МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационных систем и технологий

Специальность 1-40 01 01 «Программное обеспечение информационных технологий»

Специализация 1-40 01 01 10 «Программное обеспечение информационных технологий (программирование интернет-приложений)»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

Web-приложение «Салон красоты»

Выполнил студент Голодок Анастасия Юрьевна

(Ф.И.О.)

Руководитель проекта ст. преп. Дубовик М. В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой и. о. Блинова Е.А.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты ст. преп. Дубовик М. В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролёр ст. преп. Дубовик М. В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2024

**Содержание**

[Введение 3](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266197)

[1 Аналитический обзор литературы и постановка задачи 4](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266198)

[1.1 Обзор прототипов 4](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266199)

[1.1.1 SkillBox 4](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266200)

[1.1.2 TeachMeSkills 5](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266201)

[1.1.3 GeekBrains 5](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266202)

[1.1.4 Udemy 6](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266203)

[1.2 Постановка задачи 7](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266204)

[1.3 Описание используемых технологий 8](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266205)

[2 Разработка архитектуры проекта 10](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266206)

[2.1 Обобщенная структура проекта 10](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266207)

[2.2 Диаграмма UML 11](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266208)

[2.3 Модель базы данных 12](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266209)

[2.4 Идентификация и авторизация 16](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266210)

[3 Создание (реализация) программного средства 17](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266211)

[3.1 Реализация серверной части 17](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266212)

[3.2 Обмен сообщениями с клиентом в реальном времени 19](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266213)

[3.3 Реализация клиентской части 20](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266214)

[3.4 Конечные точки 22](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266215)

[4 Тестирование 23](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266216)

[4.1 Форма входа 23](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266217)

[4.2 Форма регистрации 24](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266218)

[4.3 Форма создания профиля пользователя 25](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266219)

[5 Руководство пользователя 27](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266220)

[5.1 Пользователь 27](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266221)

[5.2 Администратор 31](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266222)

[Заключение 36](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266223)

[Список используемых источников 37](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266224)

[Приложение A 38](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266225)

[Приложение Б 40](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266226)

[Приложение В 42](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266227)

[Приложение Г 43](file:///C:\Users\nasty\Downloads\Пояснительная%20записка.docx#_Toc135266228)

# **Введение**

В современном мире технологий и удобства онлайн-сервисов, сфера красоты и здоровья также начинает активно использовать цифровые инструменты для улучшения своих услуг и взаимодействия с клиентами. Разработка веб-приложений для салонов красоты становится важным направлением в индустрии, предлагая инновационные решения для управления бизнесом и обеспечения удобства клиентам.

Подобные приложения предлагают широкий спектр функций, начиная от онлайн-записи на услуги и управления расписанием мастеров до предоставления персонализированных рекомендаций. Вместе с тем, такие платформы обеспечивают салоны красоты необходимыми инструментами для эффективного управления своими ресурсами, а также повышения уровня сервиса.

Именно в этой связи разработка веб-приложения для салона красоты представляет собой интересный и перспективный проект, который объединяет в себе элементы технической инновации, дизайна и понимания потребностей индустрии красоты. Важно учитывать, как технические аспекты разработки, так и аспекты пользовательского опыта, чтобы создать продукт, который будет успешно интегрирован в повседневную деятельность салонов красоты и оценен клиентами.

В результате можно получить продукт, который может значительно улучшить взаимодействие между салонами и их клиентами, повысить эффективность бизнеса и улучшить качество предоставляемых услуг.

Целью данного курсового проектирования является разработка программного средства, предназначенного для обеспечения функциональности салона красоты. Для достижения поставленной цели необходимо выполнить следующие задачи:

* рассмотреть и проанализировать существующие аналоги;
* спроектировать web-приложение;
* разработать web-приложение;
* протестировать web-приложение;
* описать руководство пользователя.

# **1 Постановка задачи**

## **1.1 Обзор аналогов**

В современном мире набирают популярность сервисы, упрощающие процесс бронирования, записи на услуги и получения информации. Пользователи предпочитают делать все онлайн, вместо траты времени на звонки и поездки в нужное место. Также популярность таких сервисов обусловлена удобством использования, получения доступа в любое время и в любом месте без особых усилий.

## **1.1.1 Культура маникюра**

Рассмотрим в качестве примера сайт салона красоты «Культура маникюра» представленный на рисунке 1.1.



Рисунок 1.1 – Интерфейс главной страницы

Бронь услуги осуществляется в 4 этапа:

1. Выбор услуги;
2. Выбор доступного мастера и времени (рисунок 1.2);
3. Выбор конкретного вида услуги;
4. Заполнение личной информации.

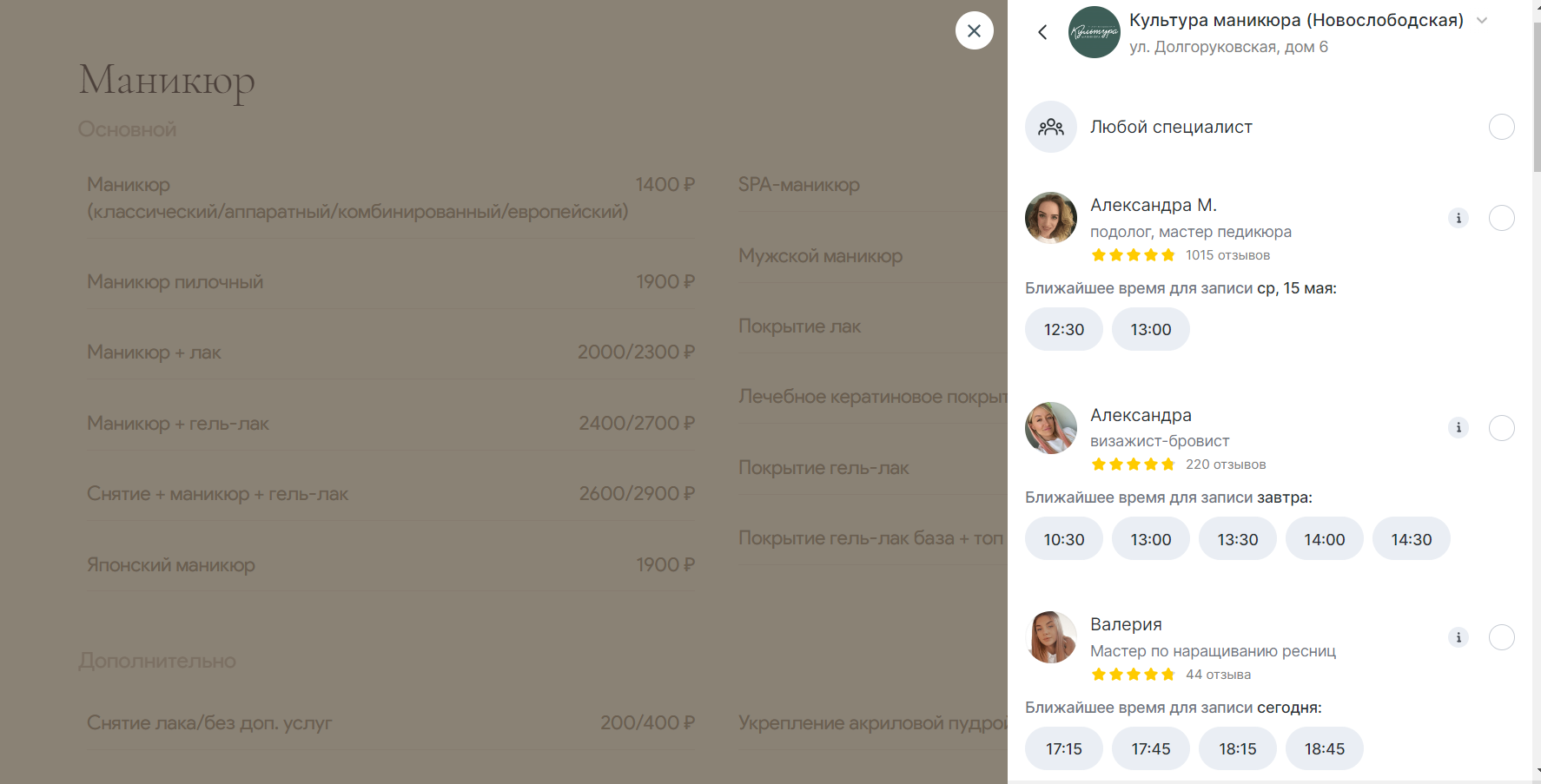


Рисунок 1.2 – Интерфейс выбора времени у мастера

Web-приложение выглядит достаточно стильно, демонстрирует высокий уровень салона, удобная последовательность и интерфейс при записи на услугу, осуществлена проверка на корректность введенных данных. Присутствует информация о каждой услуге, и цена. Из недостатков: на первый взгляд непонятно, как просмотреть или оставить отзыв. В остальном все сделано красиво и корректно.

## **1.1.2 Сафина**

Далее рассмотрим сайт салона красоты «Сафина». Сайт предлагает большое количество услуг. Запись на услугу также происходит в 4 этапа (рисунок 1.3).

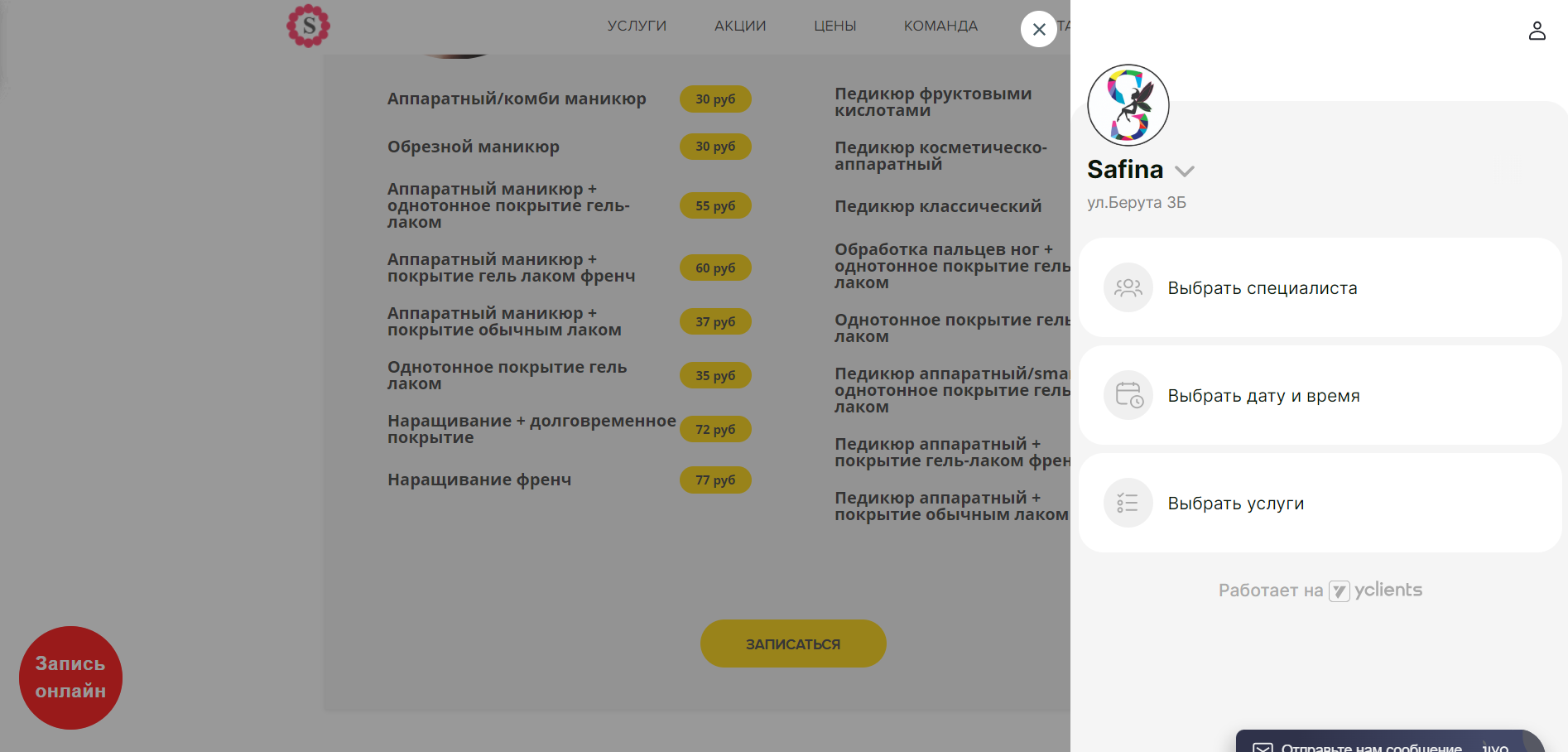


Рисунок 1.3 – Интерфейс страницы осуществления записи

Функциональность сайта не отличается от предыдущего аналога, удобная последовательность при записи на процедуру. Что касается дизайна, он более простой, выглядит не так стильно, как первый вариант, что ставит его на уровень ниже. Недостатков в работе с приложением нет.

## **1.2 Формирование требований**

Исходя из анализа web-приложений, были определены задачи будущего продукта.

Итогом разработки должно стать веб-приложение для просмотра информации об услугах, предлагаемых салоном красоты, а также осуществление записи на эти услуги.

Необходимо разработать несколько интерфейсов: для пользователя и для администратора.

Интерфейс пользователя должен давать возможность просмотра услуг, мастеров, осуществления записи и изменения личной информации о пользователе.

Интерфейс администратора должен позволять проводить операции с содержимым сайта, описаниями. Сайт должен быть выполнен в спокойных цветах и оттенках.

