей нажать на кнопку удаления напротив нужного пользователя, затем подтвердить действие в модальном окне. | Пользователь остаётся на текущей странице. Удалённый учитель больше не отображается в списке. Все связанные с ним данные удаляются из базы данных. |
| 7 | Блокировка/разблокировка курса | Под ролью «Администратор» перейти на страницу (/admin-mode/courses), в списке курсов напротив нужного курса нажать на кнопку «Заблокировать» (или «Разблокировать» – если курс уже заблокирован). | Пользователь остаётся на текущей странице. В базе данных обновляются значения полей: published – на «blocked» и status – на «private». При разблокировке: published = «published», status = «public». |
| 8 | Создание курса учителем | Под ролью «Учитель» перейти на страницу (/teacher-mode), нажать кнопку «Новый курс». После этого пользователь перенаправляется на страницу создания курса (/teacher-mode/create), где следует указать название курса и нажать кнопку «Продолжить». | Пользователь перенаправляется на страницу редактирования курса (/teacher-mode/course/:id). В базе данных создаётся новая запись курса с начальными (базовыми) значениями. |
| 9 | Обновление курса | Под ролью «Учитель» перейти на страницу редактирования курса (/teacher-mode/course/:id), нажать на кнопку «Edit» напротив нужного элемента, внести изменения и нажать кнопку «Save». | Пользователь остаётся на текущей странице. Изменённое значение сохраняется в базе данных и отображается в пользовательском интерфейсе. |
| 10 | Удаление курса | Под ролью «Учитель» перейти на страницу редактирования курса (/teacher-mode/course/:id), нажать на кнопку удаления в правом верхнем углу. | Пользователь перенаправляется на страницу (/teacher-mode). Удалённый курс исчезает из списка. Все данные о курсе и его связи удаляются из базы данных. |
| 11 | Создание раздела курса | Под ролью «Учитель» на странице (/teacher-mode/course/:id), в блоке с разделами курса нажать кнопку «Add», в появившемся поле ввести название раздела и нажать кнопку «Save». | Пользователь остаётся на текущей странице. В списке разделов отображается новый раздел. В базе данных создаётся запись с начальными (базовыми) значениями. |

Продолжение таблицы 4.1

|  |  |  |  |
| --- | --- | --- | --- |
| № | Функция | Описание тестирования | Ожидаемый результат |
| 12 | Удаление раздела курса | Под ролью «Учитель» перейти на страницу (/teacher-mode/course/:id/:partId), в правом верхнем углу нажать на кнопку удаления. | Пользователь перенаправляется на страницу (/teacher-mode/course/:id). Удалённый раздел исчезает из списка. Все данные о разделе удаляются из базы данных. |
| 13 | Добавление изображение курса | Под ролью «Учитель» на странице (/teacher-mode/course/:id), нажать кнопку «Add» в блоке изображения. Открывается форма для вставки ссылки на фотографию. Вставить корректную ссылку и подтвердить. | Пользователь остаётся на текущей странице. Изображение обновляется. В базе данных сохраняется ссылка на изображение. |
| 14 | Загрузка изображения курса | Под ролью «Учитель» на странице (/teacher-mode/course/:id) нажать кнопку загрузки изображения. Открывается проводник для выбора файла. Выбрать изображение и подтвердить. | Пользователь остаётся на текущей странице. Изображение обновляется. В базе данных сохраняется ссылка на загруженное изображение. |
| 15 | Обновление статуса курса | Под ролью «Учитель» перейти на страницу (/teacher-mode/course/:id), в правом верхнем углу нажать кнопку «Опубликовать». | Пользователь остаётся на текущей странице. В базе данных поле status изменяется на значение «public». |
| 16 | Обновление статуса раздела курса | Под ролью «Учитель» перейти на страницу (/teacher-mode/course/:id/:partId), в правом верхнем углу нажать кнопку «Опубликовать». | Пользователь остаётся на текущей странице. В базе данных поле status изменяется на значение «active». На странице появляется кнопка «Скрыть» для обратного действия. |
| 17 | Подписка на курс | Под ролью «Ученик» перейти на страницу (/course/:id), нажать на кнопку «Начать обучение». | Пользователь перенаправляется на страницу (/course/:id/parts). В базе данных в таблице Progress создаётся новая запись о подписке на курс. |
| 18 | Обновление прогресса | Под ролью «Ученик» перейти на страницу (/course/:id/parts), на выбранном разделе нажать на кнопку «Завершить». | Пользователь перенаправляется на следующий раздел. В базе данных изменяется значение прогресса выбранного курса. |

Продолжение таблицы 4.1

|  |  |  |  |
| --- | --- | --- | --- |
| № | Функция | Описание тестирования | Ожидаемый результат |
| 19 | Создание отзыва | Под ролью «Ученик» перейти на страницу (/course/:id/parts), нажать на кнопку «Оставить отзыв», заполнить форму корректными значениями (текст и рейтинг), затем нажать кнопку «Сохранить отзыв». | Пользователь остаётся на текущей странице. В базе данных сохраняется новый отзыв. Добавленный отзыв отображается на странице. |
| 20 | Удаление отзыва (Ученик) | Под ролью «Ученик» перейти на страницу (/course/:id/parts), нажать на кнопку «Оставить отзыв», затем нажать на кнопку удаления в блоке ранее оставленного отзыва. | Пользователь остаётся на текущей странице. Отзыв удаляется из базы данных и исчезает с интерфейса. |
| 21 | Удаление отзыва (Администратор) | Под ролью «Администратор» перейти на страницу (/course/:id), нажать на кнопку удаления в блоке выбранного отзыва в списке отзывов курса. | Пользователь остаётся на текущей странице. Отзыв удаляется из базы данных и исчезает с интерфейса. |

После проведённого ручного тестирования всех основных функций web-приложения демонстрирует устойчивую работу и обеспечивает необходимую функциональность для администраторов, учителей и учащихся.

# 4.2 Выводы по разделу

В результате проведённого ручного тестирования были проверены ключевые функциональные возможности разработанного web-приложения. Результаты тестирования подтвердили, что все реализованные конечные точки корректно выполняют возложенные на них задачи и обеспечивают ожидаемое поведение в соответствии с требованиями технического задания. Обработка ошибок и валидация входных данных реализованы на должном уровне, что способствует повышению надёжности и устойчивости системы при работе с некорректными или неполными запросами.

# Руководство пользователя

# 5.1 Руководство для роли «гость»

При запуске приложения пользователь по умолчанию перенаправляется на страницу регистрации и аутентификации. На данной странице представлена форма, позволяющая либо создать новую учётную запись, либо выполнить вход в уже существующую. Внешний вид страницы аутентификации представлен на рисунке 5.1.

