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:**

Веб-приложение «itProger»

Выполнил студент Миневич Кристина Викторовна

(Ф.И.О.)

Руководитель проекта ст. преп. Дубовик М. В.

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

Заведующий кафедрой к.т.н., доц. Смелов В.В.

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

Консультанты ст. преп. Дубовик М. В.

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

Норм контролёр ст. преп. Дубовик М. В.

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

Курсовой проект защищен с оценкой

Минск 2023

**Содержание**

[Введение 3](#_Toc135266197)

[1 Аналитический обзор литературы и постановка задачи 4](#_Toc135266198)

[1.1 Обзор прототипов 4](#_Toc135266199)

[1.1.1 SkillBox 4](#_Toc135266200)

[1.1.2 TeachMeSkills 5](#_Toc135266201)

[1.1.3 GeekBrains 5](#_Toc135266202)

[1.1.4 Udemy 6](#_Toc135266203)

[1.2 Постановка задачи 7](#_Toc135266204)

[1.3 Описание используемых технологий 8](#_Toc135266205)

[2 Разработка архитектуры проекта 10](#_Toc135266206)

[2.1 Обобщенная структура проекта 10](#_Toc135266207)

[2.2 Диаграмма UML 11](#_Toc135266208)

[2.3 Модель базы данных 12](#_Toc135266209)

[2.4 Идентификация и авторизация 16](#_Toc135266210)

[3 Создание (реализация) программного средства 17](#_Toc135266211)

[3.1 Реализация серверной части 17](#_Toc135266212)

[3.2 Обмен сообщениями с клиентом в реальном времени 19](#_Toc135266213)

[3.3 Реализация клиентской части 20](#_Toc135266214)

[3.4 Конечные точки 22](#_Toc135266215)

[4 Тестирование 23](#_Toc135266216)

[4.1 Форма входа 23](#_Toc135266217)

[4.2 Форма регистрации 24](#_Toc135266218)

[4.3 Форма создания профиля пользователя 25](#_Toc135266219)

[5 Руководство пользователя 27](#_Toc135266220)

[5.1 Пользователь 27](#_Toc135266221)

[5.2 Администратор 31](#_Toc135266222)

[Заключение 36](#_Toc135266223)

[Список используемых источников 37](#_Toc135266224)

[Приложение A 38](#_Toc135266225)

[Приложение Б 40](#_Toc135266226)

[Приложение В 42](#_Toc135266227)

[Приложение Г 43](#_Toc135266228)

# **Введение**

Электронное обучение развивается, практикуется и совершенствуется уже много лет. Тема актуальна как в СМИ, так и среди научного и образовательного сообществ.

Многие компании, предоставляющие свободное обучение своим сотрудникам, имеют свою внутреннюю закрытую систему, в которой предоставлена информация о тренингах, которые может предложить для них компания. Чаще всего это компании, ориентированные на разработку ПО. Также существуют сервисы, находящиеся в свободном доступе, предназначенные для предоставления доступа к тренингам всем желающим на платной или бесплатной основе, например, Udemy, PluralSight и т.д.

Сегодня огромен выбор образовательных программ. Каждый желающий может записаться на онлайн-курсы по интересам, пройти различные дистанционные программы. Подобный формат позволяет освоить понравившуюся специальность огромному количеству людей, не имеющих достаточно времени и финансов для получения высшего образования классическим способом. Также онлайн-обучение дает возможность учиться у тех, кто находится от тебя территориально далеко и в любое время. Благодаря ему можно найти курс подходящего вам уровня и подходящего автора – выбор огромен.

Изучение программирования становится все более доступным благодаря непрерывному росту количества онлайн ресурсов, которые помогают в этом всем желающим. Плюс таких ресурсов — в неограниченном количестве знаний, которые они могут дать и в высокой квалификации преподавателей. Минус — мотивация должна возникнуть у вас сама и качество полученных знаний зависит лишь от того, сколько усилий вы приложили.

При этом важно понимать, что любые формы обучения имеют свои плюсы и минусы, и онлайн-курсы – не исключение. А значит, тема совершенствования электронного обучения будет оставаться актуальной и получать свое развитие как с теоретической, так и с технологической точек зрения. Именно в этом, я считаю, и заключается актуальность выбранной темы — разработка веб-приложения для просмотра доступных тренингов «itProger».

# **1 Аналитический обзор литературы и постановка задачи**

В наше время большой популярностью пользуются сервисы, упрощающие получение информации о той или иной предметной области. Множество пользователей предпочитают вместо посещения самих компаний, предоставляющих обучение, посещать веб-сервисы, где удобно расположена информация. Популярность подобных сервисов достигается тем, что их использование очень удобно и не требует особых усилий, а также в связи с тем, что получить доступ к информации можно из любой точки мира.

Основные достоинства интернет-приложений, которые делают их выгодными и все более популярными:

* экономия времени;
* удобность использования;
* доступность.

На сегодняшний день классические тренинг-центры становятся все менее популярными. На смену им приходят онлайн-сервисы.

## **1.1 Обзор прототипов**

Рассмотрим примеры web-приложений, на которых можно получить информацию о тренингах.

1.1.1 SkillBox

Skillbox – образовательная платформа, в котором обучают онлайн 90+ программам для получения востребованных IT профессий во всём мире.

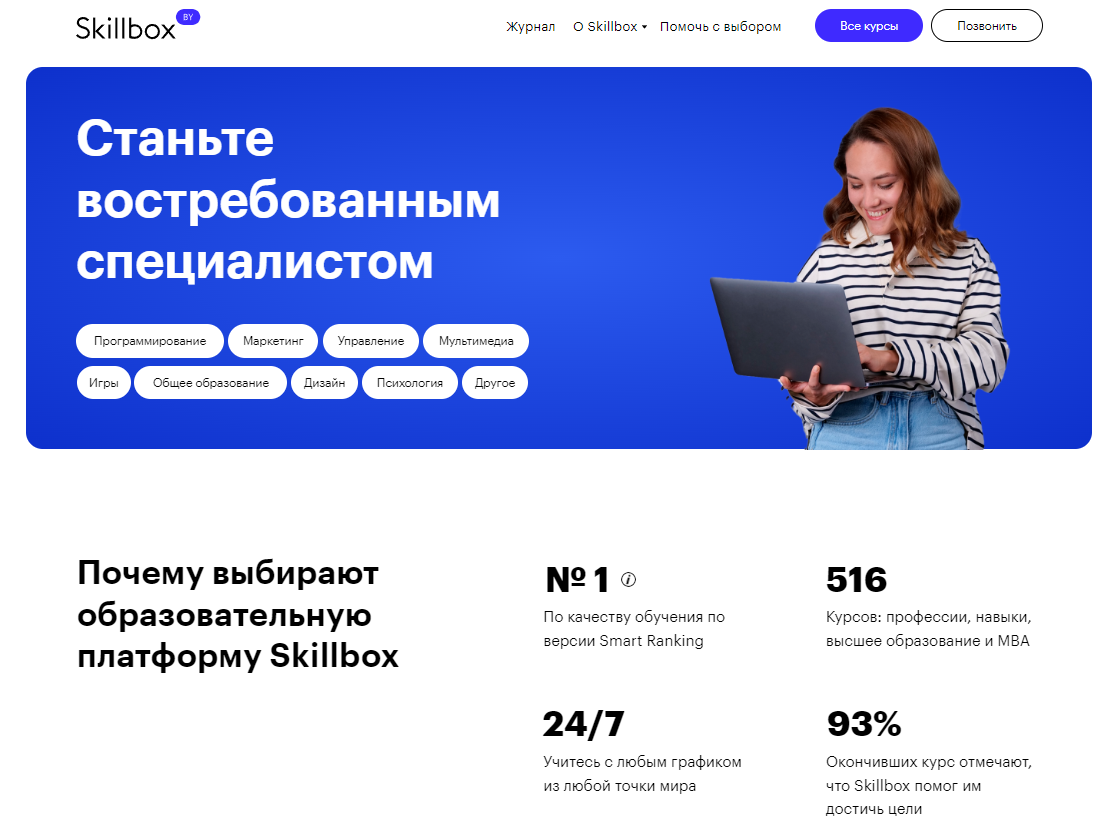


Рисунок 1.1 – Образовательная платформа Skillbox

Здесь обучают профессиям, которые будут всегда востребованы в мире IT. В Skillbox разноформатное обучение. Самым основным являются видеолекции. В процессе обучения на платформе пользователи закрепляют знания с помощью тестов и практических заданий, которые проверяет куратор.