Таким образом, в данном курсовом проекте требуется реализовать следующие задачи:

– создать пользовательский интерфейс для взаимодействия с опциями приложения;

– регистрировать и авторизовать пользователей;

– изменять информацию о пользователях;

– просматривать услуги;

– осуществлять запись на услуги;

– добавлять, изменять и удалять услуги;

– добавлять, изменять и удалять мастеров;

– добавлять и просматривать отзывы.

# **2 Проектирование web-приложения**

Разработка архитектуры проекта – важная задача в процессе работы над приложением, потому что в зависимости от неё определяется уровень связности между компонентами приложения, и насколько легко можно будет это приложение расширить.

## **2.1 Архитектура приложения**

Приложение построено на основе клиент-серверной архитектуры.

Клиентская часть (frontend) использует React и MUI Components для реализации пользовательского интерфейса. React предоставляет эффективное модульное решение для создания интерфейса. MUI позволяет использовать готовые шаблоны и компоненты. Чтобы получить актуальные данные и обновить информацию, клиентская часть отправляет HTTPS-запросы на сервер.

Серверная часть (backend) реализована с использованием Node.js, который является средой выполнения JavaScript на сервере и обеспечивает высокую производительность и масштабируемость. Для создания веб-приложений на основе Node.js был выбран гибкий и минималистичный фреймворк Express. Сервер обрабатывает запросы от клиента, включая маршрутизацию запросов к соответствующим обработчикам. Данная часть будет обращаться к базе данных по протоколу TCP.

Для работы с базой данных была выбрана реляционная СУБД PostgreSQL. Node.js сервер использует ORM Prisma для CRUD операций.

Взаимодействие между клиентской и серверной частью приложения осуществляется следующим образом: клиент отправляет HTTPS-запросы на сервер с помощью axios. Сервер обрабатывает запросы, выполняет необходимые операции в базе данных и формирует HTTPS-ответы в формате JSON. Клиентская часть обрабатывает ответы от сервера и обновляет пользовательский интерфейс.

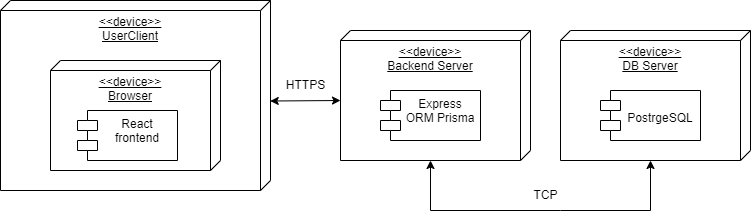


Рисунок 2.1 – Диаграмма развёртывания приложения

На диаграмме представлены все вышеописанные связи основных модулей приложения.

## **2.2 Диаграмма UML**

ДиаграммаUML–это графическое представление набора элементов,изображаемое в виде связанного графа с вершинами (сущностями) и ребрами (отношениями). Диаграмма использования представлена на рисунке 2.2.

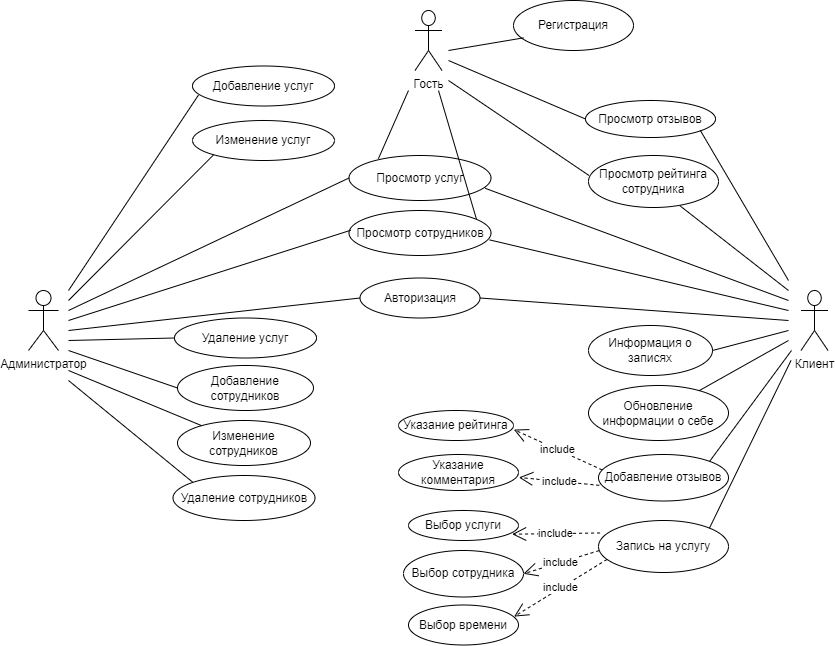


Рисунок 2.2 – Диаграмма вариантов использования

На данной диаграмме представлены 3 роли:

* Гость;
* Клиент;
* Администратор.

У гостя есть возможности регистрации, просмотра рейтинга и отзывов сотрудника, просмотра сотрудников и услуг, предоставляемых этими сотрудниками.

Роль клиента позволяет делать то же самое, что и гость, но еще можно авторизоваться, оставлять отзывы, записываться на услугу, а также просматривать свой профиль и редактировать информацию о себе.

Администратор может производить различные действия с услугами и сотрудниками, такие как добавление, изменение и удаление, а также просмотр информации.

## **2.3 Структура базы данных**

В разрабатываемом программном средстве существует возможность добавления, изменения данных в базе данных, взаимодействие происходит через запросы.

Для реализации поставленной в курсовом проектировании задачи была создана база данных «BeautySalon». Для её создания использовалась система управления реляционными базами данных PostgreSQL.

Выбор данного СУБД произошел по нескольким причинам: производительность и масштабируемость, расширяемость, безопасность, надёжность.

Все эти преимущества делают PostgreSQL привлекательным выбором для разработки и управления базами данных, включая курсовые проекты. Он предлагает гибкость, производительность и надежность, необходимые для успешной работы с данными и решения сложных задач.

Для базы данных «BeautySalon» было разработано 7 таблиц: Users, Employees, Registration, Services, Reviews, Schedule, EmployeesServices.

Таблица Users представляет собой данные о пользователе, состоит из столбцов:

* userID – идентификатор пользователя, тип number, первичный ключ;
* name – имя пользователя, тип nvarchar2(50);
* phone – телефон пользователя, nvarchar2(20);
* email – адрес электронной почты пользователя, тип nvarchar2(100);
* password – пароль пользователя, тип nvarchar2(50),
* role – роль (USER/ADMIN).

Таблица Employees представляет собой информацию о сотрудниках, состоит из столбцов:

* employeeID – идентификатор сотрудника, тип number, первичный ключ;
* serviceID – идентификатор услуги, тип number, внешний ключ;
* name – имя сотрудника, тип nvarchar2(50);
* positions – специализация сотрудника, тип nvarchar2(50);
* phone – телефон сотрудника, nvarchar2(20);
* email – адрес электронной почты сотрудника, тип nvarchar2(100).

Таблица Registration предназначена для хранения информации о записи на услугу, состоит из следующих столбцов:

* registrationID – идентификатор записи, тип number, первичный ключ;
* clientID – идентификатор клиента, тип number, внешний ключ;
* employeeID – идентификатор сотрудника, тип number, внешний ключ;
* dateTime – дата и время записи, тип TIMESTAMP;
* notes – комментарий, тип nvarchar2(255).

Таблица Servicesпредставляет собой информацию о сервисах, состоит из столбцов:

* serviceID – идентификатор услуги, тип number, первичный ключ;
* name – название услуги, тип nvarchar2(100), внешний ключ;
* description – описание услуги, тип nvarchar2(255);
* price – цена услуги, тип decimal(10,2).

Таблица Reviewsпредставляет собой информацию об отзывах, состоит из столбцов:

* reviewID – идентификатор отзыва, тип number, первичный ключ;
* userID – идентификатор пользователя, тип number, внешний ключ;
* employeeID – идентификатор сотрудника, тип number, внешний ключ;
* rating – рейтинг сотрудника, тип number;
* comm – отзыв, тип nvarchar2(255).

Таблица Scheduleпредставляет собой расписание сотрудников, состоит из столбцов:

* scheduleID – идентификатор расписания, тип number, первичный ключ;
* employeeID – идентификатор сотрудника, тип number, внешний ключ;
* date – день, тип DateTime;
* startTime – начальное время, тип DateTime;
* endTime – конечное время, тип DateTime.

Таблица EmployeesServices представляет собой связываение сотрудников и услуг для обеспечения связи многие ко многим.

Взаимосвязь всех таблиц проектируемой базы данных представлена на рисунке 2.3.

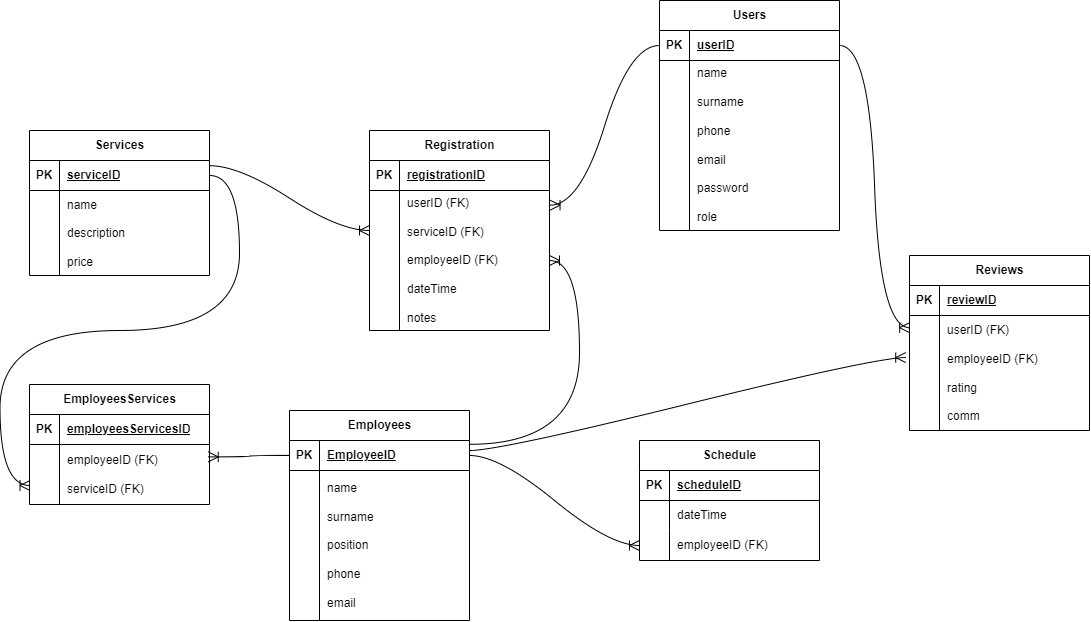


Рисунок 2.3 – Взаимосвязь таблиц базы данных

Листинг создания базы данных, используя ORM Prisma, представлен в приложении А.

Таким образом были созданы все необходимые для работы приложения таблицы в базе данных.

# **3 Разработка web-приложения**

## **3.1 Реализация серверной части**

При написании сервера на NodeJS используется фреймворк Express.

Express – это минималистичный и гибкий веб-фреймворк для приложений Node.js, предоставляющий обширный набор функций для мобильных и веб-приложений. Express не мешает общей производительности приложения, т.к. представляет собой тонкий слой основных функций веб-приложений. Также в нём удобно реализована маршрутизация. Имея в своем распоряжении множество служебных методов HTTP и промежуточных обработчиков, создать надежный API можно быстро и легко.

Проект разделен на несколько основных частей, которые расположены по разным директориям. На рисунке 3.1 представлена структура проекта.

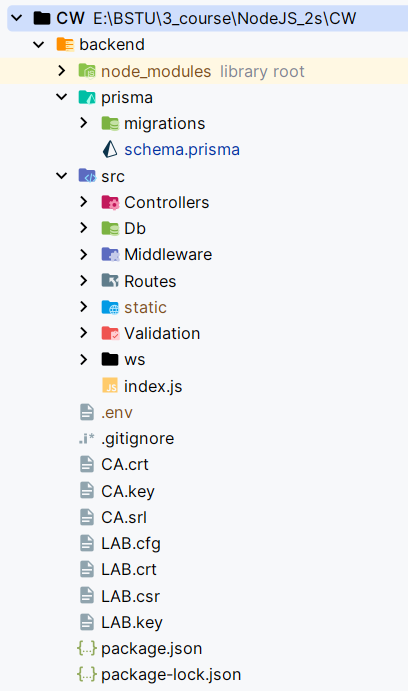


Рисунок 3.1 – Структура директорий проекта сервера

В директории «Сontrollers» расположены контроллеры. Контроллеры обеспечивают взаимодействие пользователя и системы, обрабатывая запросы. Они используют модель для реализации необходимого действия. Содержимое директории представлено на рисунке 3.2.

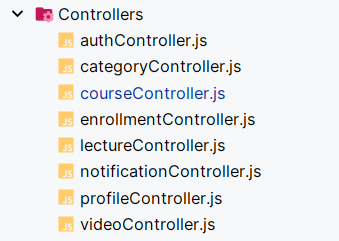


Рисунок 3.2 – Структура директории «Сontrollers»

Например, контроллер authController реализует вход, выход и регистрацию в приложении, проверяют данные для регистрации при вводе в форму, проверяется адрес электронной почты, является ли свободным, так как наличие двух пользователей с одинаковым email в приложении невозможно.

Пример одного из методов контроллера, а именно функция логина, приведена в Приложении В.

Директория «Middleware» содержит промежуточные обработчики которые нужны для проверки JWT токена и доступности действия исходя из роли запрашивающего.