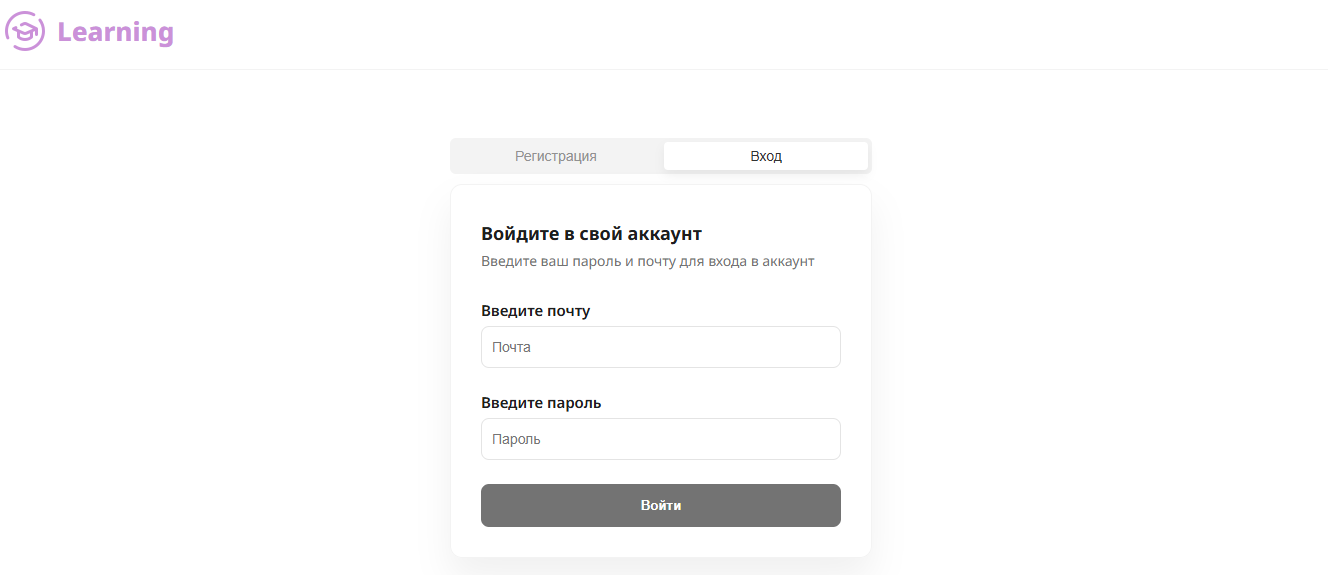


Рисунок 5.1 – Страница аутентификации

По умолчанию отображается форма входа в систему. Для переключения на форму регистрации необходимо нажать соответствующую кнопку, расположенную в верхней части формы. При регистрации требуется заполнить все обязательные поля: имя пользователя, адрес электронной почты и пароль. После заполнения формы следует нажать кнопку «Зарегистрироваться». В случае успешной валидации данных пользователь автоматически перенаправляется на главную страницу приложения.

Если у пользователя уже имеется учетная запись, необходимо воспользоваться формой входа: ввести электронную почту и пароль, после чего нажать кнопку «Войти». После успешного входа пользователь перенаправляется на главную страницу.

# 5.2 Руководство для роли «ученик»

После успешной аутентификации пользователь автоматически перенаправляется на главную страницу приложения. На данной странице отображаются все доступные курсы для прохождения обучения. Навигация по курсам упрощается за счёт панели фильтрации, расположенной в верхней части интерфейса, с помощью которой можно отфильтровать курсы по тематике. Также в верхней части страницы расположена поисковая строка, позволяющая производить поиск курсов по их названию или ключевым словам. Визуальное представление главной страницы приведено на рисунке 5.2.

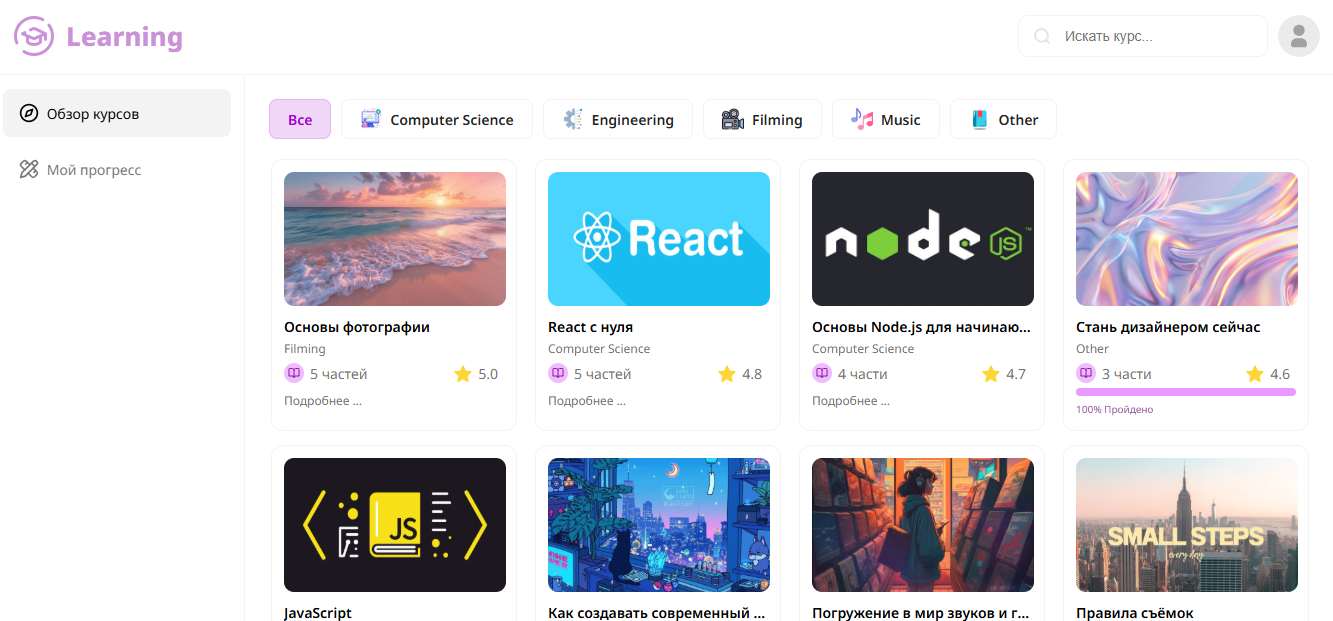


Рисунок 5.2 – Главная страница

При нажатии на карточку курса пользователь переходит на страницу, содержащую детальное описание курса. Эта страница представлена на рисунке 5.3.