Тут обучают больше 20+ профессиям, связанных с программированием и возможностью трудоустроиться по этому направлению. Вам на выбор предлагается стать разработчиками PHP, Android, Java, веб, Python, Frontend, C#, Fullstack, iOS, 1С, а также геймдизайнерами, тестировщиками или специалистами по анализу данных и кибербезопасности.

Сайт, представленный на рисунке 1.1, имеет интуитивно-понятный интерфейс, приятный дизайн и обладает высокой функциональностью.

1.1.2 TeachMeSkills

TeachMeSkills –это школа программирования, где учат востребованным сегодня знаниям. Все программы составлены Senior- и Lead-разработчиками ведущих IT компаний специально для новичков в IT*.* Главная страница сайта представлена на рисунке 1.2.

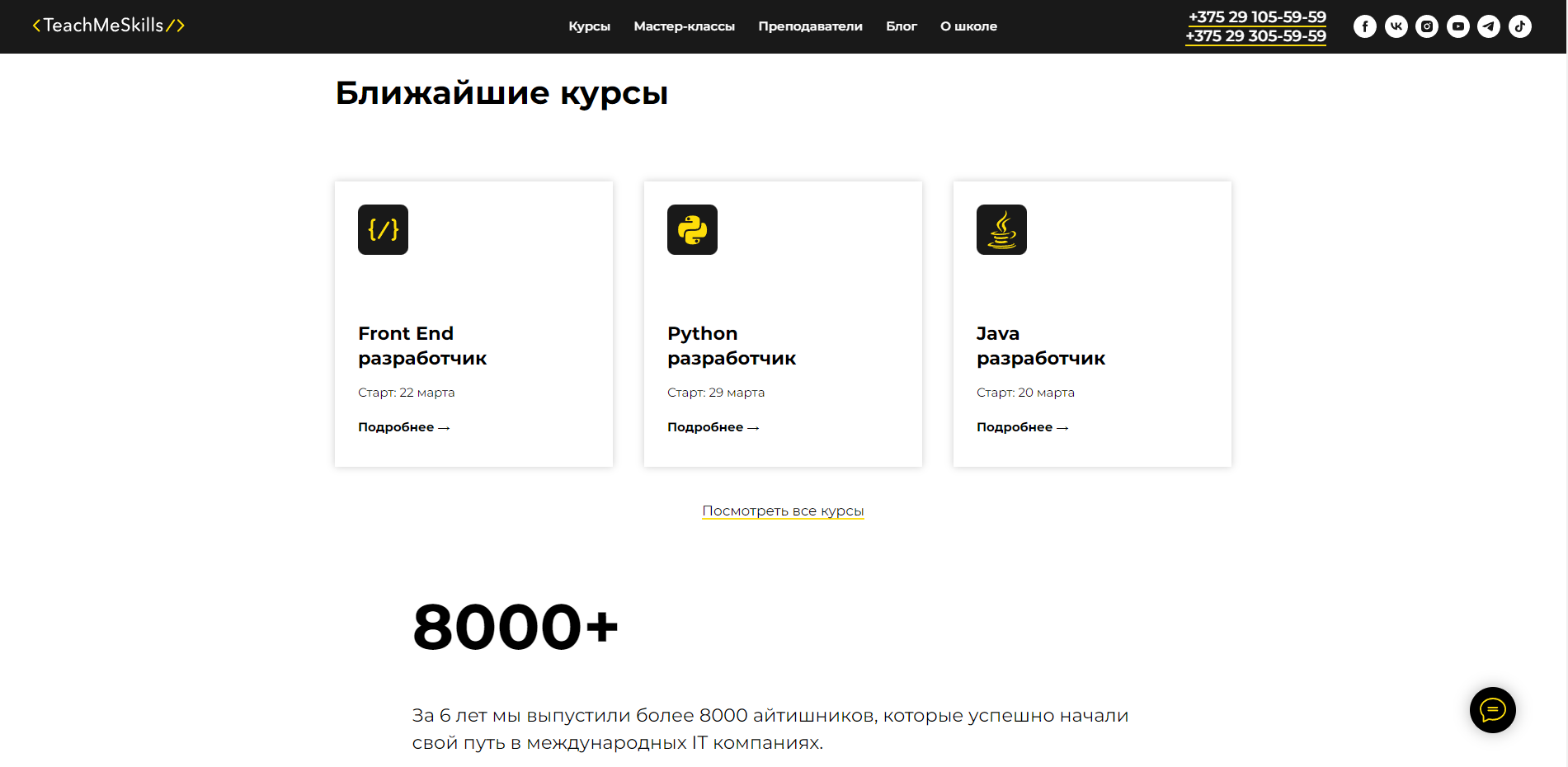


Рисунок 1.2 – Главная страница TeachMeSkills

Школа предлагает как онлайн-обучение, так и оффлайн в Минске. Имеется возможность ознакомиться с преподавательским составом, а также почитать интересные статьи о сфере IT.

Сайт обладает динамическими стилями и информативным слайдером на главной странице.

1.1.3 GeekBrains

В этой онлайн-школе можно найти практически всё, что касается прямо или косвенно IT-сферы. Тут обучают не только программированию или дизайну, но и интернет-маркетингу или системному администрированию. Какой курс GeekBrains не открой – везде есть расписанная программа со всеми подробностями и отзывы от пользователей, проходивших тот или иной курс. Для студентов предусмотрена доступная программа стажировок и получение сертификатов о прохождении курсов. Главная страница сайта представлена на рисунке 1.3.