Директория «prisma» в файле schema.prisma содержит модели, необходимые для генерации таблиц в базе данных. Конфигурация подключения импортируется из файла .env. Благодаря правильно прописанной модели, изменить базу данных не сложно, достаточно сделать миграцию.

Директория «Routes» содержит роутеры, которые реализуют маршрутизацию. Ниже в листинге 3.1 представлено добавление роутеров с учётом их расположения в директориях проекта. Также мы настраиваем приложение на использование роутеров по определённому пути.

const authRouter = require("./Routes/authRouter");  
const categoryRouter = require("./Routes/categoryRouter");  
const courseRouter = require("./Routes/courseRouter");  
const lectureRouter = require("./Routes/lectureRouter");  
const videoRouter = require("./Routes/videoRouter");  
const enrollmentRouter = require("./Routes/enrollmentRouter");  
const profileRouter = require("./Routes/profileRouter");  
const notificationRouter = require("./Routes/notificationRouter");

Листинг 3.1 – Регистрация роутеров

Директория «static» содержит статические файлы: видео, использующиеся на стороне клиента, которые необходимы для создания лекций по курсу.

В директории «Validation» разработаны различные промежуточные проверки на корректный ввод данных. Например, чтобы поля были непустыми, проверка на количество символов в пароле и т.д. Содержимое директории представлено на рисунке 3.3.



Рисунок 3.3 – Структура директории «Validations»

Секретные данные для работы приложения, такие как порт, на котором будет запущен сервер, строка подключения к базе данных, секрет для JWT-токена, а также названия сертификатов и ключей для HTTPS протокола содержатся в .env файле.

В файле index.js настраивается сервер. В нем создаётся объект сервера, регистрируются плагины и запускается сервер на порту, хранящимся в env-файле. Помимо всего прочего, там настраивается CORS для корректного доступа с веб-клиентом.

## **3.2** **Обмен сообщениями с клиентом в реальном времени**

При разработке приложения использованы веб-сокеты для поддержки двусторонней связи.

WebSocket — протокол связи поверх TCP-соединения, предназначенный для обмена сообщениями между браузером и веб-сервером в режиме реального времени.

Для установления соединения WebSocket клиент и сервер используют протокол, похожий на HTTP. Клиент формирует особый HTTP-запрос, на который сервер отвечает определенным образом.

Запрос от клиента на сервер представлен на рисунке 3.4

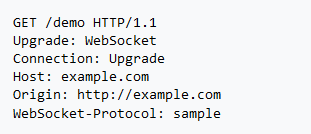


Рисунок 3.4 – Запрос клиента для установки соединения WebSocket

Если сервер поддерживает такой протокол, то в ответ клиенту об этом, что протокол WebSocket будет использоваться для текущего соединения. Ответ можно увидеть на рисунке 3.5

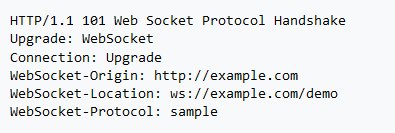


Рисунок 3.5 – Ответ от сервера, в случае успешной смены протокола

Веб-сокеты в данном приложении реализованы с помощью библиотеки Socket.io и используются для реализации реакции других клиентов, подключенных к серверу, на какие-либо действия, таких как отправка уведомление о создании нового курса или обновление курса администратором.

После установки соединения по данному протоколу начинается обмен между сервером и пользователем. Сервер пересылает сообщения и другие данные о коммуникации клиентского приложения и сервера.

## **3.3 Реализация клиентской части**

Для клиентской части приложения был использован фреймворк React JS. С помощью React JS создавать интерактивные пользовательские интерфейсы приятно и просто. Достаточно описать, как части интерфейса приложения выглядят в разных состояниях. React будет своевременно их обновлять, когда данные изменяются.

Декларативные представления сделают код более предсказуемым и упростят отладку.

Поскольку логика компонента написана на JavaScript, а не содержится в шаблонах, можно с лёгкостью передавать самые разные данные по всему приложению и держать состояние вне DOM.

В дополнение используются библиотеки react-bootstrap, socket.io-client, redux, axios, fontawesome. Структура директорий проекта приведена на рисунке 3.6.

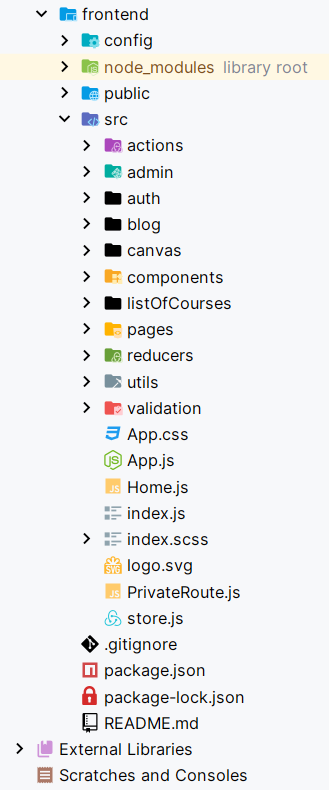


Рисунок 3.6 – Структура директорий проекта клиента

Подробно опишем некоторые файлы и папки в корне проекта frontend.

В директории «actions» хранятся запросы на сервер, связанные с авторизацией и профилем пользователей. «admin» - компоненты для прорисовки страниц, доступные только администратору. В папке «auth» находятся компоненты для логина и регистрации, «blog» содержит компоненты для управления курсами и лекциями. Так как в проекте используется библиотека для работы с диаграммами, отображающими статистику для администратора, её подключение находится в директории «canvas». Для хранения компонентов, позволяющих работать с профилем пользователей: заполнение, редактирование, отображение, просмотр всех профилей администратором, была создана отдельная папка «components». «listOfCourses» - директория, содержащая компоненты для просмотра курсов, на которые подписаны студенты, а также просмотра всех доступных курсов для всех пользователей. В папке «pages» находится страница для вывода ошибок, связанных с несуществующими маршрутами. «reducers» - редьюсеры, т.е. функции, которые принимают предыдущее состояние и экшен (state и action) и возвращают следующее состояние (новую версию предыдущего). В папке «utils» реализованы функции для добавления токена к заголовку, экспорт функции для взаимодействия с сервером, а также создание и экспорт сокета. Директория «validation» предназначена для проверки на непустые данные. Home.js – это компонент стартовой страницы, index.js – файл, предназначенный для сопоставления маршрутов и компонентов, являющийся также точкой входа в приложение, store.js – глобальное хранилище проекта. Точка входа в приложение – файл index.js.

Таким образом, созданные директории помогают понять общую структуру проектов проектируемого программного средства и легко ориентироваться в ней.

## **3.4 Конечные точки**

Описание доступных запросов взаимодействия приведено в Приложении Г. Для взаимодействия с частью запросов необходима авторизация. Токен должен содержаться в заголовках запросов.

Конечные точки (эндпоинты) приложения являются важным аспектом клиент-серверной архитектуры. Они представляют собой определенные URL-адреса на сервере, к которым клиенты могут обращаться для выполнения определенных операций или получения данных.

Показывая эндпоинты, мы видим описание доступных операций и ресурсов, которые клиенты могут использовать для взаимодействия с сервером. Каждый эндпоинт обычно связан с определенной функциональностью или сущностью в приложении.

Таким образом, предоставление эндпоинтов приложения позволяет ясно определить доступные операции и ресурсы, а также облегчает взаимодействие между клиентской и серверной частями приложения. Это важная составляющая разработки клиент-серверных приложений, которая помогает создать функциональное и эффективное приложение, а также позволяет более детально показать разделение эндпоинтов на сущности в веб-приложении.

# **4 Тестирование web-приложения**

В этой главе будут рассмотрены основные элементы интерфейса и протестирован интерфейс веб-приложения.

Приложение во многих местах позволяет пользователю вводить данные. Разработанное приложение устойчиво к вводу неверной информации, и сообщает пользователю об ошибках.

## **4.1 Тестирование web-приложения**

Для того, чтобы выполнять какие-либо действия, пользователю необходимо авторизоваться, либо же перейти по ссылке и зарегистрироваться. Форма регистрации с проверкой на корректные данные представлена на рисунке 4.1.

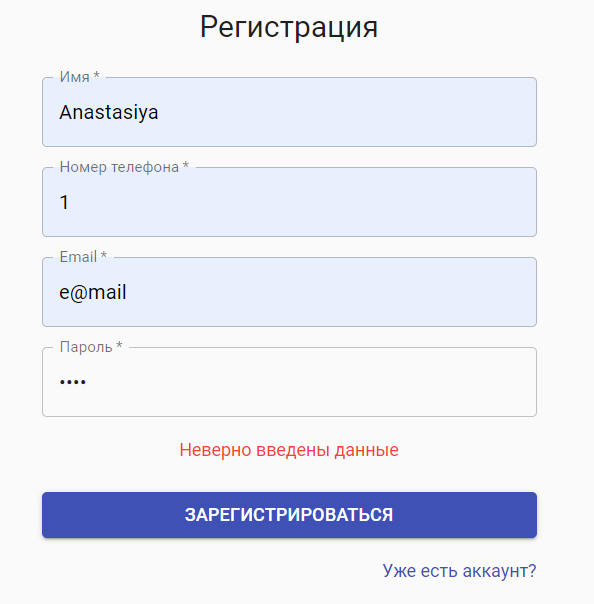


Рисунок 4.1 – Форма регистрации при неправильных данных

При вводе правильного логина и пароля пользователь перенаправляется на главную страницу приложения, в случае неверного ввода данных, появляются сообщения внизу поля, сигнализируя о неправильном вводе email или пароля. Интерфейс формы входа можно увидеть на рисунке 4.2.

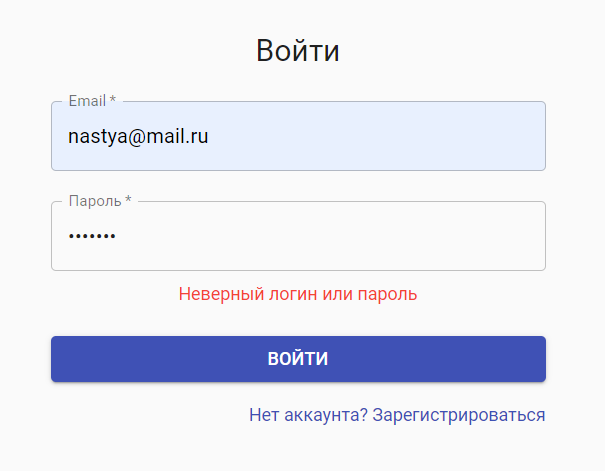


Рисунок 4.2 – Форма входа при неправильных данных

Существует также проверка на существование пользователя с введённым адресом электронной почты. При попытке войти в приложение через пользователя с уже зарегистрированным email, также появляется ошибка.

Если данные указаны неверно, то пользователь не сможет сделать запрос для регистрации. Только после успешной регистрации пользователь будет перенаправлен на страницу логина.

Далее будет протестирована запись на услугу, если пользователь не авторизован. В случае, когда пользователь вошел в свой аккаунт, нажимая кнопку «Записаться», он попадает на страницу записи на услугу, представленную на рисунке 4.3.

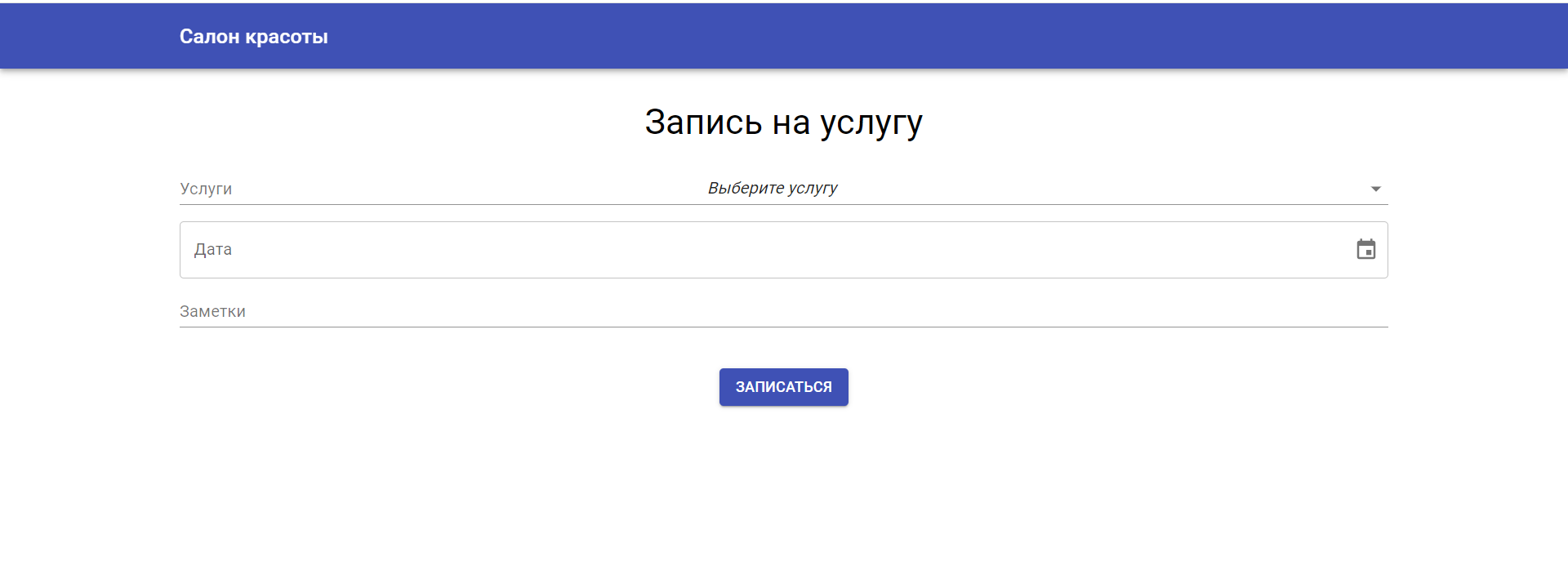


Рисунок 4.3 – Форма для записи на услугу

Но если пользователь не авторизован, то его перенаправляет на страницу входа.