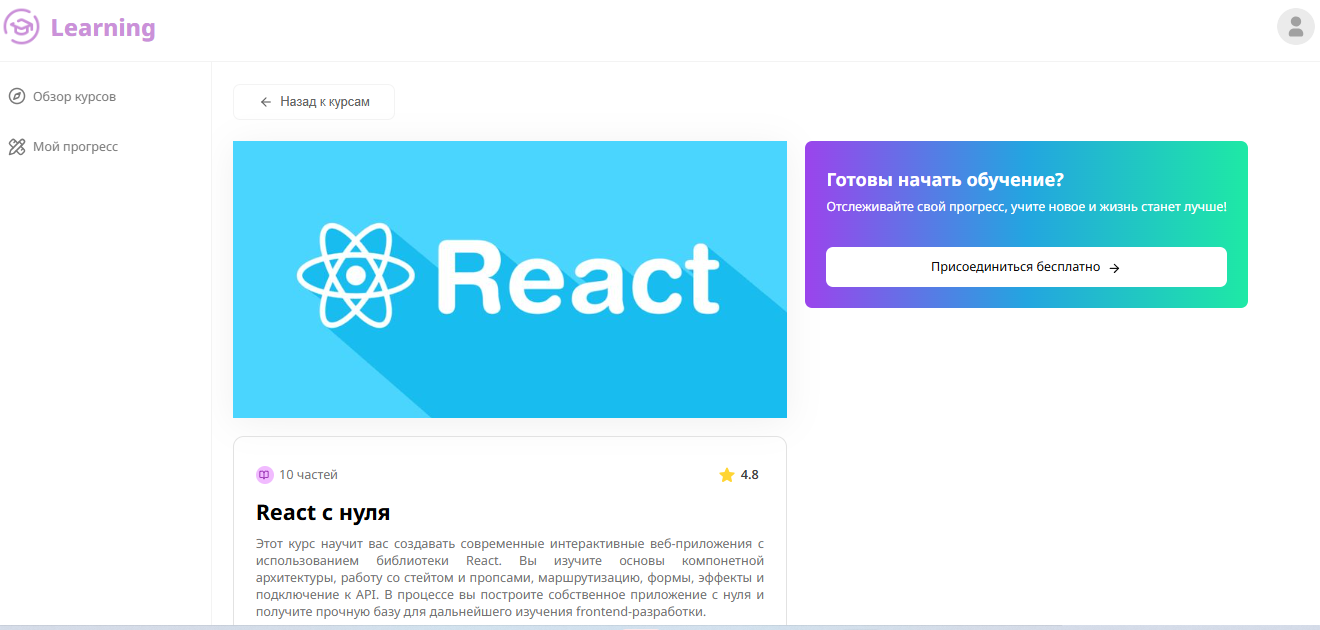


Рисунок 5.3 – Страница с детальным описанием курса

На странице описания пользователь может ознакомиться с подробной информацией о выбранном курсе, а также просмотреть комментарии, оставленные другими участниками. При желании пройти обучение пользователь может нажать кнопку подписки, расположенную справа на странице.

После нажатия на кнопку подписки пользователь перенаправляется на страницу с содержимым выбранного курса, включающим его тематические разделы. Визуальное представление данной страницы приведено на рисунке 5.4.

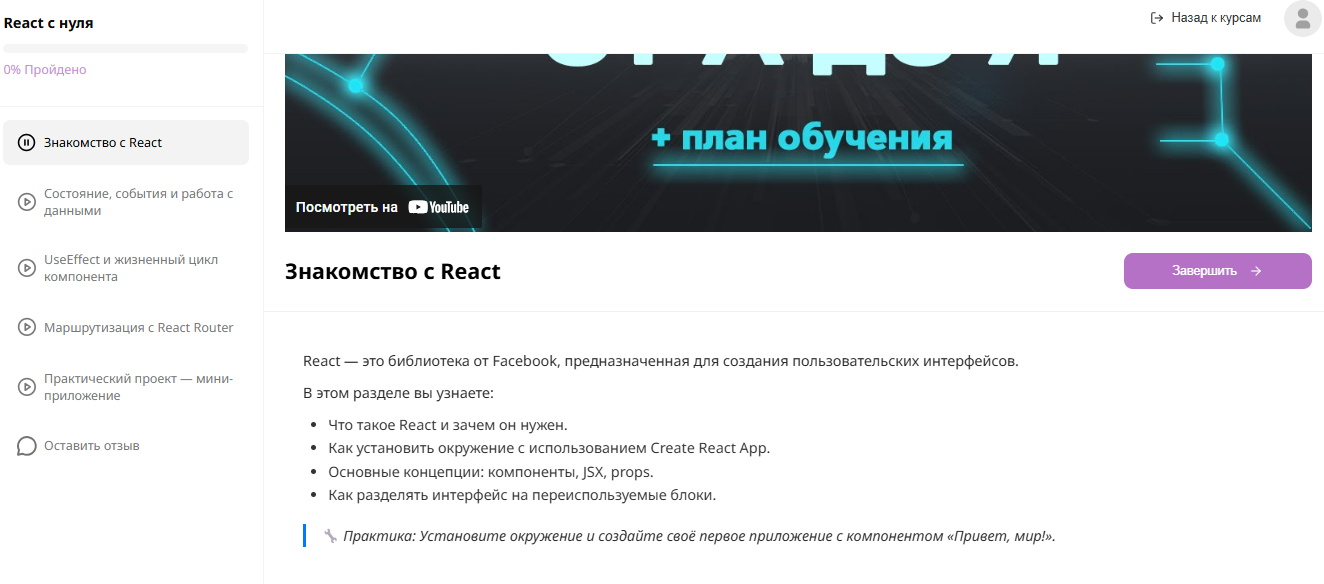


Рисунок 5.4 – Страница с разделами курса

На данной странице слева расположено навигационное меню, позволяющее переходить между разделами курса. В верхней части навигационной панели отображается индикатор прогресса, показывающий процент завершения курса. При нажатии на кнопку «Завершить» в рамках конкретного раздела, система помечает данный раздел как пройденный и автоматически обновляет процент завершённости курса. В нижней части навигационного меню размещена кнопка перехода на страницу отзывов, где пользователь может оставить собственный отзыв.

При нажатии на кнопку «Оставить отзыв» на странице с разделами курса пользователь переходит на страницу, содержащую форму для добавления либо удаления отзыва. Визуальное представление данной страницы приведено на рисунке 5.5.