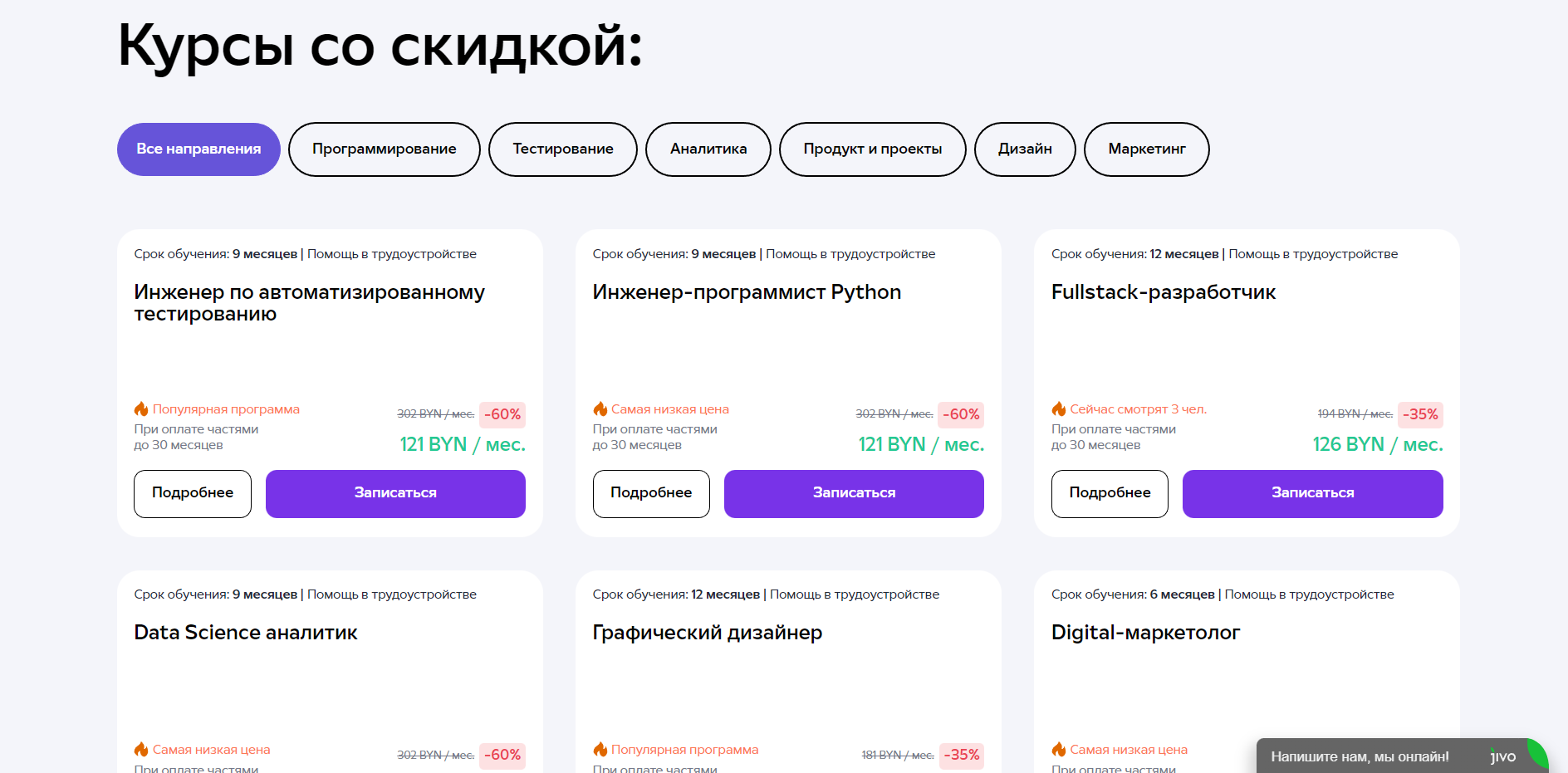


Рисунок 1.3 – Главная страница GeekBrains

Преимущество этого образовательного портала в доступе ко множеству бесплатного контента, но самый козырь – это возможность обучения у топовых айтишников. Осилить современные профессии в мире диджитал можно на факультетах веб-, iOS-, Go-, Java- и Python-разработки, искусственного интеллекта, DevOps и других.

Если Вы новичок и не можете понять, что же нравится и подходит именно вам, команда GeekBrains готова безвозмездно помочь и определиться с будущим выбором.

Онлайн-уроки, разбор заданий вместе с преподавателями и возможность общение со своими одногруппниками – делает обучение похожее на офлайновую реальность. И только тогда, когда ученики подтверждают свои умения путем тестирования они получают сертификат.

1.1.4 Udemy

Ещё одна платформа для изучения программирования – Udemy.com. Здесь доступно более двухсот тысяч онлайн-видеокурсов, а новые появляются каждый месяц! Имеется доступ к личному кабинету студента, подборка курсов на четырнадцати языках и возможность просматривать информацию об преподавателях, создавших курс.

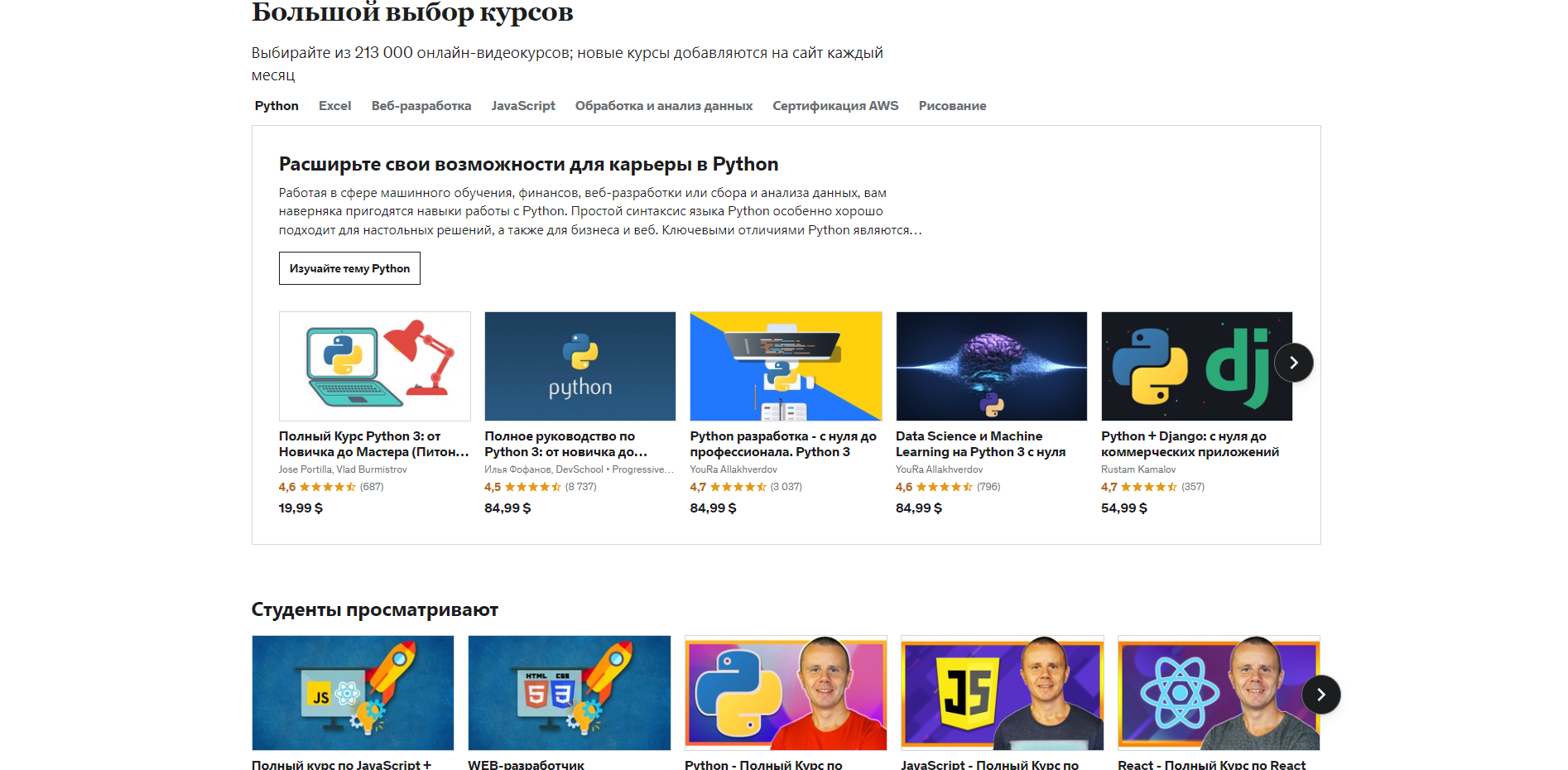


Рисунок 1.4 – Платформа Udemy.com

Ещё одним преимуществом данной платформы является то, что студент может проходить курс в удобном для него графике: временных ограничений нет.

Для того, чтобы понять, подходит для пользователя курс или нет, имеется возможность ознакомиться с описанием курса, а также в материалах курса можно воспользоваться предпросмотром некоторых лекций. Студенты, записанные на курс, имеют возможность оставлять отзывы, что тоже является преимуществом выбрать именно данную школу для изучения программирования.

## **1.2 Постановка задачи**

Анализ существующих сайтов и приложений позволил определиться с постановкой задачи.

Итогом разработки должно стать веб-приложение для просмотра информации о курсах, предлагаемых тренинг-центром IT-компании и изменения (администрирования) данной информации.

Разумеется, необходимо разработать несколько интерфейсов: для пользователя и для администратора.

Интерфейс пользователя упрощен. Он должен давать возможность посмотреть каталог курсов и лекций по ним, а также изменить личную информацию о самом пользователе.

Интерфейс администратора должен предусматривать всевозможные операции с содержимым сайта, описаниями. Сайт должен быть выполнен в спокойных ненавязчивых тонах.

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

– сохранять пользовательскую информацию в базе данных;

– получать информацию из базы данных;

– создавать пользовательский интерфейс для взаимодействия с базой данных;

– регистрировать и авторизовывать пользователей;

– при необходимости удалять информацию из базы данных;

– просматривать уроки/лекции;

– добавлять уроки;

– добавлять материалы в урок.

## **1.3 Описание используемых технологий**

При проектировании программного средства в качестве программной платформы для сервера был выбран Node.js. Node.js – программная платформа для разработки серверных web-приложений на основе движка JavaScript Chrome V8. Является средой исполнения приложений на JavaScript. Также ориентирована на события, поддерживает асинхронность и является однопоточной. Также был использован пакет Express.js, который является популярным веб-фреймворком, написанным на JavaScript и работающий внутри среды исполнения Node.js. Express не мешает общей производительности приложения т.к. представляет собой тонкий слой основных функций веб-приложений. Также в нём удобно настраивать маршруты приложения, поскольку он использует уже давно известные методы http.

Рассмотрим основные особенности Node.js.

– Скорость. Одной из основных привлекательных особенностей Node.js является скорость. JavaScript-код, выполняемый в среде Node.js, может быть в два раза быстрее, чем код, написанный на компилируемых языках, вроде C или Java, и на порядки быстрее интерпретируемых языков наподобие Python или Ruby.

– Простота. Платформа Node.js проста в освоении и использовании.