Следующим этапом осуществлена проверка доступа

Еще одним этапом является проверка возможных ошибок со стороны администратора. На рисунке 4.5 представлен результат добавления услуги, которая уже существует.

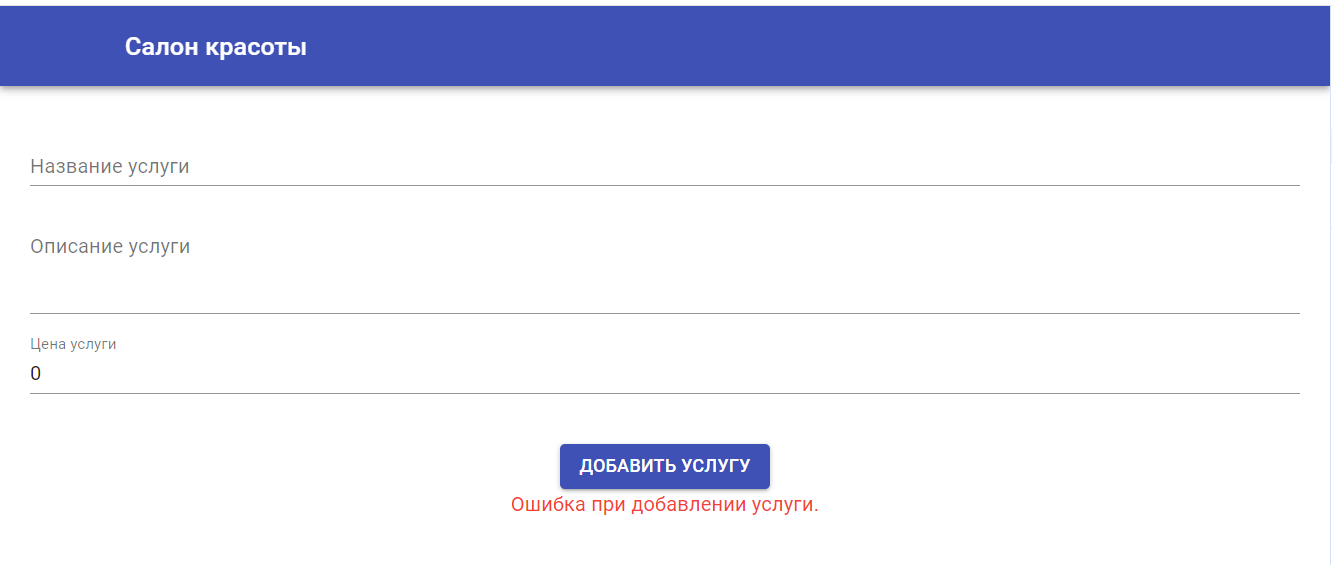


Рисунок 4.5 – Форма для записи на услугу

При добавлении сотрудника осуществляется проверка на корректность введенных данных, выбора услуг, а также правильность дат и времени. Eсли что-то неверно заполнено, выводится сообщение с ошибкой и форма сбрасывается.

# **5 Руководство пользователя**

В данном разделе находится ознакомление с остальным функционалом, доступным пользователям.

## **5.1 Гость**

При первом переходе к приложению пользователь попадает на главную страницу, вид которой приведен на рисунке 5.1. На ней можно посмотреть список предоставляемых услуг и цены на них, список сотрудников их рейтинг и отзывы, чтобы выбрать подходящего.

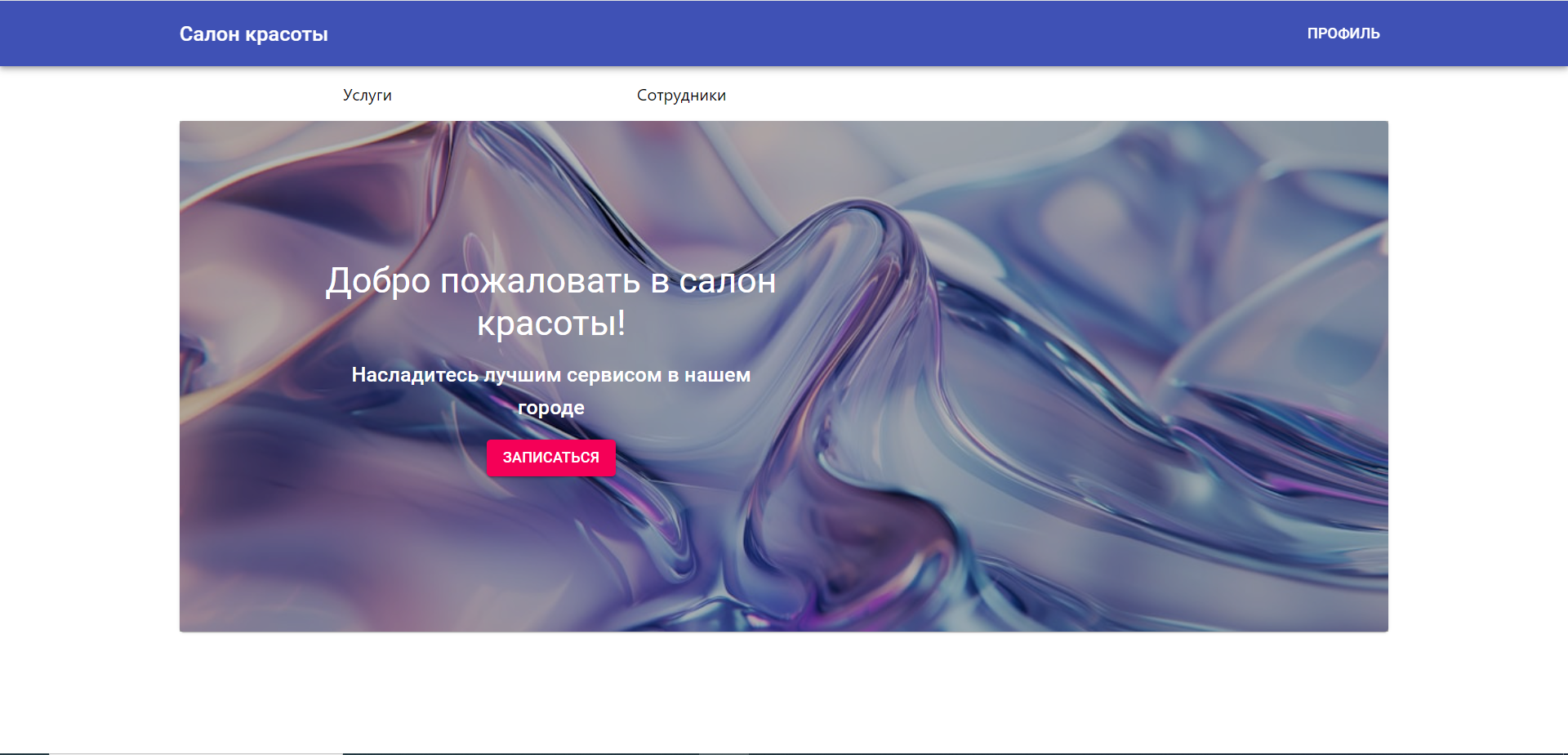


Рисунок 5.1 – Главная страница

Для просмотра рейтинга и отзывов сотрудников нужно перейти на страницу «Сотрудники» и кликнуть по интересующему, пример на рисунке 5.2

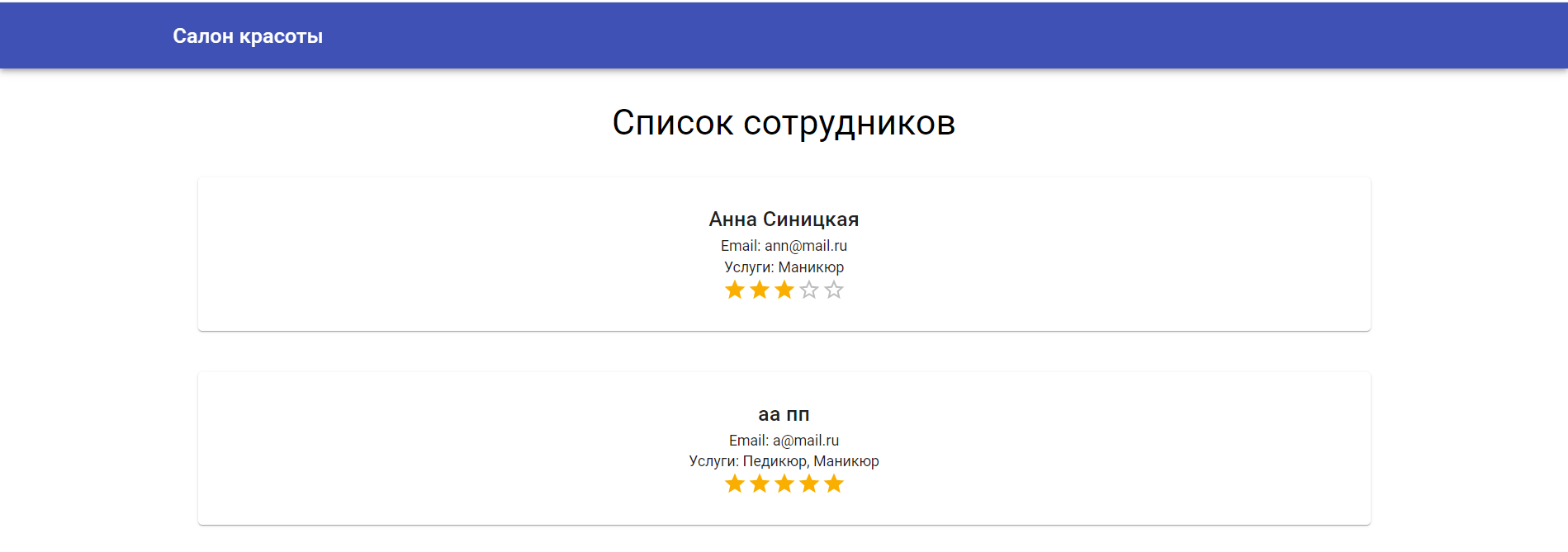


Рисунок 5.2 – Список сотрудников

Для доступа к записи на услугу, отправки отзыва и остальным функциям необходимо авторизоваться, для этого можно нажать кнопку «Войти». Также при нажатии кнопки «Записаться» неавторизованного пользователя перенаправляет на страницу входа. Страница авторизации приведена на рисунке 5.3.

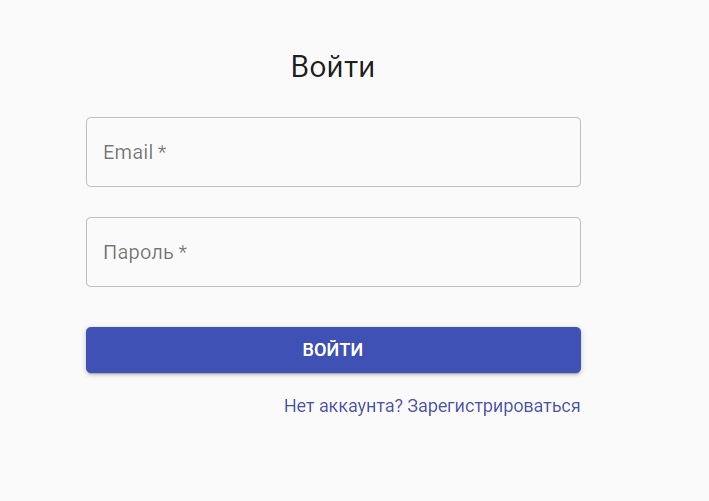


Рисунок 5.2 – Форма авторизации

В случае, если аккаунта ещё нет, нужно перейти по ссылке и заполнить форму. После регистрации пользователь снова попадает на страницу входа.

## **5.2 Пользователь**

Нажав на «Courses» и выбрав в выпадающем списке «My courses», открывается страница со всеми курсами, которая представлена на рисунке 5.3.