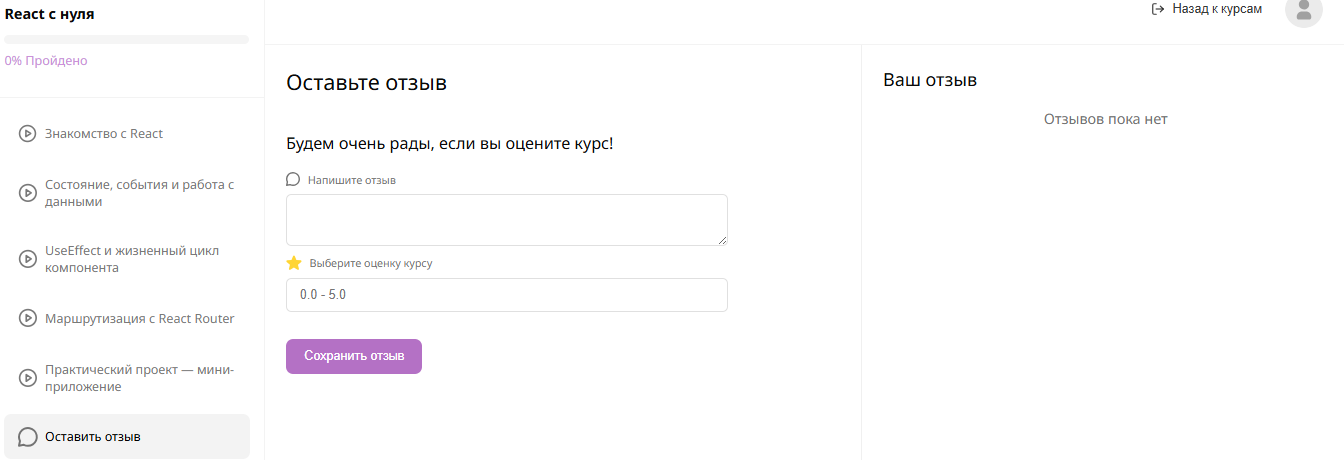


Рисунок 5.5 – Страница добавления отзыва

На данной странице пользователь может заполнить форму отзыва, указав текст отзыва и оценку курса. После заполнения всех обязательных полей отзыв сохраняется нажатием кнопки «Сохранить отзыв».Сохранённый отзыв отображается в правой части интерфейса.

При необходимости пользователь может удалить отзыв, нажав на значок корзины, расположенный в правом верхнем углу соответствующего блока отзыва.

Пользователю доступна страница, на которой отображаются только те курсы, на которые он подписан. Для перехода к данной странице необходимо вернуться на главную страницу и в левом навигационном меню выбрать кнопку «Мой прогресс». Визуальное представление страницы показано на рисунке 5.6.

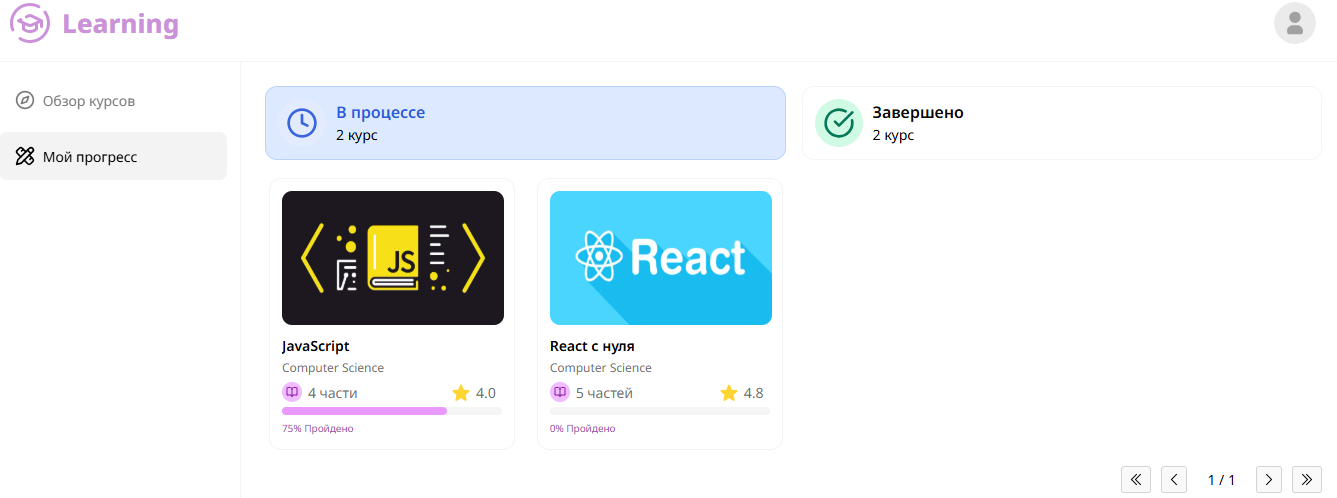


Рисунок 5.6 – Страница прогресса

На странице представлены карточки курсов, по которым пользователь в настоящее время проходит обучение. По умолчанию отображаются только курсы со статусом «в процессе прохождения», то есть не завершённые полностью. При нажатии на кнопку «Завершено» отображаются курсы, которые пользователь прошёл полностью.

Для изменения фотографии или имени пользователя предусмотрена отдельная страница настроек. Для перехода на неё необходимо нажать на кнопку в правом верхнем углу экрана, доступную на всех страницах приложения, за исключением страниц аутентификации и регистрации. Изначально эта кнопка содержит стандартное изображение профиля, одинаковое для всех пользователей. После загрузки или изменения фотографии пользователя кнопка отображает выбранное изображение.

При нажатии на кнопку открывается небольшое модальное окно с дополнительными опциями: кнопкой перехода к настройкам профиля и кнопкой выхода из аккаунта. Для изменения данных профиля следует выбрать кнопку «Настройки» в модальном окне.

Пользователь переходит на страницу настроек, содержащую форму для изменения имени и загрузки фотографии. Для замены изображения необходимо нажать на текущее изображение профиля – это может быть либо стандартное изображение, либо уже установленная фотография.

Кроме того, на данной странице пользователь имеет возможность удалить свой профиль, что приводит к удалению всех его данных из системы. Визуальное представление страницы настроек приведено на рисунке 5.7.

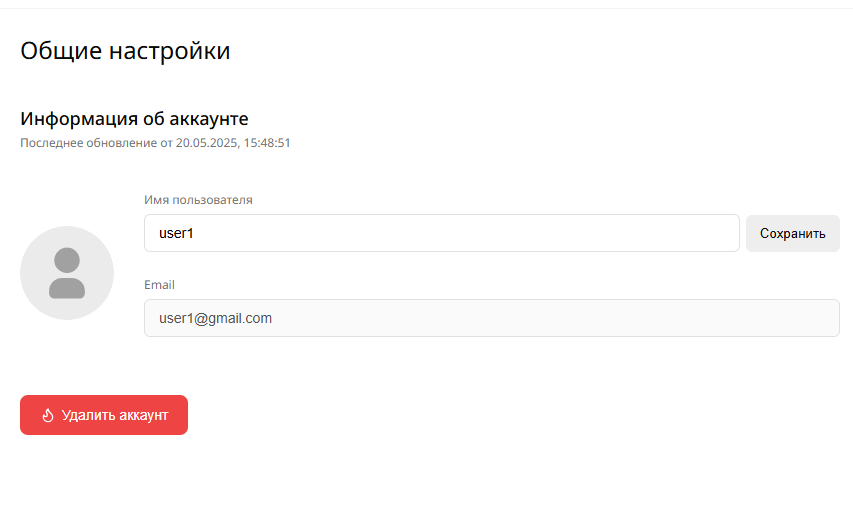


Рисунок 5.7 – Страница настройки профиля

Для изменения имени пользователь вводит новое значение в соответствующее поле и сохраняет изменения нажатием кнопки «Сохранить». В случае успешной валидации данные обновляются в системе.

Для удаления аккаунта следует нажать кнопку «Удалить аккаунт», после чего откроется модальное окно с подтверждением данного действия.