– Движок V8. В основе Node.js, помимо других решений, лежит оперсорсный JavaScript-движок V8 от Google, применяемый в браузере Google Chrome и в других браузерах.

– Асинхронность. JavaScript значительно упрощает написание асинхронного и неблокирующего кода с использованием единственного потока, функций обратного вызова и подхода к разработке, основанной на событиях.

Для проектирования базы данных используется система управления базами данных Prisma. Prisma – это объектно-реляционная система управления базами данных (ORM) с открытым исходным кодом, предназначенная для разработки веб-приложений. Она позволяет разработчикам работать с базами данных с помощью удобного и интуитивно понятного API на языке TypeScript или JavaScript. Prisma предоставляет множество преимуществ, которые делают его удобным инструментом для работы с базами данных:

* Простота использования: Prisma обеспечивает интуитивно понятный и удобный интерфейс для работы с базами данных, что делает его простым в использовании даже для новичков.
* Мощная система миграций: Система миграций Prisma позволяет легко изменять структуру баз данных, что делает ее удобной для поддержки и масштабирования приложений.
* Высокая производительность: Prisma использует современные технологии и архитектуру, чтобы обеспечить высокую производительность работы с базами данных.
* Гибкость: Prisma поддерживает работу с различными базами данных, что делает его удобным инструментом для разработки приложений с различными требованиями.
* Автоматически генерируемый код: Prisma генерирует код на основе схемы базы данных, что позволяет сократить время и усилия, затрачиваемые на написание кода.

Для написания клиентской части использовалась библиотека JavaScript – ReactJS. React — это библиотека для создания пользовательских интерфейсов. Он поощряет создание многократно используемых компонентов пользовательского интерфейса, представляющие данные, которые со временем изменяются. Многие люди используют React как V в MVC. React абстрагирован от DOM, предлагая более простую модель программирования и лучшую производительность. React реализует односторонний реактивный поток данных, который уменьшает объем шаблонов и упрощает понимание кода, по сравнению с традиционной привязкой данных.

Преимущества React:

– Виртуальный DOM: React использует виртуальный DOM (Document Object Model), который представляет легковесную копию реального DOM. React эффективно обновляет только необходимые части виртуального DOM и применяет эти изменения к реальному DOM, что улучшает производительность и отзывчивость приложения.

– Однонаправленный поток данных: в React данные передаются от родительских компонентов к дочерним компонентам через пропсы. Это упрощает отслеживание и понимание потока данных в приложении, а также делает его более предсказуемым и легким для отладки.

– Реактивные обновления: React автоматически обновляет пользовательский интерфейс при изменении состояния или пропсов компонентов. Это позволяет создавать динамические и интерактивные приложения без необходимости явного обновления DOM.

– Расширяемость: React обладает широким экосистемой сторонних библиотек и инструментов, которые позволяют расширять его функциональность и упрощать разработку. Например, с помощью библиотеки Redux можно реализовать управление состоянием приложения, а с помощью React Router - роутинг и навигацию.

# **2 Разработка архитектуры проекта**

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.

Основные задачи разработки архитектуры проекта:

* Выделение программных подсистем и отображение на них внешних функций (заданных по внешнем описании) программного средства;
* Определение способов взаимодействия между выделенными программными подсистемами.

С учетом принимаемых на этом этапе решений производится дальнейшая конкретизация и функциональных спецификаций.

## **2.1 Обобщенная структура проекта**

Курсовой проект построен на клиент-серверной архитектуре. Клиент-серверная архитектура – это архитектура, которая подразумевает две компоненты: клиент и сервер. Клиент является инициатором соединения. В качестве сервера будет выступать приложение на Node.js. В качестве клиента будет выступать приложение c асинхронным UI (React). Схема развёртывания представлена на рисунке 2.1.

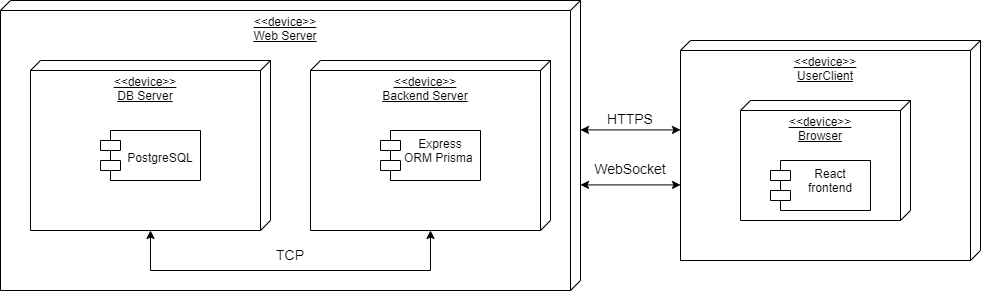


Рисунок 2.1 – Схема развёртывания

Выбрана клиент-серверная архитектура, так как она хорошо подходит для разработки web-приложений, и нагрузка будет распределена между клиентом и сервером. Также отсутствует дублирование кода сервера на клиенте. Все данные хранятся на сервере, который обычно защищен лучше большинства клиентов. Еще, так как все вычисления выполняются на сервере, снижаются системные требования к клиенту.

Таким образом, сервер принимает запрос клиента, обрабатывает запрос, работает с базой данных и возвращает клиенту ответ. Клиент будет являться инициатором соединения с сервером.

## **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.

В языке UML вариант использования изображается в виде овала, помеченного именем представляемого варианта. Варианты использования могут быть связаны с участвующими в них действующими лицами (actors), изображаемыми в виде человечков и представляющими различные роли пользователей системы или внешние системы, взаимодействующие с ней.

Варианты использования могут быть связаны друг с другом тремя видами связей: обобщением (generalization), расширением (extend relationship) и включением (include relationship).

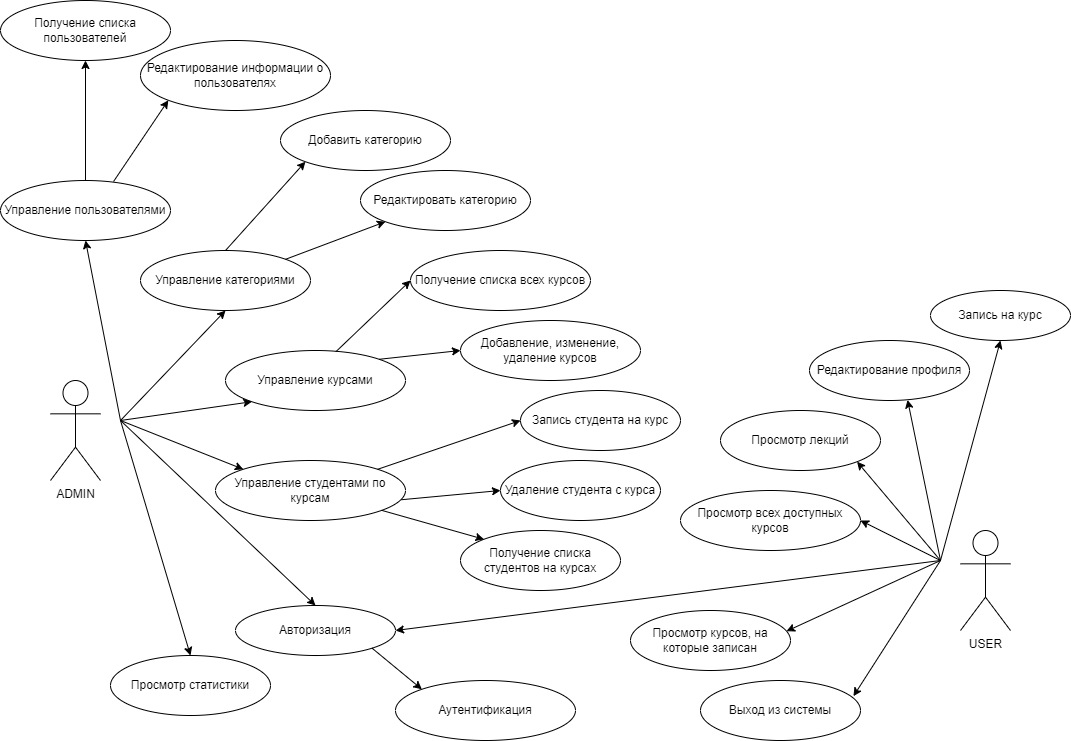


Рисунок 2.2 – Диаграмма вариантов использования

На данной диаграмме представлены 2 варианта юзеров, которые имеют различные возможности.