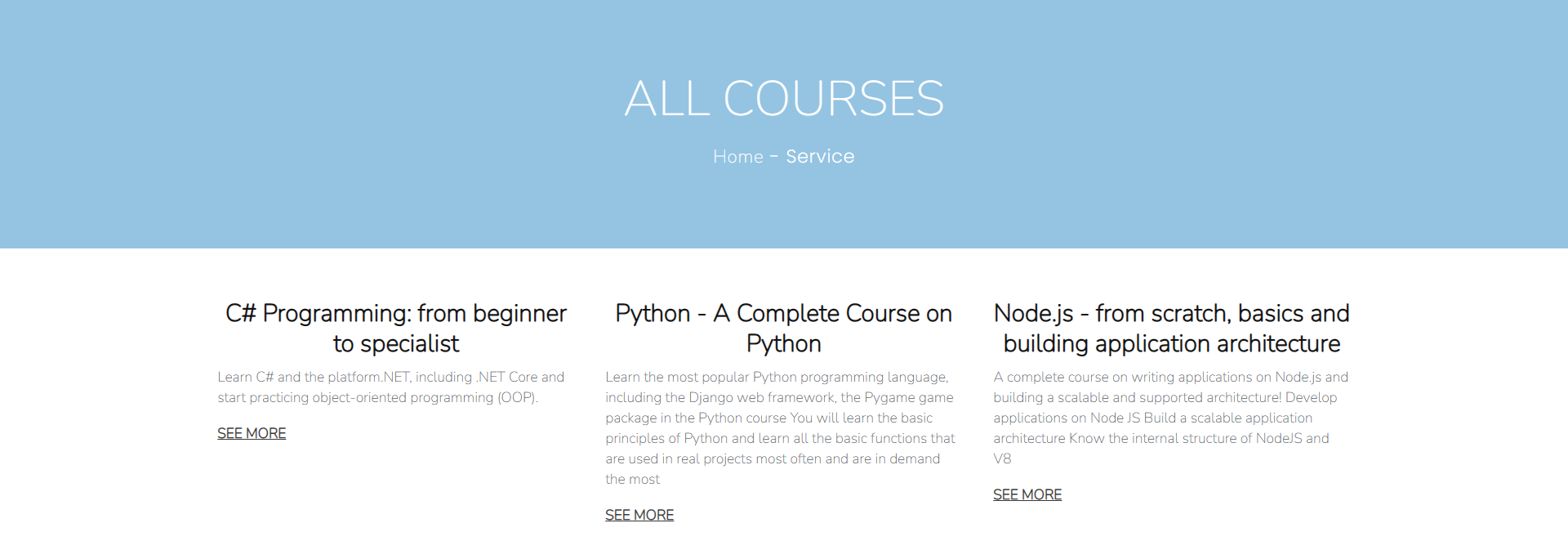


Рисунок 5.3 – Просмотр списка курсов

Выбрав в выпадающем списке «My courses», студенту будут отображаться лишь те курсы, на которые он записан, это позволяет легко и быстро находить интересующие курсы (рисунок 5.4).

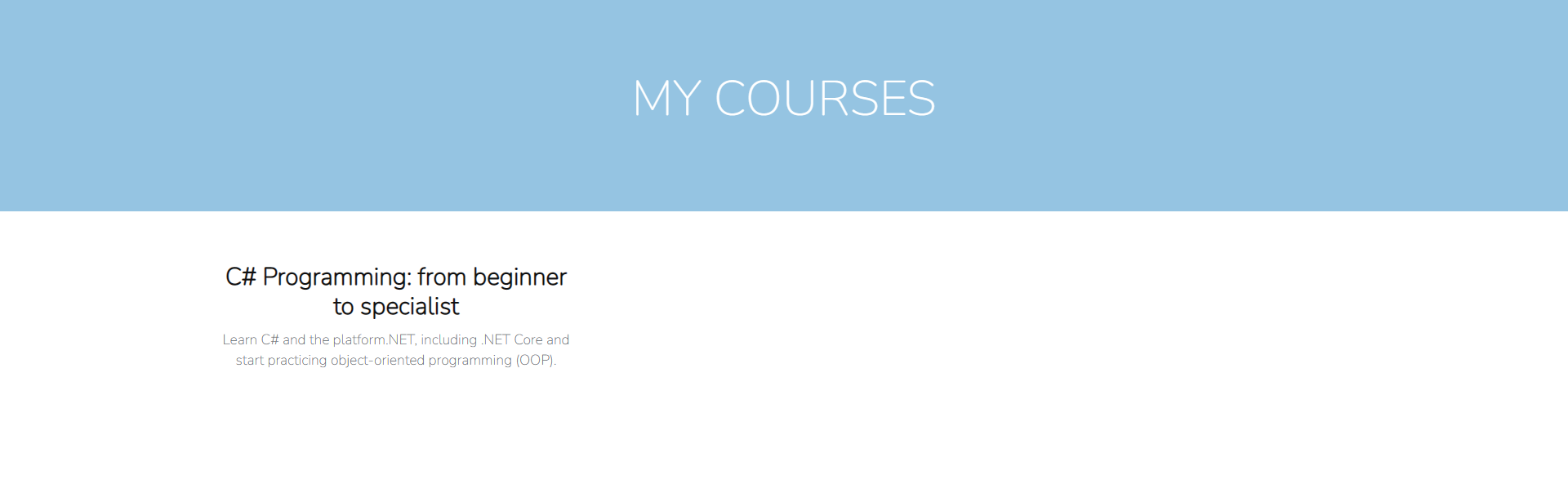


Рисунок 5.4 – Просмотр курсов, на которые записан студент

Для просмотра лекций нужно просто нажать на желаемый курс и загрузится список лекций. При переходе на различные лекции пользователь будет видеть материалы, прикрепленные к ним. Это отображено на рисунке 5.5.

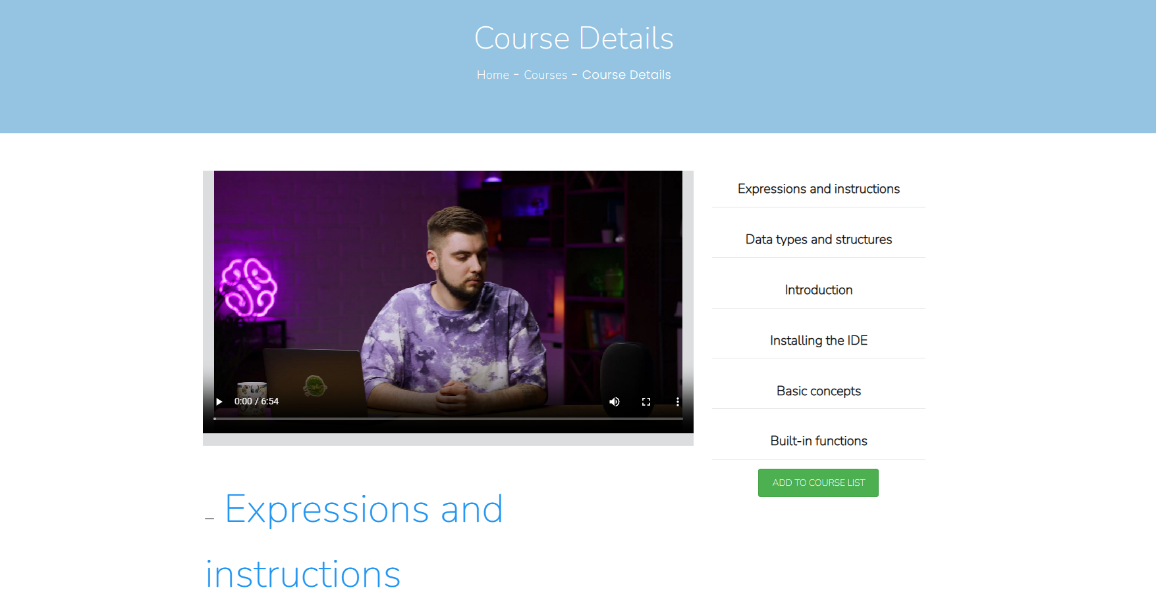


Рисунок 5.5 – Просмотр лекций

Также пользователь может заполнить свой профиль. Для этого выбираем вкладку «Profile» и нажимаем «Create Profile», как на рисунке 5.6.

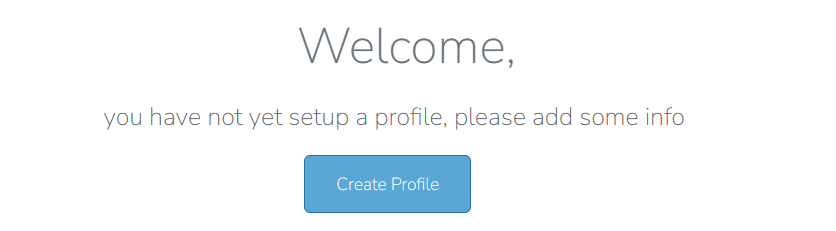


Рисунок 5.6 – Создание профиля

После нажатии кнопки «Create Profile», появится форма для создания профиля, представленная на рисунке 5.7. В форме есть поля, которые являются обязательными. Если хотя бы одно из них будет не заполнено, то пользователь не сможет создать профиль и будет предупрежден о том, что ему нужно добавить значения в обязательные поля.

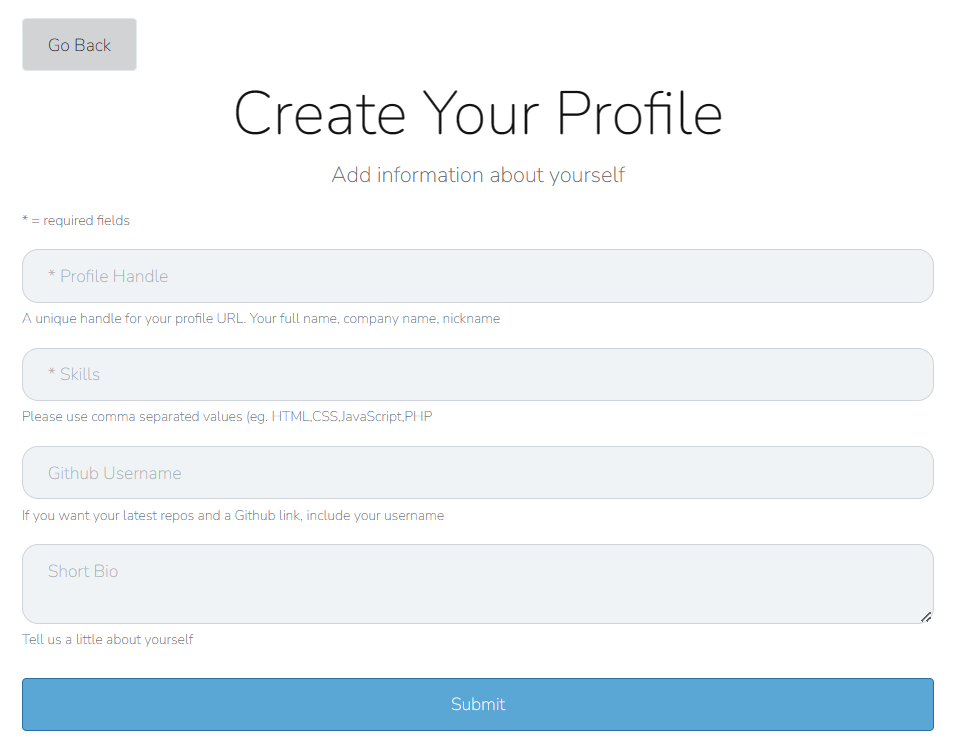


Рисунок 5.7 – Создание профиля

После заполнения всех полей необходимо нажать на «Submit» для сохранения профиля. И появится окно, представленное на рисунке 5.8.

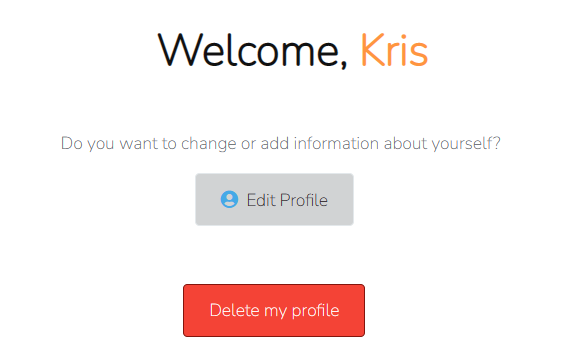


Рисунок 5.8 – Профиль пользователя

Для просмотра профиля необходимо нажать на свое имя пользователя. (рисунок 5.9).

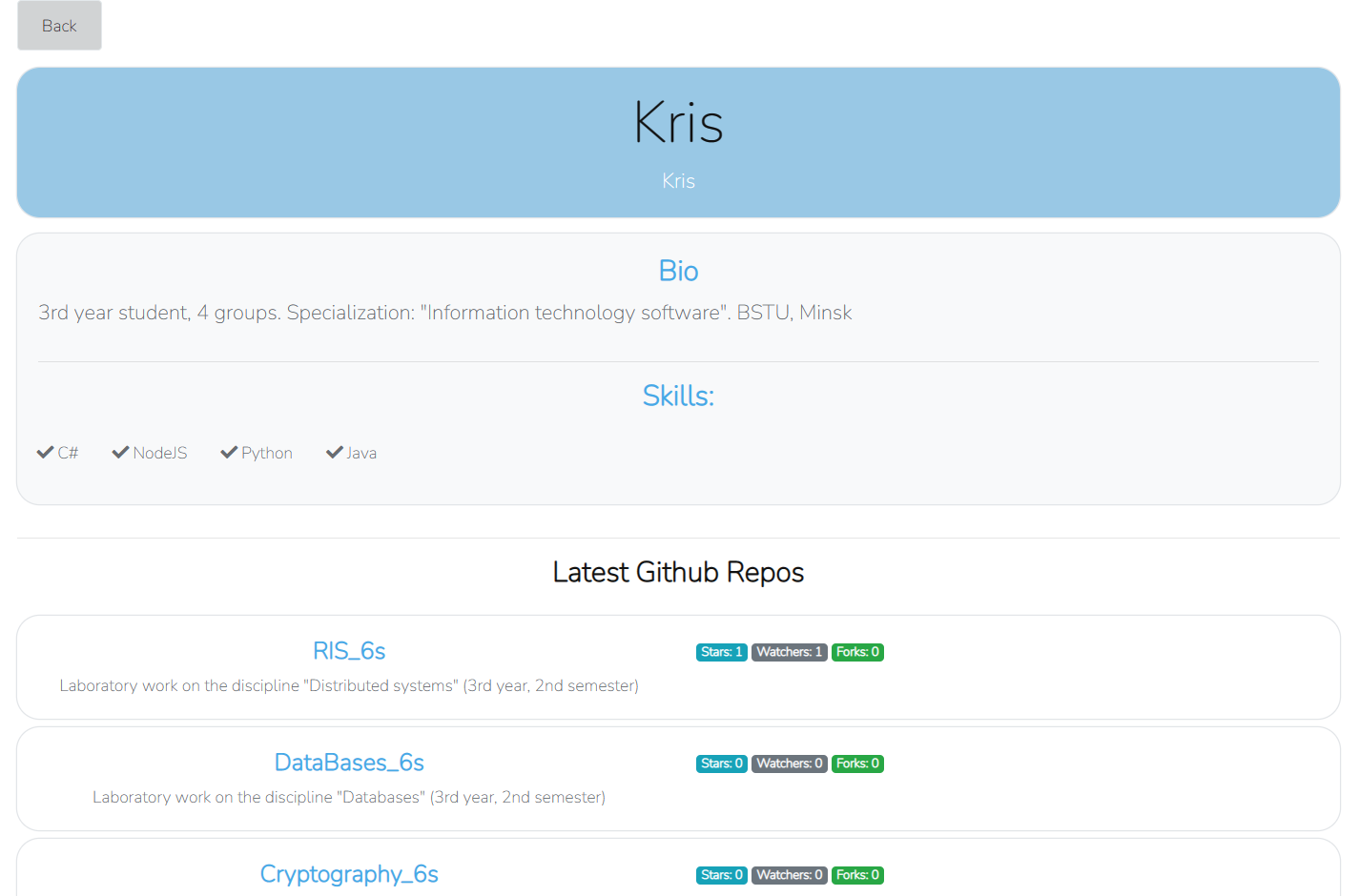


Рисунок 5.9 – Просмотр профиля

Здесь отображается информация, указанная при добавлении профиля, а также подгружаются репозитории из указанного GitHub аккаунта. Пользователь может удалить свой аккаунт, нажав «Delete my profile».