# 5.3 Руководство для роли «учитель»

Для пользователя с ролью «учитель», при открытии модального окна (вызываемого нажатием на кнопку в правом верхнем углу экрана), кроме стандартных опций «Настройки» и «Выход», дополнительно отображается кнопка «Учитель». При нажатии на данную кнопку пользователь перенаправляется на страницу, содержащую таблицу курсов, созданных им самостоятельно. Визуальное представление данной страницы приведено на рисунке 5.8.

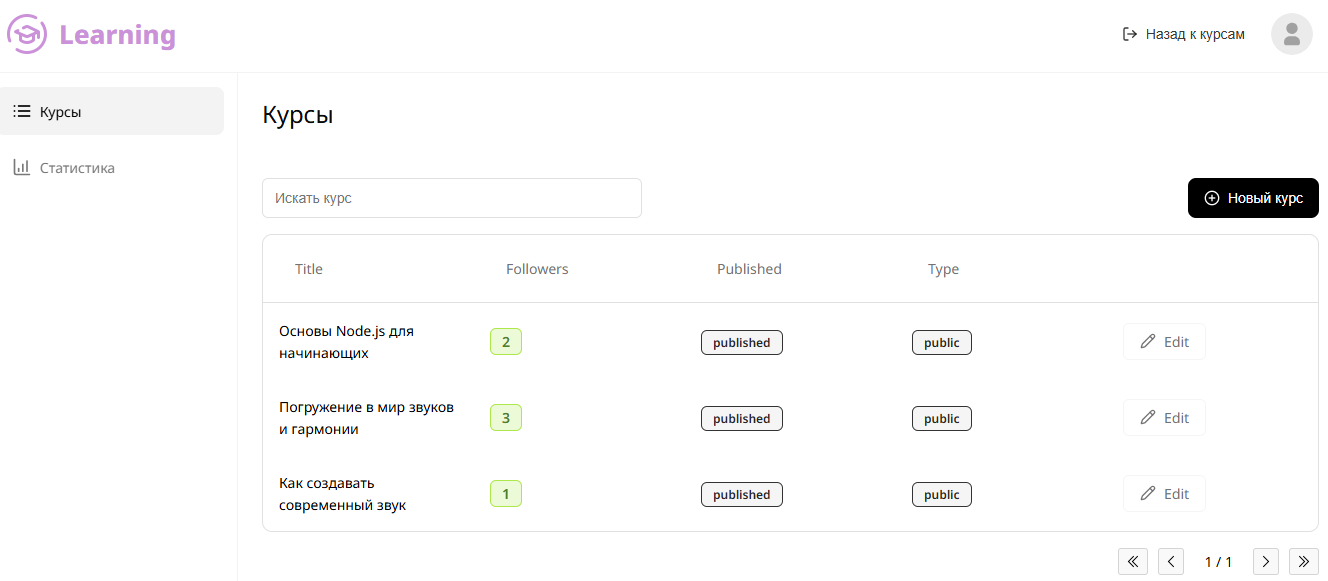


Рисунок 5.8 – Страница с курсами учителя

На этой странице отображаются все курсы, созданные текущим пользователем с ролью учителя. Для удобства навигации предусмотрена строка поиска по названию курсов. Каждая строка курса содержит кнопку «Edit», при нажатии на которую открывается страница детального редактирования всех параметров выбранного курса.

Для создания нового курса необходимо нажать на кнопку «Новый курс», после чего открывается форма ввода названия. При успешной валидации введённых данных происходит автоматический переход на страницу редактирования курса, где становится доступна подробная настройка его содержимого. Эта страница представлена на рисунке 5.9.

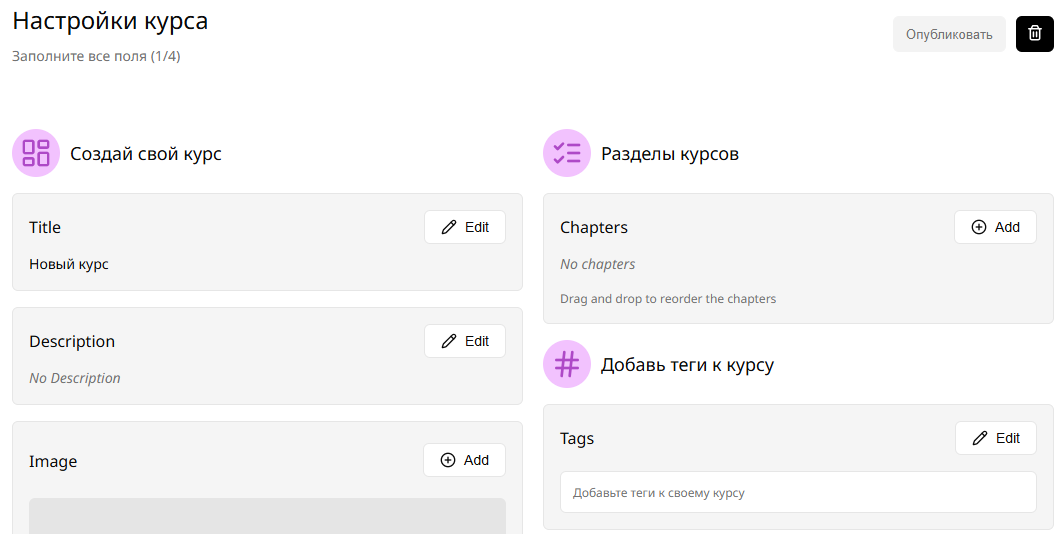


Рисунок 5.9 – Страница редактирования курса

Для того чтобы курс стал доступен другим пользователям, необходимо заполнить четыре обязательных поля. Для редактирования любого из этих полей следует нажать на кнопку «Edit», расположенную в соответствующем блоке. После нажатия появляется форма ввода и кнопка сохранения изменений.

После добавления разделов курса они отображаются в виде списка строк. В конце каждой строки располагается кнопка редактирования. При нажатии на эту кнопку пользователь перенаправляется на страницу детального редактирования выбранного раздела курса. Страница представлена на рисунке 5.10.