В первом варианте (ADMIN) есть возможности авторизации, просмотра статистики по записям на курсы и количеству курсов по каждой категории, управления пользователями, категориями, курсами и записям студентов на курсы.

Во втором же варианте (USER) можно авторизоваться, просмотреть все доступные курсы, созданные администратором, записаться на курсы, просмотреть лекции по курсам, создать или обновить профиль пользователя, а также просмотреть курсы, на которые записан текущий пользователь.

## **2.3 Модель 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 «itProger». Для её создания использовалась система управления реляционными базами данных PostgreSQL.

Выбор данного СУБД произошел по нескольким очевидным причинам: производительность и масштабируемость (она может обрабатывать сложные запросы эффективно и обеспечивать стабильную работу даже при высоких нагрузках), расширяемость (множество встроенных и сторонних расширений, которые позволяют расширять функциональность базы данных), безопасность (обладает широким набором механизмов безопасности, включая аутентификацию, авторизацию и шифрование данных), надёжность (обеспечивает механизмы сохранности данных, включая транзакции, резервное копирование и восстановление, что позволяет избежать потери данных и обеспечить их целостность).

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 «itProger» было разработано 7 таблиц:

User – для хранения информации о пользователе;

Profile – данные для профиля пользователя;

Category – названия категорий для курсов;

Course – информация о курсе;

Lecture – данные из лекции;

Enrollment – информация о студентах, записанных на определенные курсы;

Notification – уведомления, приходящие пользователю, когда администратор создаёт или изменяет курс.

Взаимосвязь всех таблиц проектируемой базы данных представлена на рисунке 2.3.

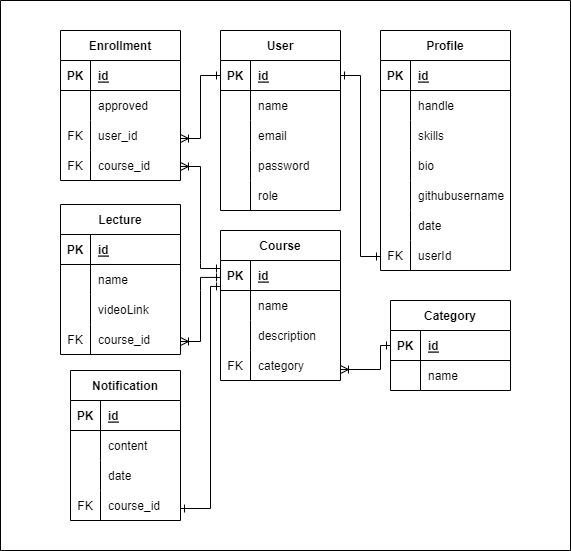


Рисунок 2.3 – Взаимосвязь таблиц базы данных

Листинг создания базы данных, используя ORM Prisma, представлен в приложении А. Рассмотрим каждую таблицу по отдельности.

Таблица User представляет собой данные о пользователе. Перечень полей таблицы User приведен в таблице 2.1

Таблица 2.1 – Описание полей таблицы User

|  |  |
| --- | --- |
| Поле | Типы данных, ограничения целостности |
| id | integer nextval() PRIMARY KEY |
| name | text NOT NULL |
| email | text NOT NULL |
| password | text NOT NULL |
| role | enum DEFAULT ‘USER’ NOT NULL |

Охарактеризуем каждое поле данной таблицы:

– id – идентификатор пользователя, первичный ключ;

– name – имя пользователя;

– email – адрес электронной почты пользователя;

– password – пароль пользователя;

– role – роль (USER или ADMIN).

Таблица Category представляет собой названия категорий курсов. Перечень полей таблицы Category приведен в таблице 2.2

Таблица 2.2 – Описание полей таблицы Category

|  |  |
| --- | --- |
| Поле | Типы данных, ограничения целостности |
| id | integer nextval() PRIMARY KEY |
| name | text NOT NULL |

Охарактеризуем каждое поле таблицы Category:

– id – идентификатор категории, первичный ключ;

– name – название категории.

Таблица Course представляет собой данные о курсе. Перечень полей таблицы Course приведен в таблице 2.3

Таблица 2.3 – Описание полей таблицы Course

|  |  |
| --- | --- |
| Поле | Типы данных, ограничения целостности |
| id | integer nextval() PRIMARY KEY |
| name | text NOT NULL |
| description | text |
| category | integer FOREIGN KEY |

Охарактеризуем каждое поле таблицы Course:

– id – идентификатор курса, первичный ключ;

– name – название курса;

– description – описание курса;

– category – категория, к которой относится данный курс, внешний ключ.

В таблице Lecture хранятся данные о лекции. Перечень полей таблицы Lecture приведен в таблице 2.4

Таблица 2.4 – Описание полей таблицы Lecture

|  |  |
| --- | --- |
| Поле | Типы данных, ограничения целостности |
| id | integer nextval() PRIMARY KEY |
| name | text NOT NULL |
| content | text |
| course\_id | integer FOREIGN KEY |
| videoLink | text |

Охарактеризуем каждое поле таблицы Lecture:

– id – идентификатор лекции, первичный ключ;

– name – название лекции;

– content – контент лекции (её краткое описание);

– course\_id – идентификатор курса, к которому относится данная лекция, внешний ключ;

– videoLink – ссылка на видео данной лекции.

Таблица Profile представляет собой профиль пользователя. Перечень полей таблицы Profile приведен в таблице 2.5

Таблица 2.5 – Описание полей таблицы Profile

|  |  |
| --- | --- |
| Поле | Типы данных, ограничения целостности |
| id | integer nextval() PRIMARY KEY |
| userId | integer NOT NULL FOREIGN KEY |
| handle | text |
| skills | text[] |
| bio | text |
| githubusername | text |
| date | timestamp(3) without time zone NOT NULL DEFAULT CURRENT\_TIMESTAMP |

Охарактеризуем каждое поле таблицы Profile:

– id – идентификатор профиля пользователя, первичный ключ;

– userId – идентификатор пользователя, заполняющего профиль, внешний ключ;

– handle – заголовок профиля;

– skills – список навыков пользователя;

– bio – описание пользователя;

– githubusername – никнейм пользователя на GitHub;

– date – время создания и заполнения профиля.

Таблица Enrollment предназначена для хранения записей пользователей на курсы. Перечень полей таблицы Enrollment приведен в таблице 2.6

Таблица 2.6 – Описание полей таблицы Enrollment

|  |  |
| --- | --- |
| Поле | Типы данных, ограничения целостности |
| id | integer nextval() PRIMARY KEY |
| user\_id | integer NOT NULL FOREIGN KEY |
| course\_id | integer NOT NULL FOREIGN KEY |
| approved | boolean NOT NULL DEFAULT false |

Охарактеризуем каждое поле таблицы Enrollment:

– id – идентификатор записи, первичный ключ;

– user\_id – идентификатор пользователя, внешний ключ;

– course\_id – идентификатор курса, внешний ключ;

– approved – идентификатор записи, показывающий записан ли пользователь на курс или нет;

Таблица Notificationпредназначена для хранения уведомлений. Перечень полей таблицы Notification приведен в таблице 2.7

Таблица 2.7 – Описание полей таблицы Notification

|  |  |
| --- | --- |
| Поле | Типы данных, ограничения целостности |
| id | integer nextval() PRIMARY KEY |
| courseId | integer NOT NULL |

Продолжение таблицы 2.7

|  |  |
| --- | --- |
| content | text NOT NULL |
| date | timestamp(3) without time zone NOT NULL DEFAULT CURRENT\_TIMESTAMP |

Охарактеризуем каждое поле таблицы Notification:

– id – идентификатор уведомления, первичный ключ;

– courseId – идентификатор курса, внешний ключ;

– content – текст уведомления;

– date – дата создания уведомления;

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

## **2.4 Идентификация и авторизация**

Для выполнения задач при разработке приложения, существует необходимость регистрации, авторизации и идентификации, для этой задачи используется JWT.

JSON Web Token (JWT) — это открытый стандарт для создания токенов доступа, основанный на формате JSON. Как правило, используется для передачи данных для аутентификации в клиент-серверных приложениях. Токены создаются сервером, подписываются секретным ключом и передаются клиенту, который в дальнейшем использует данный токен для подтверждения своей личности.

Состоит из трех частей: заголовка, полезной нагрузки и зашифрованных данных, которые и являются уникальными благодаря секретным ключам.