Также пользователю доступна страница «Notifications», представленная на рисунке 5.10, на которой отображаются уведомления, когда администратор создал новый курс или изменил существующий.

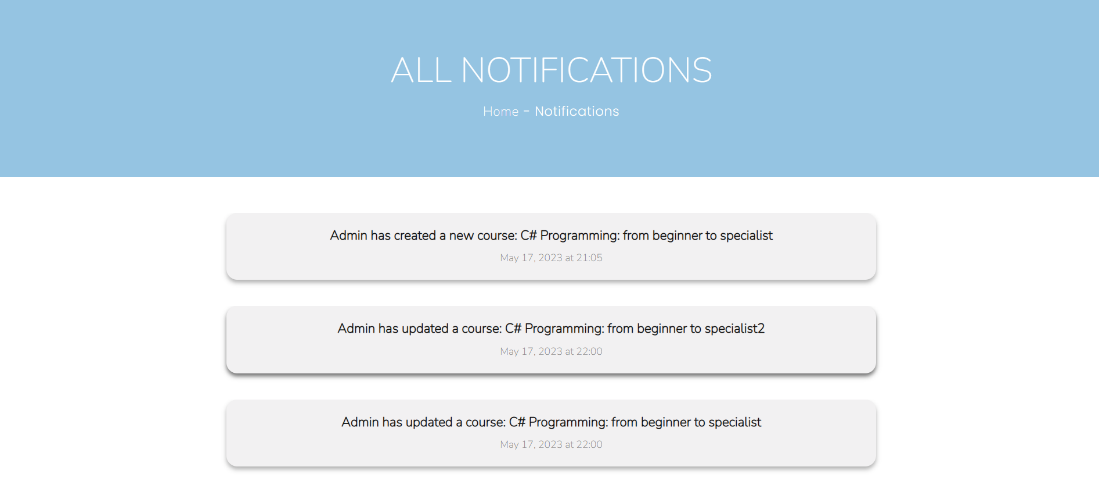


Рисунок 5.10 – Страница уведомлений пользователя

Для выхода из системы достаточно нажать на кнопку «Logout».

## **5.2 Администратор**

После входа в систему главная страница администратора точно такая же, как и у обычного пользователя, но с различными пунктами меню, которые представлены на рисунке 5.11.



Рисунок 5.11 – Возможности администратора

Администратор имеет возможность просматривать статистику в системе. Диаграммы можно посмотреть на вкладке «Statistic», страница которой представлена на рисунке 5.12.

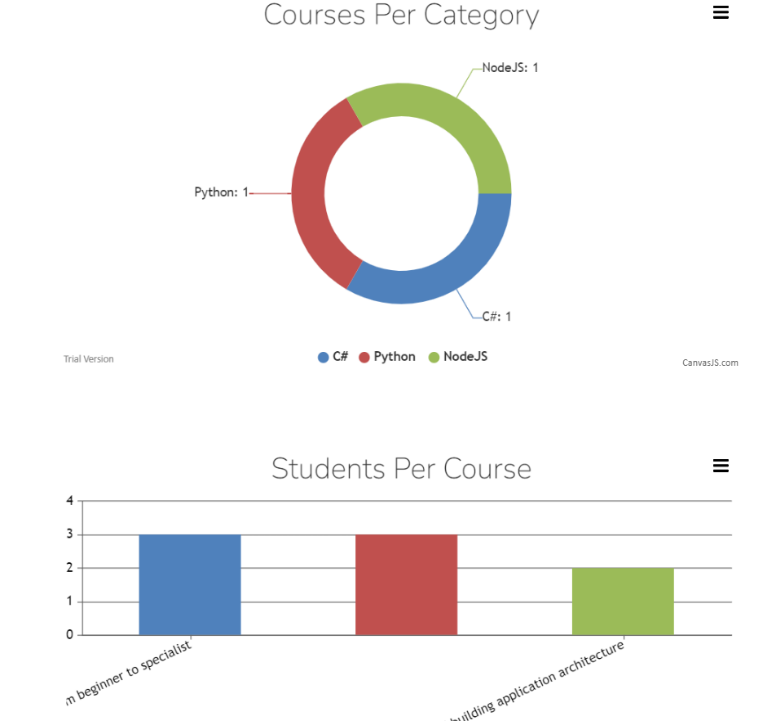


Рисунок 5.12 – Статистика в приложении

Диаграммы администратор может распечатать или сохранить себе в различных представленных форматах. Они отображают количество курсов по различным категориям и количество студентов, которые проходят какой-то определенный курс.

Также есть возможность управления пользователями – вкладка «Users», страница которой отображена на рисунке 5.13.

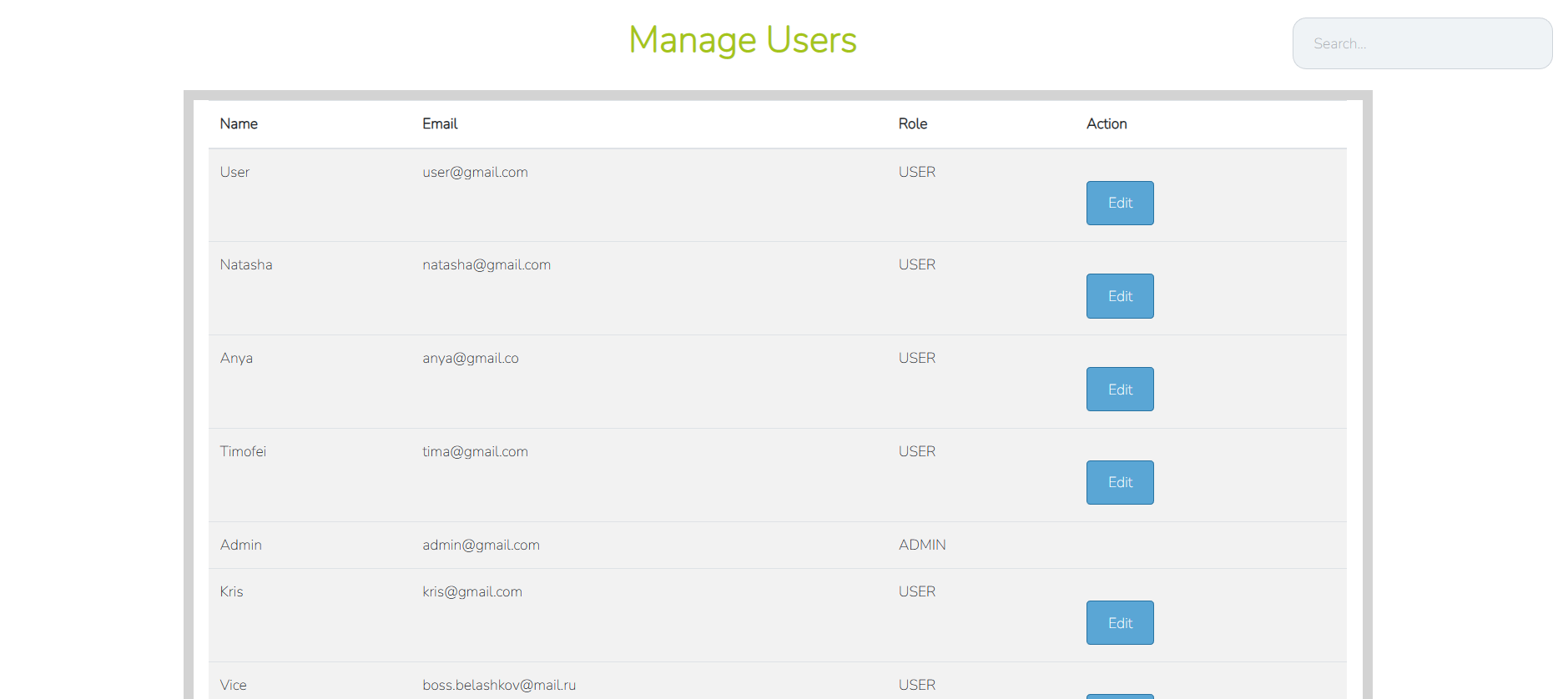


Рисунок 5.13 – Управление пользователями

Есть возможность поиска по таблице пользователей. Для того чтобы отредактировать информацию о каком-то пользователе, нужно в строке с ним нажать на «Edit», внести изменения в появившейся форме и сохранить их.

Также есть возможность управления курсами. Это осуществляется на вкладке «Courses», страница которой представлена на рисунке 5.14.

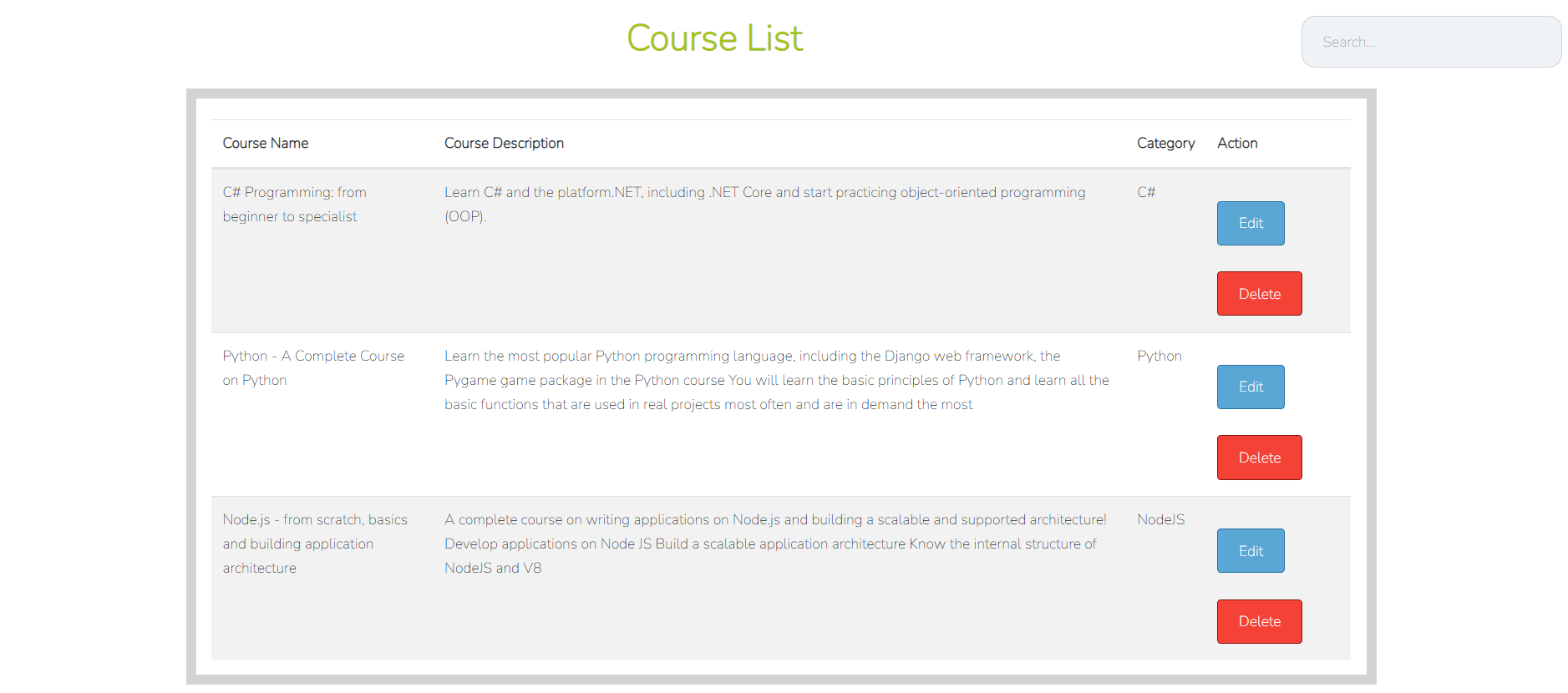


Рисунок 5.14 – Управление курсами

Точно так же как и с пользователями, администратор может редактировать информацию при нажатии на «Edit».

Для того, чтобы добавить категорию для курсов, нужно перейти на вкладку «Categories» и в появившейся форме (рисунок 5.15) нажать «Create Category».