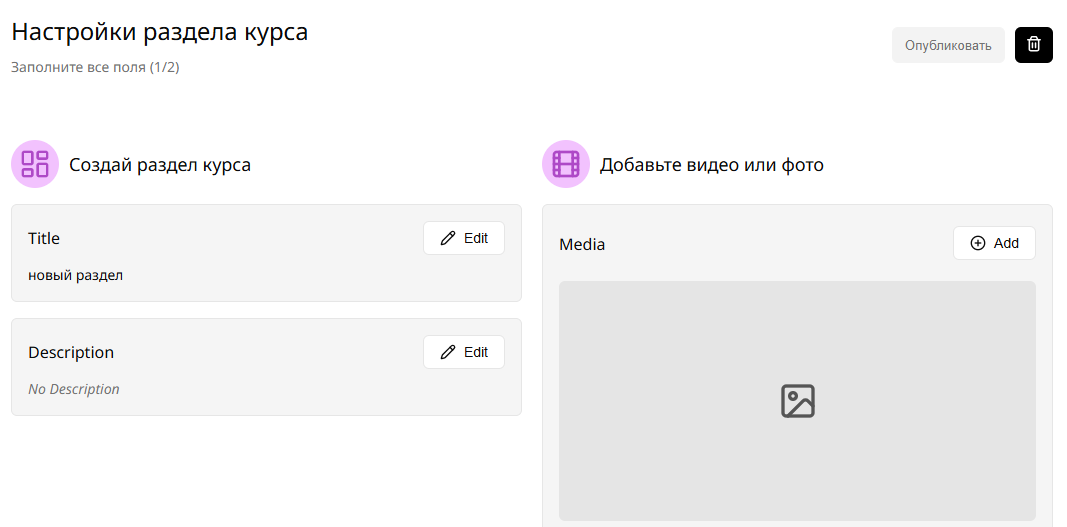


Рисунок 5.10 – Страница редактирования раздела курса

После заполнения всех необходимых полей следует нажать на кнопку «Опубликовать», после чего соответствующий раздел становится активным и доступным для использования. Аналогично, при заполнении всех обязательных полей курса, на странице редактирования необходимо нажать кнопку «Опубликовать», чтобы сделать курс общедоступным для всех пользователей.

После публикации как курса, так и отдельных его разделов, появляется возможность изменить их статус на приватный. Для этого предусмотрена соответствующая кнопка, позволяющая сделать курс или раздел недоступным для других пользователей.

Для учителя также доступна страница для просмотра статистики. Для этого нужно нажать на кнопку «Статистика» в левом навигационном меню.

# 5.4 Руководство для роли «администратор»

Для пользователя с ролью «администратор», при открытии модального окна (вызываемого нажатием на кнопку в правом верхнем углу экрана), кроме стандартных опций «Настройки» и «Выход», дополнительно отображается кнопка «Администратор». При нажатии на данную кнопку пользователь перенаправляется на страницу, содержащую таблицу учителей приложения. Визуальное представление данной страницы приведено на рисунке 5.11.

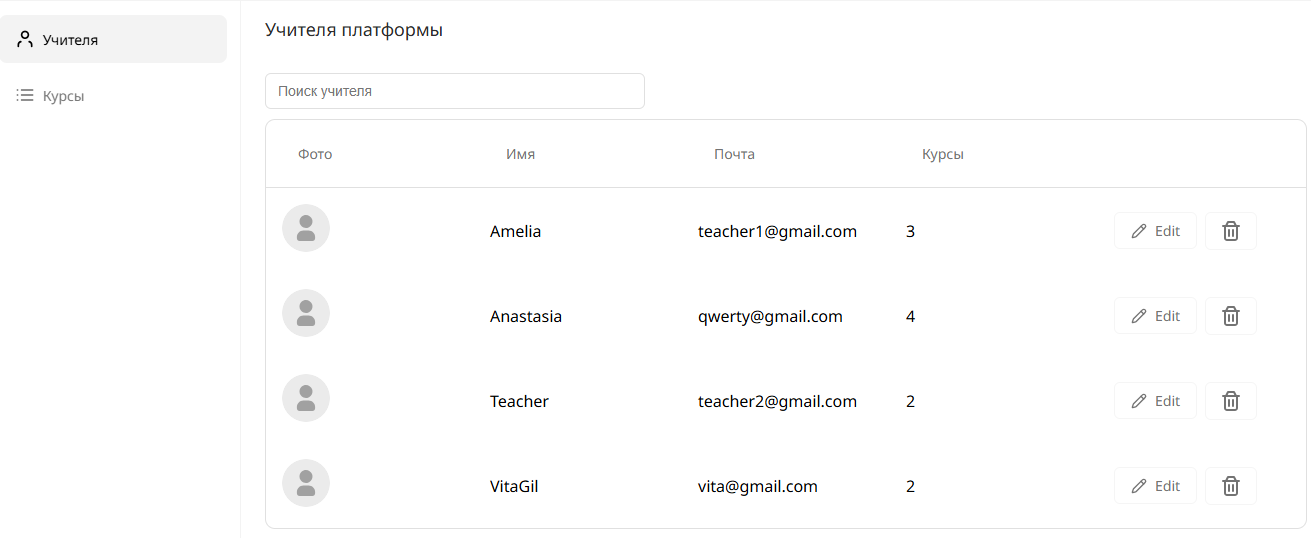


Рисунок 5.11 – Страница администратора

Администратор имеет возможность управлять списком преподавателей. Он может удалить преподавателя из списка или изменить его имя с помощью соответствующих кнопок, расположенных напротив каждой записи. Также для этой таблицы есть поиск по имени и почте учителя. Кроме того, на данной странице предусмотрена форма для добавления нового преподавателя.

При нажатии на кнопку «Курсы» осуществляется переход на страницу с таблицей всех публичных курсов, доступных в системе. На этой странице администратор может просматривать количество учеников, подписанных на каждый курс, а также при необходимости заблокировать курс, сделав его недоступным для пользователей. Разблокировка курса также доступна по кнопке в строке курса, если он был ранее заблокирован. Также для этой таблицы присутствует поисковая строка, которая осуществляет поиск по названию курса.

# 5.5 Выводы по разделу

Руководство пользователя охватывает весь функционал приложения. Реализованы простые процессы регистрации, авторизации и изменения профиля.

Пользователи с ролью «ученик» имеют доступ к основному функционалу приложения. Они могут просматривать список курсов, использовать поиск и фильтрацию по категориям, подписываться на курсы и проходить их по разделам. Ученик отслеживает свой прогресс, оставляет отзывы и оценки. Кроме того, доступна возможность редактирования и удаления собственного профиля. Все эти действия направлены на комфортное и последовательное прохождение образовательного контента на платформе.

Пользователи с ролью «учитель» обладают расширенными возможностями, предназначенными для создания и управления образовательным контентом. Учителя могут создавать курсы и наполнять их разделами, редактировать или удалять их при необходимости. Также предусмотрено управление статусом курсов и отдельных разделов – возможность временно скрывать материалы от учеников. Для анализа эффективности своих курсов учителя имеют доступ к статистике, отображающей количество подписанных учеников. Это позволяет улучшать содержание курсов и адаптировать обучение под нужды аудитории.

Пользователи с ролью «администратор» обладают полным доступом к управлению системой и контролю над всеми пользователями и контентом платформы. Администратор может создавать, редактировать и удалять учётные записи учителей, а также следить за контентом – удалять отзывы и блокировать нежелательные или нарушающие правила курсы. Также администратор имеет доступ к просмотру общей статистики по курсам, включая количество учеников, подписанных на каждый курс. Это обеспечивает поддержание порядка и высокого качества образовательного контента на платформе.