Аутентификация – процедура проверки подлинности идентификации пользователя. В данном случае происходит проверка путем сравнения введенного пароля с паролем, который сохранен в базе данных и соответствует email. Функция принимает email и пароль, дальше идет поиск пользователя с введенным email в базе данных и проверка пароля с помощью библиотеки BCrypt. В случае успеха происходит формирование JSON Web Token (JWT), который в дальнейшем будет использоваться для доступа к приложению, и отправка его пользователю.

Исходный код конфигурации JWT представлен в листинге 2.4

const generateAccessToken = (id, roles) => {  
 const payload = {  
 id,  
 roles: Array.isArray(roles) ? roles : [roles],  
 };  
 return jwt.sign(payload, process.env.SECRET, { expiresIn: "12h" });

Листинг 2.1 – Конфигурация JWT на сервере

На стороне клиента, при получении токена, он сохраняется в localStorage и устанавливается в заголовок «Authorization». После чего мы можем декодировать токен и получить оттуда информацию о пользователе. Методы, выполняющие эти функции, представлены в приложении Б.

# **3 Создание (реализация) программного средства**

Решение представлено двумя проектами: backend (серверная часть на Node.js) и frontend (клиентская часть – ReactJS). Для старта сначала запускается серверная часть, затем – клиентская.

## **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.

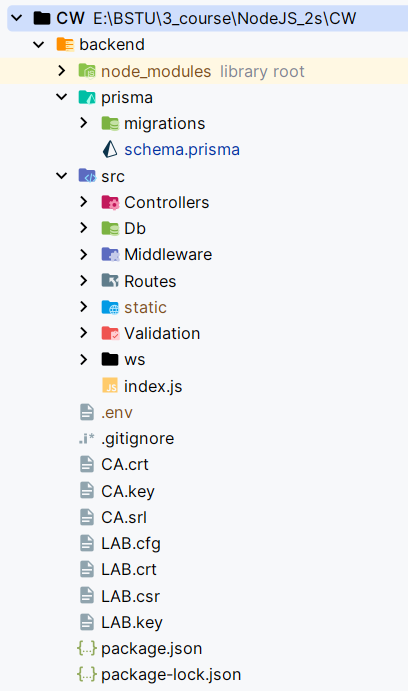


Рисунок 3.1 – Структура директорий проекта сервера

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.

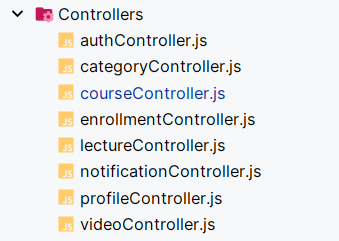


Рисунок 3.2 – Структура директории «Сontrollers»

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

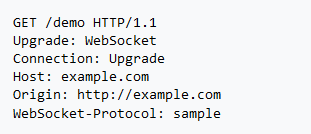


Рисунок 3.4 – Запрос клиента для установки соединения WebSocket

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

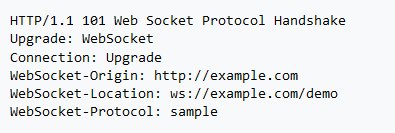


Рисунок 3.5 – Ответ от сервера, в случае успешной смены протокола

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.

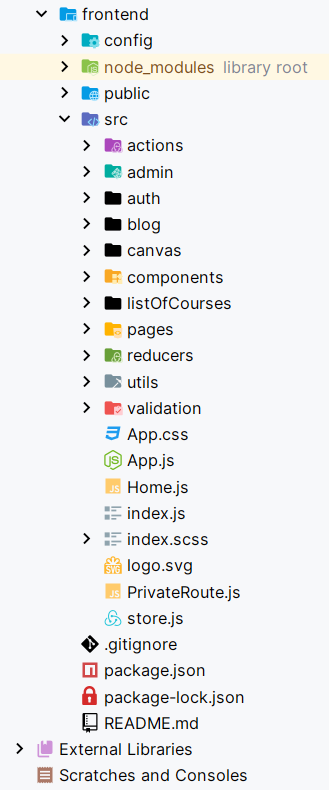


Рисунок 3.6 – Структура директорий проекта клиента

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**

В этой главе рассмотрим основные элементы интерфейса и протестируем интерфейс веб-приложения.

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

## **4.1** **Форма входа**

При переходе в браузере к приложению, пользователь видит форму для ввода адреса электронной почты и пароля. Присутствует также ссылка для перехода к форме регистрации нового пользователя. Пользователи, которые успешно прошли проверку будут переадресованы на главную страницу приложения. Интерфейс формы входа можно увидеть на рисунке 4.1.

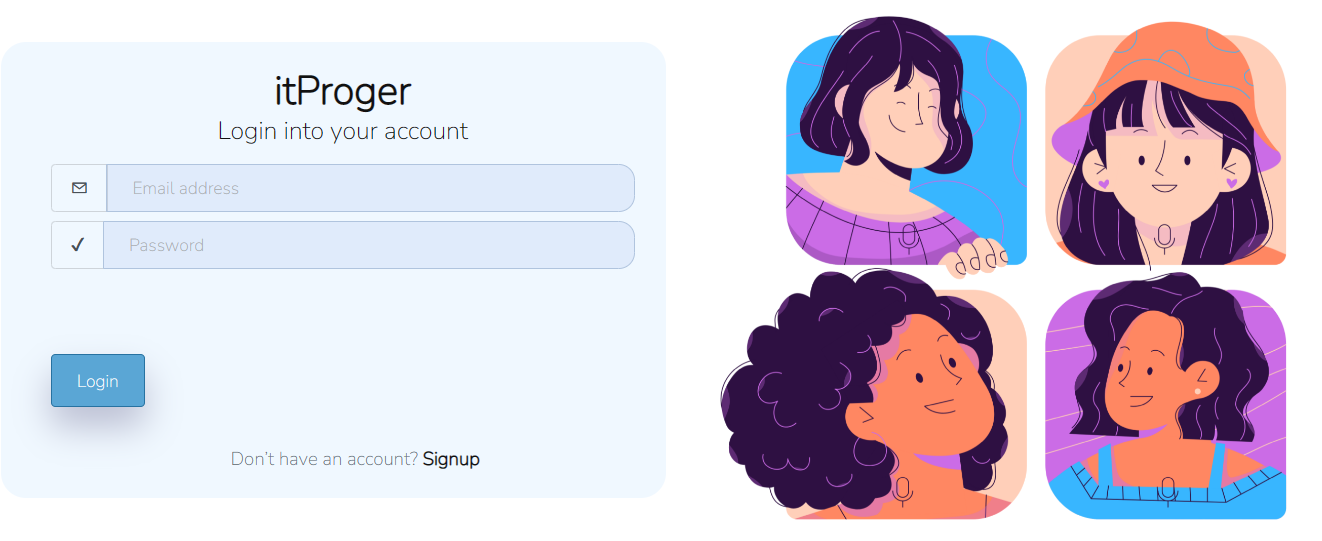


Рисунок 4.1 – Форма входа

При вводе правильного логина и пароля пользователь перенаправляется на приветственную страницу приложения, в случае неверного ввода данных, появляются красные ошибки внизу поля, сигнализируя о неправильном вводе email или пароля, такой пример показан на рисунке 4.2.

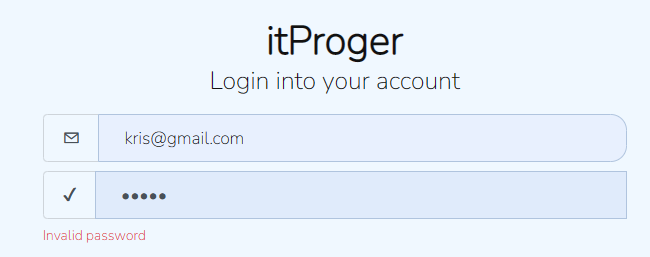


Рисунок 4.2 – Оповещение пользователя при неправильном вводе пароля

Существует также проверка на существование пользователя с введённым адресом электронной почты. При попытке войти в приложение через пользователя с несуществующим email, появляется ошибка, изображённая на рисунке 4.3.