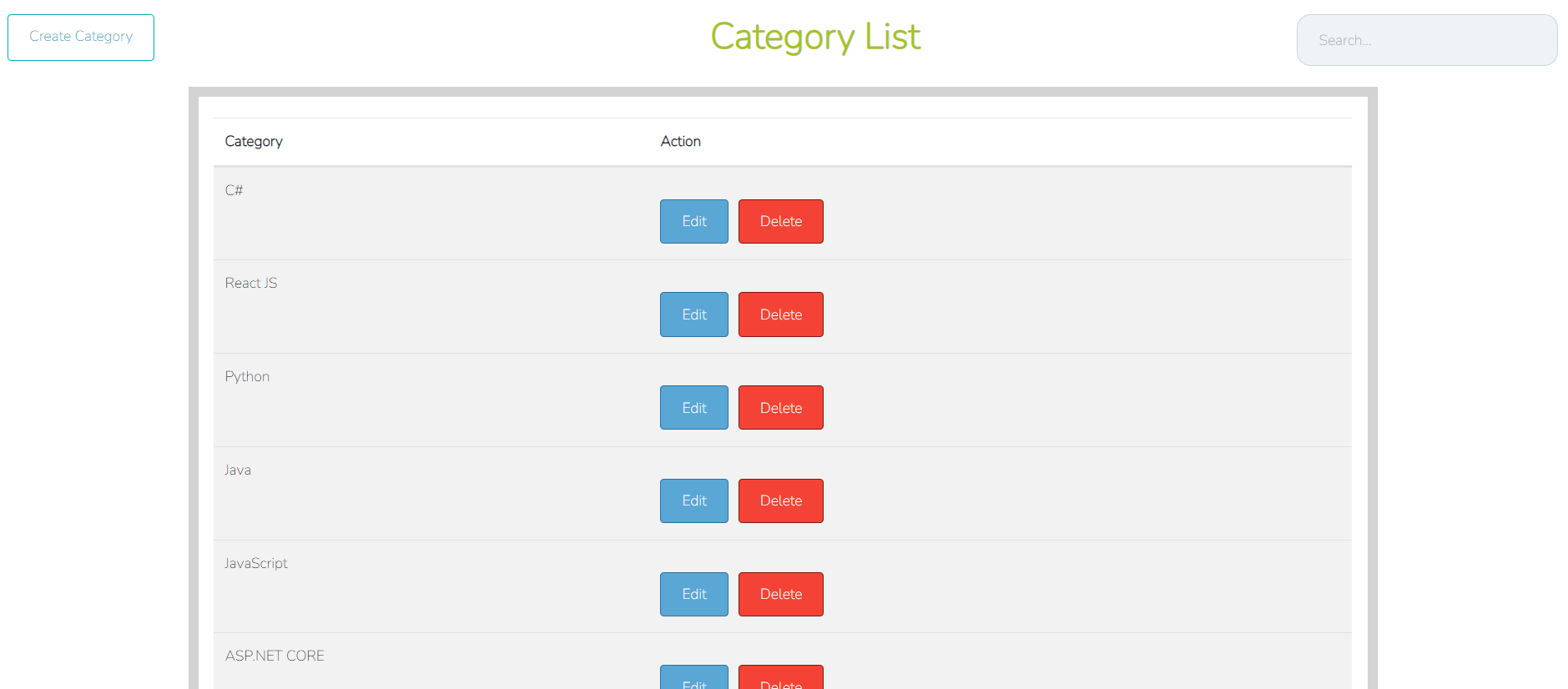


Рисунок 5.15 – Управление категориями

Как и в других вкладках, есть возможность редактирования и поиска.

Также можно просмотреть список студентов по курсам (рисунок 5.16).

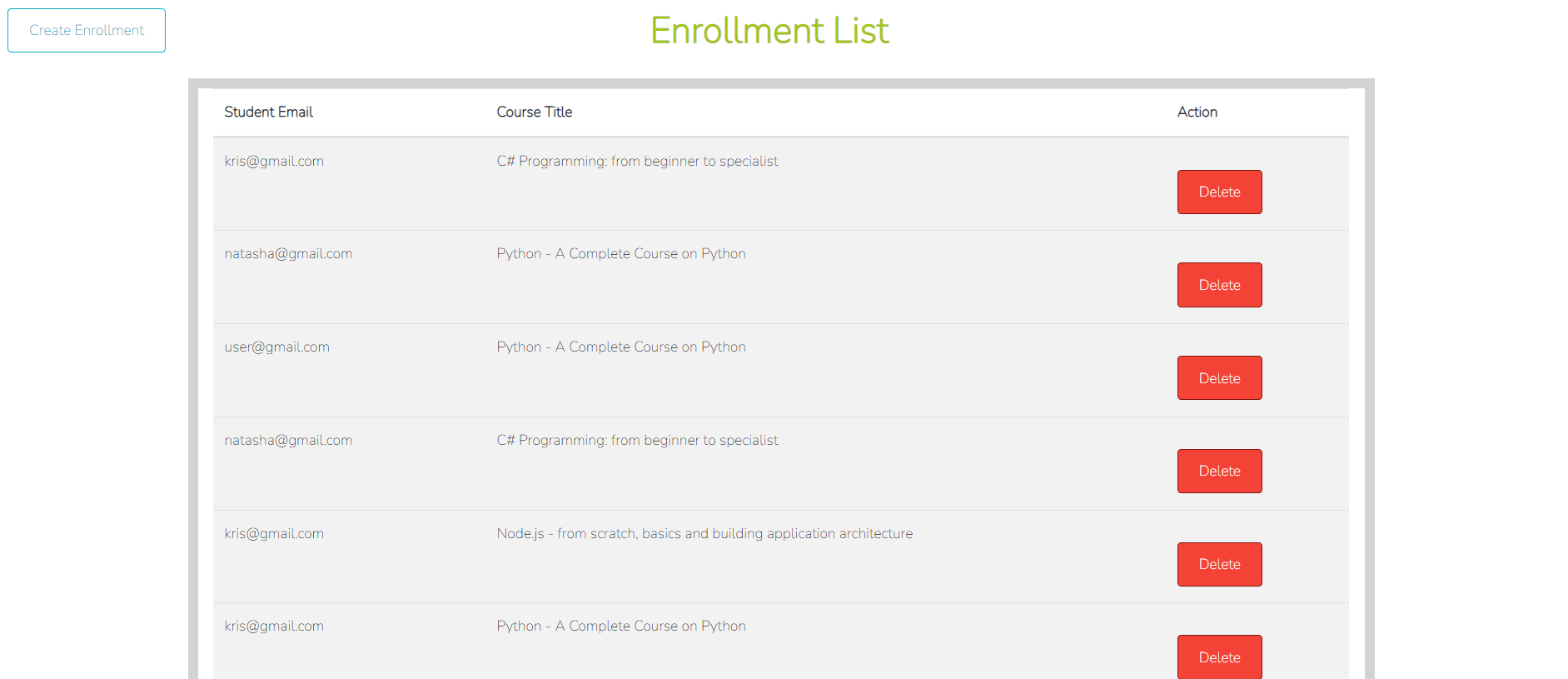


Рисунок 5.16 – Управление записями на курсы

Можно записать студента на определенный курс по кнопке «Create Enrollment», или удалить студента с курса по кнопке «Delete».

Для просмотра профилей всех пользователей, необходимо нажать на «All Profiles» в меню, после чего появится страница, представленная на рисунке 5.17.

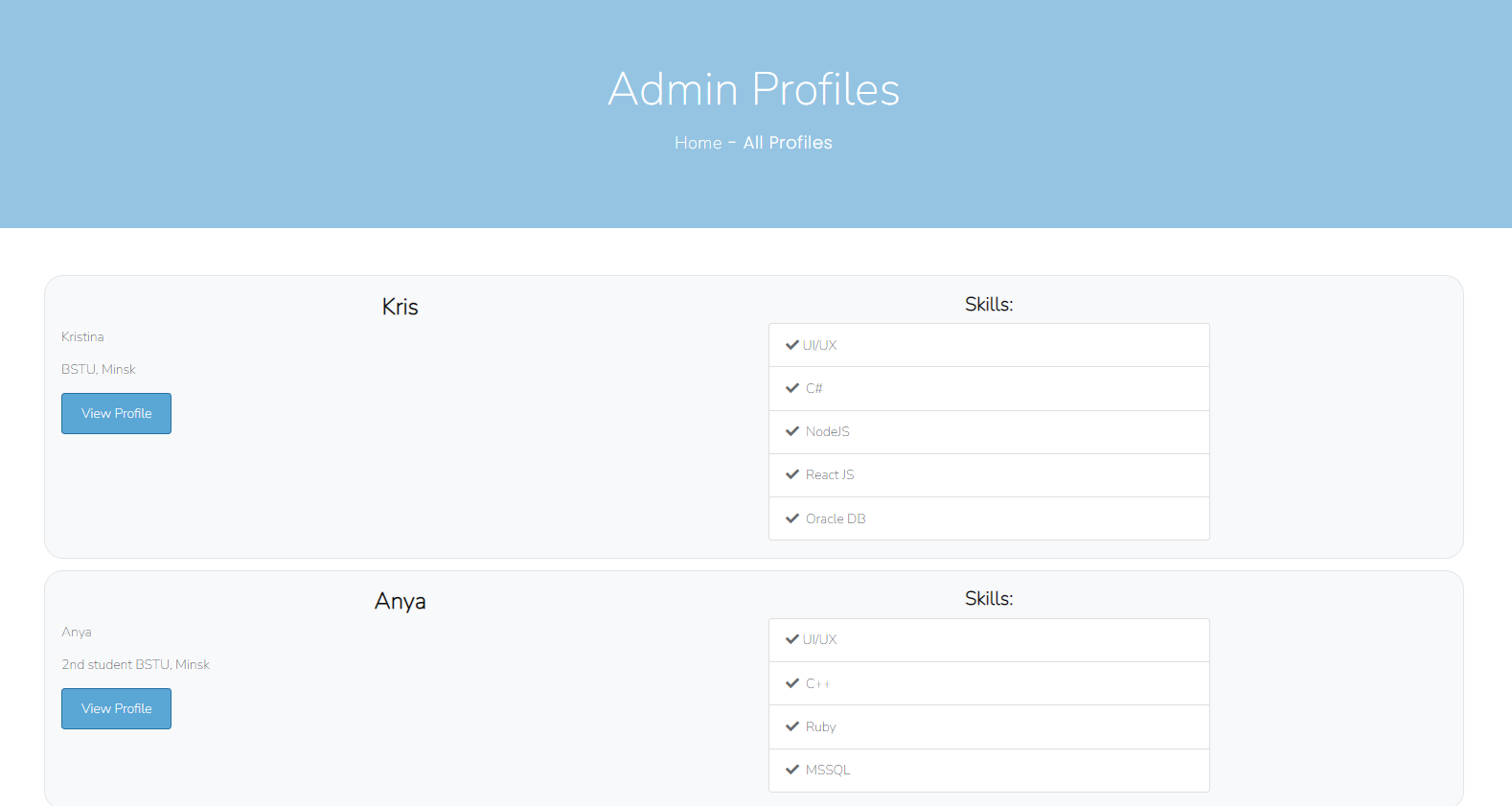


Рисунок 5.17 – Просмотр всех профилей

Для просмотра профиля необходимо нажать «View Profile» и появится страница, как мы уже видели на рисунке 5.9.

Для того, чтобы добавить курс, наводим на панель в меню «Courses» и выбираем «Add Course». После чего появится форма, для добавления курса, представленная на рисунке 5.18.

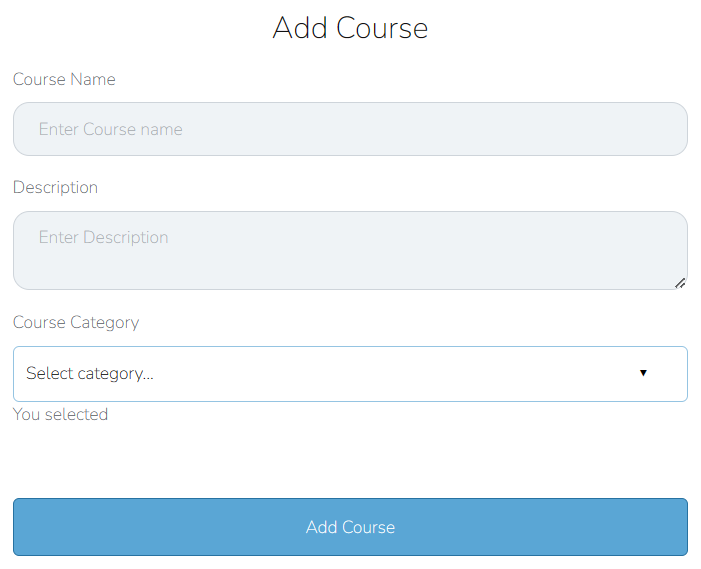


Рисунок 5.18 – Создание курса

Все поля, кроме «Description», должны быть обязательно заполнены, иначе курс не добавится. После заполнения всех полей, нажимаем «Add Course» и ваш курс будет сохранен.

Для добавления лекции к курсу, в этой же панели меню выбираем «Add Lecture». Появляется форма, представленная на рисунке 5.19.