**Заключение**

В результате было разработано современное web-приложение для создания и прохождения онлайн-курсов. Основной целью проекта являлась реализация полнофункциональной платформы, обеспечивающей удобное взаимодействие между преподавателями и учащимися, а также предоставляющей инструменты для управления учебным контентом, отслеживания прогресса и оценки качества курсов.

Разработка велась с применением современных технологий: на серверной стороне использован Node.js с фреймворком Express и ORM-библиотекой Sequelize для взаимодействия с базой данных. Аутентификация и авторизация реализованы с использованием JWT (JSON Web Token). Клиентская часть построена на React, с применением Redux Toolkit для управления состоянием приложения, а также React Router для маршрутизации. Визуальный облик интерфейса был детально проработан с использованием SASS, что позволило достичь гибкости в стилизации и адаптивности интерфейса.

Web-приложение поддерживает четыре основные роли: гость, ученик, учитель и администратор, каждая из которых обладает собственным набором прав и функциональностью. Ученики могут регистрироваться, просматривать и проходить курсы, отслеживать прогресс и оставлять отзывы. Преподаватели получают доступ к созданию, редактированию и удалению курсов, а также просмотру статистики. Реализована система отзывов и рейтингов, позволяющая оценивать качество курсов на основе обратной связи.

База данных проекта спроектирована таким образом, чтобы эффективно хранить информацию о курсах, частях курсов, прогрессе пользователей и отзывах. В базе данных разработано 7 таблиц. Также реализована автоматическая переработка средней оценки курса после добавления или удаления отзыва, что обеспечивает актуальность отображаемых данных.

В ходе ручного тестирования было проверено корректное функционирование всех ключевых функций, включая регистрацию и авторизацию, создание и редактирование курсов, управление частями курсов, подписку на курс, отслеживание прогресса, систему отзывов и администрирование.

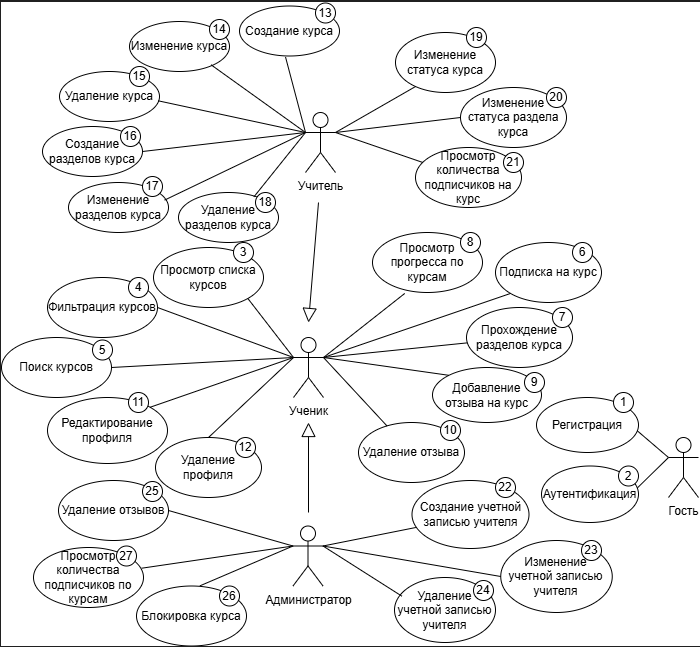
Разработанное web-приложение представляет собой готовую платформу для организации и прохождения онлайн-обучения, обладающую гибкой архитектурой, масштабируемостью и удобным пользовательским интерфейсом.

**Список используемых источников**

1. HTTP Documentation [Электронный ресурс] / Режим доступа: <https://httpwg.org/specs/> – Дата доступа: 20.02.2025.
2. Node JS Documentation [Электронный ресурс] / Режим доступа: <https://nodejs.org/docs/latest/api/> – Дата доступа: 20.02.2025.
3. Руководство по PostgreSQL [Электронный ресурс] / Режим доступа: <https://metanit.com/sql/postgresql/> – Дата доступа: 25.02.2025.
4. Udemy [Электронный ресурс] / Режим доступа: <https://www.udemy.com/> – Дата доступа: 25.02.2025.
5. Stepik [Электронный ресурс] / Режим доступа: <https://stepik.org/> – Дата доступа: 25.02.2025.
6. GeekBrains [Электронный ресурс] / Режим доступа: <https://gb.ru/> – Дата доступа: 26.02.2025.
7. React Documentation [Электронный ресурс] / Режим доступа: https://ru.legacy.reactjs.org/– Дата доступа: 01.03.2025.
8. Axios Documentation [Электронный ресурс] / Режим доступа: https://axios-http.com/docs/intro – Дата доступа: 01.03.2025
9. Руководство по Express [Электронный ресурс] / Режим доступа: <https://www.geeksforgeeks.org/express-js/> – Дата доступа: 10.03.2025.
10. Руководство по Sequelize [Электронный ресурс] / Режим доступа: <https://my-js.org/docs/guide/sequelize/> – Дата доступа: 10.03.2025.
11. Cloudinary [Электронный ресурс] / Режим доступа: https://cloudinary.com/ – Дата доступа: 20.03.2025.
12. Multer [Электронный ресурс] / Режим доступа: <https://webdiz.com.ua/nastroyka-multer/> – Дата доступа: 25.03.2025.
13. JWT [Электронный ресурс]. – Режим доступа: https://jwt.io – Дата доступа: 25.03.2025.
14. bcrypt [Электронный ресурс]. – Режим доступа: https://github.com/kelektiv/node.bcrypt.js – Дата доступа: 25.03.2025.
15. Dotenv [Электронный ресурс]. – Режим доступа: https://www.npmjs.com/package/dotenv/– Дата доступа: 25.03.2025.
16. Cors [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/ru/docs/Glossary/CORS – Дата доступа: 03.04.2025.
17. React-router documentation [Электронный ресурс]. – Режим доступа: https://reactrouter.com/ – Дата доступа: 03.04.2025
18. Reduxjs toolkit documentation [Электронный ресурс]. – Режим доступа: <https://redux-toolkit.js.org/> – Дата доступа: 03.04.2025
19. React-redux [Электронный ресурс]. – Режим доступа: https://redux.js.org – Дата доступа: 04.04.2025.
20. React-markdown [Электронный ресурс] / Режим доступа: https://www.npmjs.com/package/react-markdown – Дата доступа: 15.04.2025.