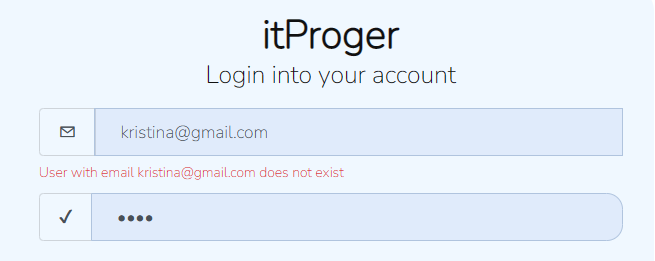


Рисунок 4.3 – Оповещение пользователя при вводе несуществующего email

## **4.2 Форма регистрации**

Форма регистрации, как можно увидеть, на рисунке 4.4 состоит из четырёх полей: имя пользователя, его email, пароль и подтверждение пароля. Все поля являются обязательными для ввода.

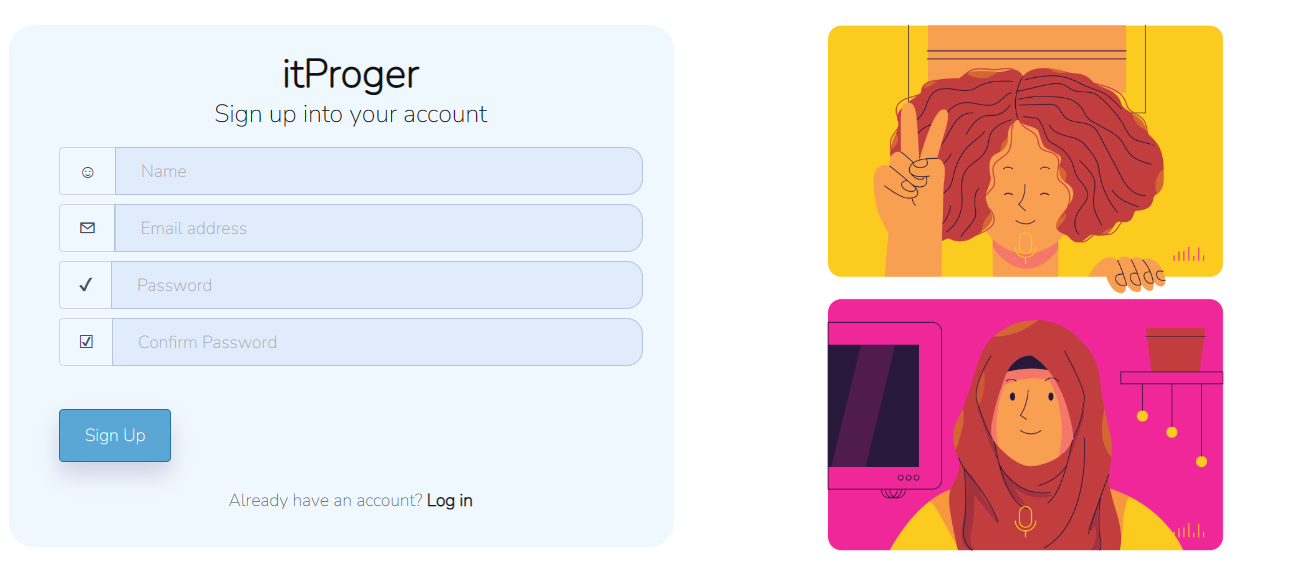


Рисунок 4.4 – Форма регистрации пользователя

При попытке отправить неверные данные, например, неправильный адрес электронной почты, пустое поле или несовпадающие пароли, будут отображены ошибки снизу данных полей. Такую ситуацию можно увидеть на рисунке 4.5.

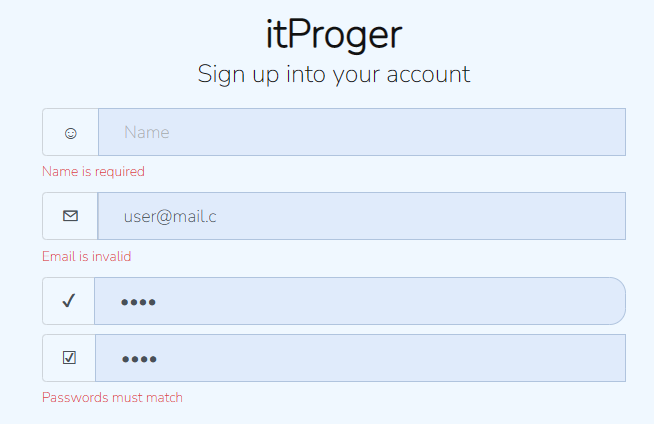


Рисунок 4.5 – Оповещение пользователя при вводе некорректных данных

Здесь же проверяется, чтобы адрес электронной почты был свободен и не повторялся с другим уже зарегистрированным пользователем, такой пример показан на рисунке 4.6.

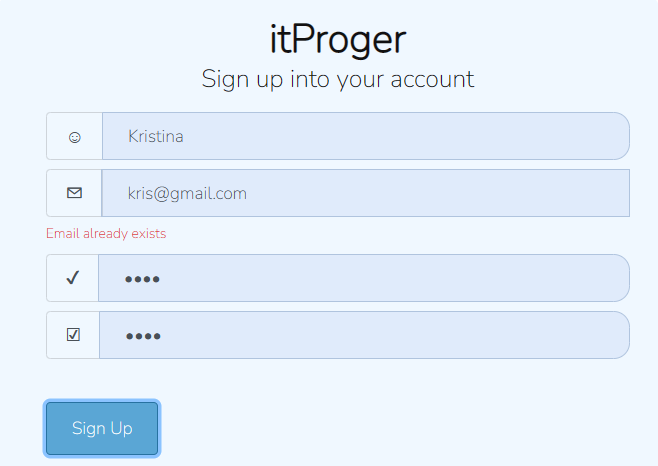


Рисунок 4.6 – Оповещение пользователя при вводе уже существующего адреса электронной почты

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

## **4.3 Форма создания профиля пользователя**

Пользователь имеет возможность создать свой профиль. Для этого необходимо ввести уникальный никнейм, навыки, которыми обладает пользователь, по желанию свой ник на GitHub’е и краткое описание. Форма для создания профиля представлена на рисунке 4.7.

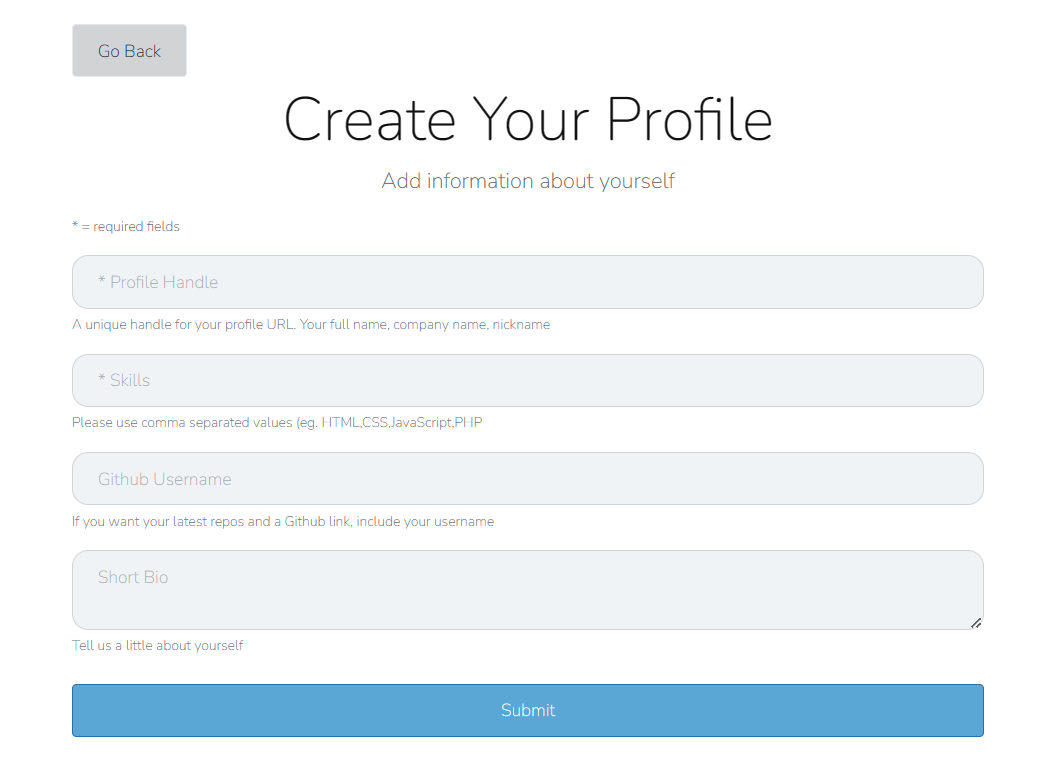


Рисунок 4.7 – Форма для создания профиля

В данной форме также присутствует валидация данных, которая требует заполнения необходимых полей. Случай представлен на рисунке 4.8.