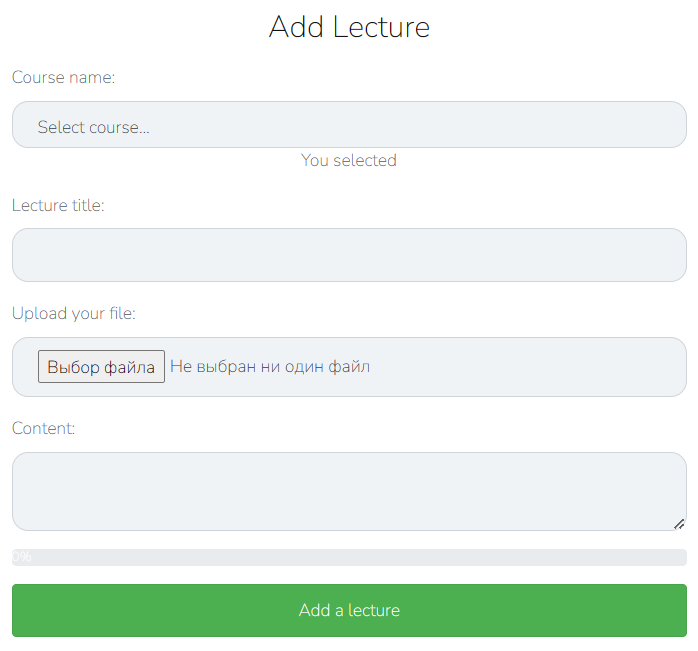


Рисунок 5.19 – Добавление лекции

Тут все поля являются обязательными: выбираем курс, к которому хотим добавить лекцию, добавляем название, текстовый контент лекции и прикрепляем видео материалы к лекции. После заполнения всех полей нажимаем «Add a lecture» и лекция будет добавлена к курсу.

Если прикрепить файл в формате, отличном от видео, то лекция не будет добавлена, а администратор увидит сообщение об ошибке.

Для просмотра всех курсов администратором – из этой же вкладки выбираем «All Courses». Выглядят эти вкладки одинаково, как представлено на рисунке 5.3.

Таким образом, был предоставлен функционал и возможности приложения, которые пользователь может использовать для достижения своих целей. Приложение предоставляет удобный и интуитивно понятный интерфейс, позволяющий пользователю взаимодействовать со всеми функциями приложения.

Были показаны основные возможности приложения, такие как создание, чтение, обновление, удаление, а также поиск информации.

В результате, пользователь получил все необходимые инструкции и руководства, чтобы эффективно использовать приложение и наслаждаться его функциональностью.

Раздел "Руководство пользователя" поможет пользователям ориентироваться и успешно взаимодействовать с приложением, улучшая их опыт использования.

# **Заключение**

В результате выполнения данной курсовой работы было разработано веб-приложение онлайн-школы программирования itProger. Благодаря наличию администратора можно контролировать и управлять действиями пользователей, создавать курсы, загружать по ним лекции, а также отслеживать статистику.

Программное средство предусматривает 2 группы пользователей, которые имеют следующие функциональные возможности.

Пользователь (USER):

* просмотр всех доступных курсов;
* добавить курс;
* просмотр добавленных курсов;
* просмотр лекций из курса;
* заполнить профиль;
* просмотр и редактирование своего профиля;
* получить уведомление о создании или обновления курса администратором;
* выйти из системы.

Администратор (ADMIN):

* просмотр статистики;
* добавить и редактировать курс;
* добавить лекцию по курсу;
* просмотр списка пользователей;
* редактирование информации о пользователях;
* просмотр профилей всех студентов;
* просмотр профиля конкретного студента;
* просмотр списка всех курсов;
* просмотр категорий курсов;
* создание категории;
* редактирование категории;
* просмотр студентов, записанных на курсы;
* записать студента на курс;
* удалить студента с курса;
* выйти из системы.

Разработанное программное средство реагирует на ошибочный ввод данных и на незаполненные обязательные поля, выводя при этом соответствующее сообщение об ошибке. Кроме того, оно имеет удобный и понятный интерфейс.

Серверная часть приложения (backend) реализована с помощью Node.js. Для написания клиентской части использовался фреймворк ReactJS. В качестве системы управления базами данных была использована PostgreSQL, ORM Prisma.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает корректно, а требования технического задания выполнены в полном объеме. Как клиентская, так и серверная части проекта имеют хороший потенциал для будущих модификаций, во многом благодаря грамотному структурированию, внимательности при написании кода, а также выборе известных и проверенных сопутствующих технологий.

# **Список используемых источников**

1. Node.js [Электронный ресурс] – Режим доступа: <https://nodejs.org/en/about/>– Дата доступа: 27.04.2023.
2. React [Электронный ресурс] – Режим доступа: https://react.dev/ – Дата доступа: 14.04.2023.
3. Socket.io [Электронный ресурс] – Режим доступа: https://socket.io/ – Дата доступа: 14.04.2023.
4. WebSocket API [Электронный ресурс] – Режим доступа: <https://learn.javascript.ru/websocket> – Дата доступа: 01.04.2023.
5. JavaScript [Электронный ресурс]. — Режим доступа: <https://ru.wikipedia.org/wiki/JavaScript> – Дата доступа: 20.04.2023.
6. Документация по React [Электронный ресурс]. — Режим доступа: https://react.dev/ – Дата доступа: 08.03.2023.
7. Документация по Redux [Электронный ресурс]. — Режим доступа: https://redux.js.org/ – Дата доступа: 08.05.2023.
8. Документация по Axios [Электронный ресурс]. — Режим доступа: https://axios-http.com/ru/docs/intro – Дата доступа: 08.03.2023.

# **Приложение A**

generator client {

  provider = "prisma-client-js"

}

datasource db {

  provider = "postgresql"

  url      = env("DATABASE\_URL")

}

model Category {

  id     Int      @id @default(autoincrement())

  name   String   @unique

  Course Course[]

}

model Course {

  id           Int            @id @default(autoincrement())

  name         String         @unique

  description  String?

  category     Int?

  Category     Category?      @relation(fields: [category], references: [id], onDelete: Cascade)

  Enrollment   Enrollment[]

  Lecture      Lecture[]

  Notification Notification[]

}

model Enrollment {

  id        Int     @id @default(autoincrement())

  user\_id   Int?

  course\_id Int?

  approved  Boolean @default(false)

  Course    Course? @relation(fields: [course\_id], references: [id], onDelete: Cascade)

  User      User?   @relation(fields: [user\_id], references: [id], onDelete: Cascade)

}

model Lecture {

  id        Int     @id @default(autoincrement())

  name      String

  videoLink String?

  content   String?

  course\_id Int?

  Course    Course? @relation(fields: [course\_id], references: [id], onDelete: Cascade)

}

Окончание приложения А

model User {

  id         Int          @id @default(autoincrement())

  name       String

  email      String       @unique

  password   String

  role       Roles        @default(USER)

  Enrollment Enrollment[]

  // Пример связи один-к-одному

  Profile    Profile?

}

model Profile {

  id             Int      @id @default(autoincrement())

  userId         Int      @unique

  handle         String

  skills         String[]

  bio            String?

  githubusername String?

  date           DateTime @default(now())

  user           User     @relation(fields: [userId], references: [id], onDelete: Cascade)

}

model Notification {

  id       Int      @id @default(autoincrement())

  courseId Int?

  content  String

  date     DateTime @default(now())

  course   Course?  @relation(fields: [courseId], references: [id], onDelete: Cascade)

}

enum Roles {

  ADMIN

  USER

}

# **Приложение Б**

import axios from "../utils/axios";  
import setAuthToken from "../utils/setAuthToken";  
import jwt\_decode from "jwt-decode";  
import { GET\_ERRORS, SET\_CURRENT\_USER } from "./types";  
  
export const registerUser = (userData, history) => (dispatch) => {  
 axios  
 .post("/auth/registration", userData)  
 .then((res) => history.push("/login"))  
 .catch((err) =>  
 dispatch({  
 type: GET\_ERRORS,  
 payload: err.response.data,  
 })  
 );  
};  
  
export const loginUser = (userData) => (dispatch) => {  
 axios  
 .post("/auth/login", userData)  
 .then((res) => {  
 const { token } = res.data; // извлекает токен из ответа сервера, если запрос успешный  
 const decoded = jwt\_decode(token); // декодирует полученный токен  
 const { id, roles } = decoded; // извлекает идентификатор пользователя  
 localStorage.setItem("jwtToken", token); // сохраняет токен в локальном хранилище  
 setAuthToken(token); // устанавливает токен в заголовок аутентификации  
 dispatch(setCurrentUser({ id, roles })); // устанавливает текущего пользователя с помощью лишь идентификатора пользователя  
 console.log(token);  
 })  
 .catch((err) => {  
 if (err.response && err.response.data) {  
 dispatch({  
 type: GET\_ERRORS,  
 payload: err.response.data,  
 });  
 }  
 });  
};  
  
export const setCurrentUser = (token) => {  
 return {  
 type: SET\_CURRENT\_USER,  
 payload: token };

};

Окончание приложения Б

export const logoutUser = () => (dispatch) => {  
 localStorage.removeItem("jwtToken"); // удаляем токен из local storage  
 setAuthToken(false); // очищаем заголовок аутентификации, чтобы указать, что пользователь не аутентифицирован  
 dispatch(setCurrentUser({})); // передаем пустой объект в качестве текущего пользователя  
};

//set default header for axios  
import axios from "axios";  
const setAuthToken = (token) => {  
 if (token) {  
 axios.defaults.headers.common["Authorization"] = `Bearer ${token}`; // Authorization - это ключ, token - это значение  
 } else {  
 delete axios.defaults.headers.common["Authorization"];  
 }  
};  
  
export default setAuthToken;

# **Приложение В**

async login(req, res) {  
 const { errors, isValid } = validateLoginInput(req.body);  
  
 // Check Validation  
 if (!isValid) {  
 return res.status(400).json(errors);  
 }  
  
 const { email, password } = req.body;  
 try {  
 const user = await DbClient.user.findUnique({  
 where: {  
 email: email,  
 },  
 });  
 if (!user) {  
 const errors = { email: `User with email ${email} does not exist` };  
 return res.status(400).json(errors);  
 }  
 const validPassword = await bcrypt.compareSync(password, user.password);  
 if (!validPassword) {  
 const errors = { password: "Invalid password" };  
 return res.status(400).json(errors);  
 }  
 const token = generateAccessToken(user.id, user.role);  
 return res.json({ token });  
 } catch (err) {  
 console.log(err);  
 console.log(err)  
 res.status(400).json({ message: "Login error" });  
 }  
}

# **Приложение Г**

|  |  |  |
| --- | --- | --- |
| Адрес | Метод | Описание |
| /auth/registration | POST | Запрос предназначен для отправки данных необходимых для регистрации. |
| /auth/login | POST | Запрос предназначен для отправки данных необходимых для аутентификации. |
| /auth/users | GET | Запрос предназначен для получения информации о всех зарегистрированных пользователях |
| /auth/is-admin | GET | Запрос предназначен для проверки является ли текущий пользователь администратором |
| /auth/currentUser | GET | Запрос предназначен для вывода информации о текущем пользователе |
| /auth/user/{id} | GET | Запрос предназначен для вывода информации о пользователе по его идентификатору |
| /auth/user/{id} | PUT | Запрос предназначен для обновления данных пользователя по его идентификатору |
| /auth/user/{id} | DELETE | Запрос предназначен для удаления пользователя |
| /category/add | POST | Запрос предназначен для добавления категории |
| /categories | GET | Запрос предназначен для вывода всех категорий |
| /category/{id} | GET | Запрос предназначен для вывода информации о категории по её идентификатору |
| /category/{id} | PUT | Запрос предназначен для обновления информации о категории по её идентификатору |

Продолжение таблицы

|  |  |  |
| --- | --- | --- |
| /category/{id} | DELETE | Запрос предназначен для удаления категории |
| /course/add | POST | Запрос предназначен для добавления курса |
| /courses | GET | Запрос предназначен для получения всех курсов |
| /course/{id} | GET | Запрос предназначен для вывода курса по его идентификатору |
| /course/{id} | PUT | Запрос предназначен для обновления информации курса по его идентификатору |
| /course/{id} | DELETE | Запрос предназначен для удаления курса |
| /course/deleteByCategory/{id} | DELETE | Запрос предназначен для удаления курса по идентификатору категории |
| /lecture/add | POST | Запрос предназначен для добавления лекции |
| /lectures | GET | Запрос предназначен для получения всех лекций |
| /uploadVideo | POST | Запрос предназначен для загрузки видео в лекцию |
| /notifications | GET | Запрос предназначен для вывода всех уведомлений пользователя |
| /profile/{id} | GET | Запрос предназначен для вывода профиля текущего пользователя |
| /profile/all | GET | Запрос предназначен для вывода всех профилей пользователей |
| /profile/ | POST | Запрос предназначен для создания профиля пользователя или обновления существующего |

Продолжение таблицы

|  |  |  |
| --- | --- | --- |
| /profile/user/{user\_id} | GET | Запрос предназначен для вывода профиля пользователя по идентификатору пользователя |
| /profile/handle/{handle} | GET | Запрос предназначен для вывода профиля конкретного пользователя по его никнейму |
| /profile/{id} | DELETE | Запрос предназначен для удаления профиля |
| /enrollmentbystudent/{id} | GET | Запрос предназначен для вывода записей пользователя на курсы по его идентификатору |
| /enrollments | GET | Запрос предназначен для вывода всех записей на курсы пользователей |
| /checkenrollment/{id} | GET | Запрос предназначен для проверки записан ли пользователь по идентификатору на курс |
| /enroll/add | POST | Запрос предназначен для добавления пользователя на курс администратором |
| /enrollmentbystudent/add/{courseId} | POST | Запрос предназначен для записи на курс пользователя |
| /enrollment/{id} | DELETE | Запрос предназначен для удаления пользователя с курса администратором |
| /enrollmentbystudent/delete/{courseId} | DELETE | Запрос предназначен для удаления пользователя с курса самим пользователем |