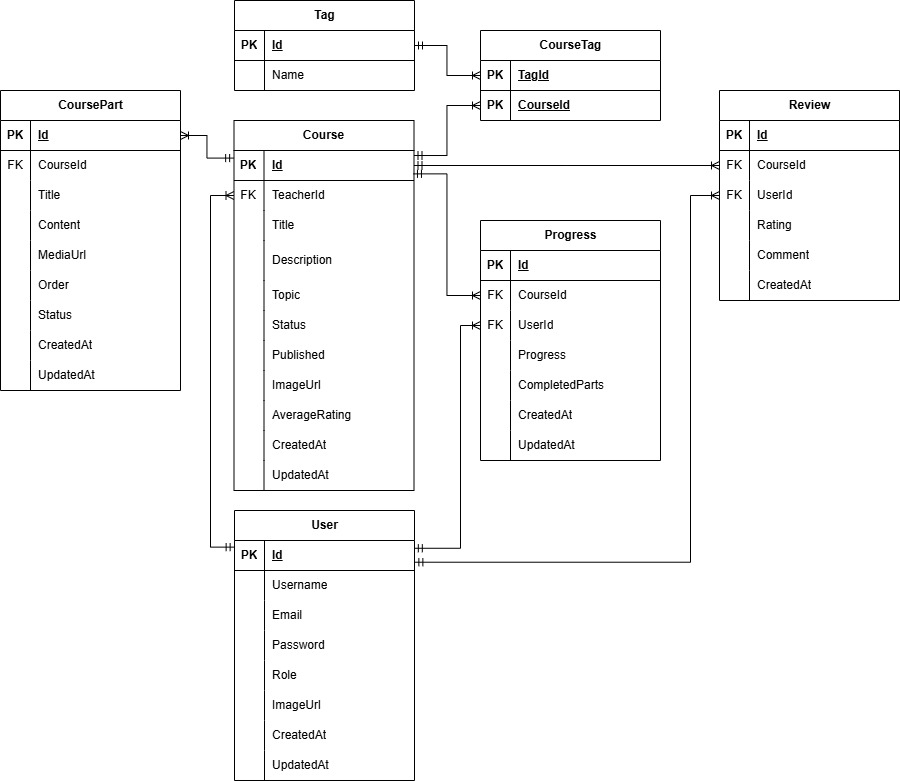
Приложение А

**Диаграмма вариантов использования**



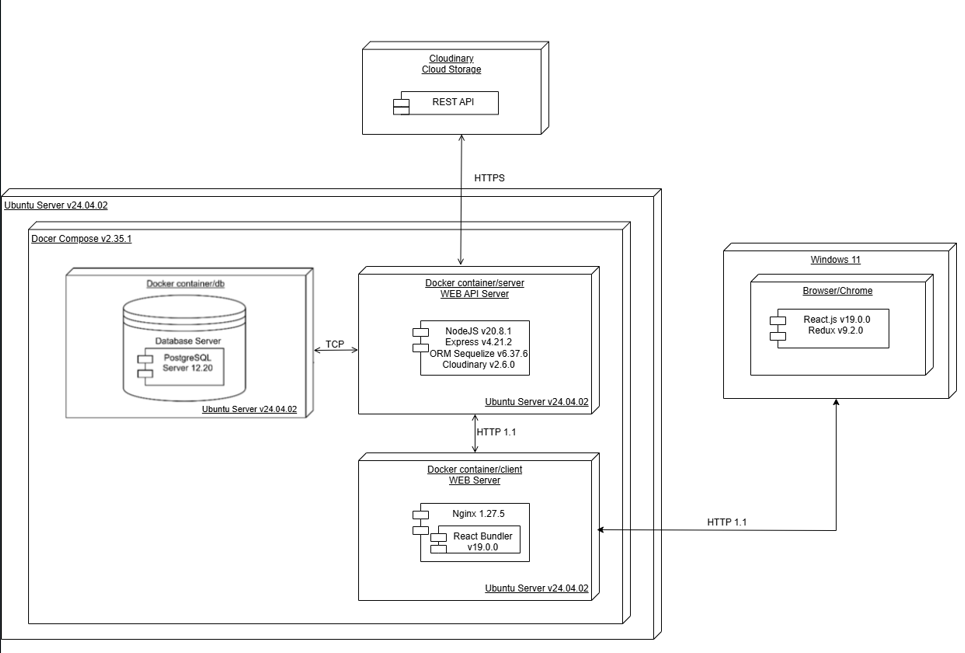
Приложение Б

**Логическая схема базы данных**



Приложение В

**Схема архитектуры приложения**

****

Приложение Г

|  |
| --- |
| export const getCoursesWithSearchFilter = async (req, res) => {  const userId = req.userId;  const { query, topic } = req.query;  const page = parseInt(req.query.page) || 1;  const limit = 8;  const offset = (page - 1) \* limit;  try {  const whereConditions = {  ...(topic && topic !== 'Все' && { topic }),  ...(query &&  query.trim() !== '' && {  [Op.or]: [  {  [Op.and]: [  Sequelize.where(Sequelize.fn('LOWER', Sequelize.col('Course.title')), {  [Op.like]: `%${query.toLowerCase()}%`,  }),  ],  },  {  id: {  [Op.in]: Sequelize.literal(`(  SELECT "courseId" FROM "CourseTags" WHERE "tagId" IN (  SELECT "id" FROM "Tags" WHERE LOWER(name) LIKE '%${query.toLowerCase()}%'  )  )`),  },  },  ],  }),  status: 'public', };  const { count, rows } = await Course.findAndCountAll({  where: whereConditions,  include: [  {  model: Tag,  through: { attributes: [] },  },  {  model: User,  attributes: ['username'],  },  {  model: Progress,  where: { userId: userId },  required: false,  attributes: ['progress'],  },  { model: CoursePart, where: { status: 'active' }, order: [['order', 'ASC']] },  ],  limit,  offset,  distinct: true,  order: [  ['averageRating', 'DESC'],  ['createdAt', 'DESC'],  ],  });  const totalPages = Math.ceil(count / limit);  res.status(200).json({  courses: rows,  currentPage: page,  totalPages,  });  } catch (err) {  console.error('Ошибка при получении данных о курсах: ', err);  res.status(500).json({ message: 'Внутренняя ошибка сервера' });  }  }; |

Листинг – Реализации метода getCoursesWithSearchFilter

Приложение Д

|  |
| --- |
| export const getInProgressCourses = async (req, res) => {  const userId = req.userId;  const limit = 4;  const { page = 1 } = req.query;  try {  const offset = (page - 1) \* limit;  const progresses = await Progress.findAll({  where: {  userId,  progress: {  [Op.lt]: 100,  },},  attributes: ['courseId'],  });  if (progresses.length === 0) {  return res.status(200).json({  courses: [],  totalPages: 1,  });}  const courseIds = progresses.map((progress) => progress.courseId);  const { count, rows: courses } = await Course.findAndCountAll({  where: {  status: 'public',  id: courseIds,  },  include: [  { model: Tag, through: { attributes: [] } },  { model: User, attributes: ['username'] },  { model: Progress,  where: { userId, progress: {[Op.lt]: 100,},},  required: false,  attributes: ['progress'],},  { model: CoursePart, where: { status: 'active' }, order: [['order', 'ASC']] },  ],  offset,  limit: parseInt(limit),  distinct: true,  });  return res.status(200).json({  courses,  totalPages: Math.ceil(count / limit),  totalCount: count,});  } catch (error) {  return res.status(500).json({ message: 'Ошибка при получении незавершённых курсов' });}}; |

Листинг – Реализации метода getInProgressCourses