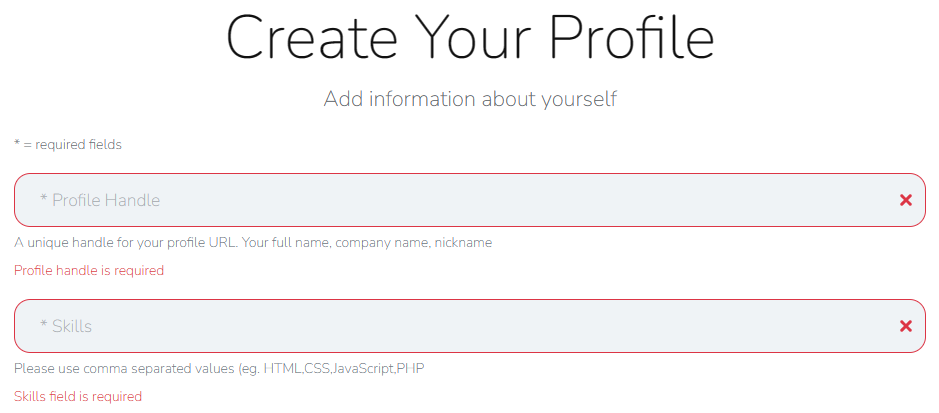


Рисунок 4.8 – Отображение ошибок, когда не будут заполнены необходимые поля

# **5 Руководство пользователя**

В данном разделе познакомимся с остальным функционалом, доступным пользователю.

## **5.1 Пользователь**

При первом переходе к приложению пользователь попадает на страницу входа. С нее он может перейти к форме регистрации, вид которой приведен на рисунке 5.1.

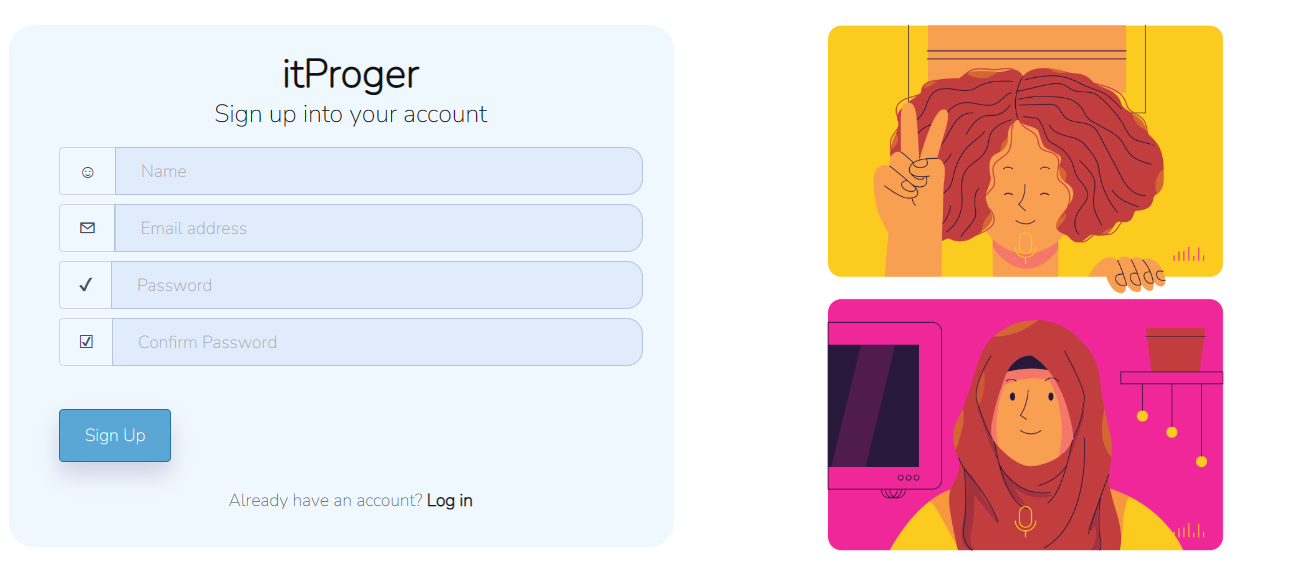


Рисунок 5.1 – Форма регистрации

После регистрации пользователь снова попадает на страницу входа. После того, как он вводит правильный email и пароль для входа, он попадает на приветственную страницу приложения, представленную на рисунке 5.2

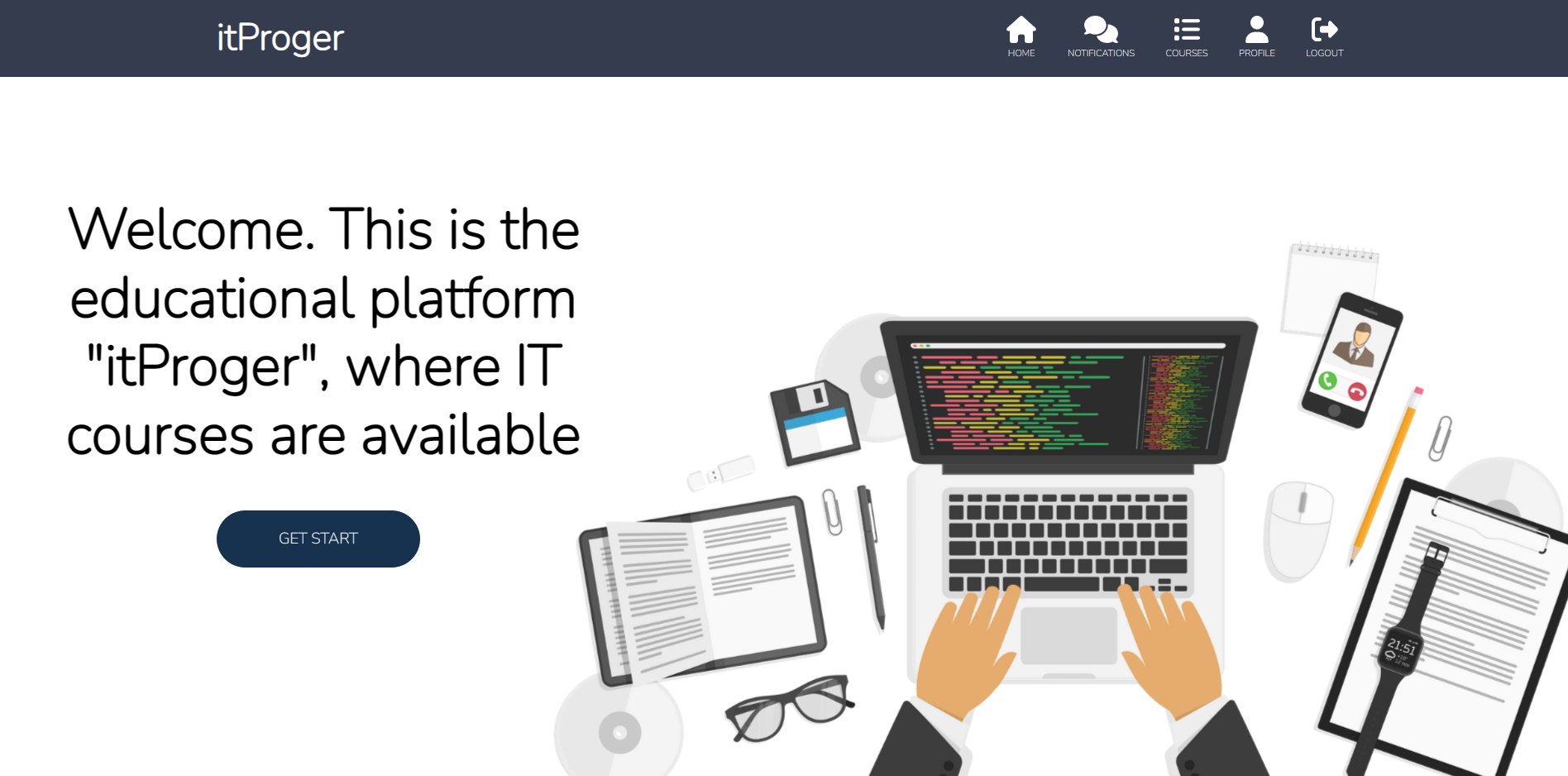


Рисунок 5.2 – Приветственная страница приложения

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

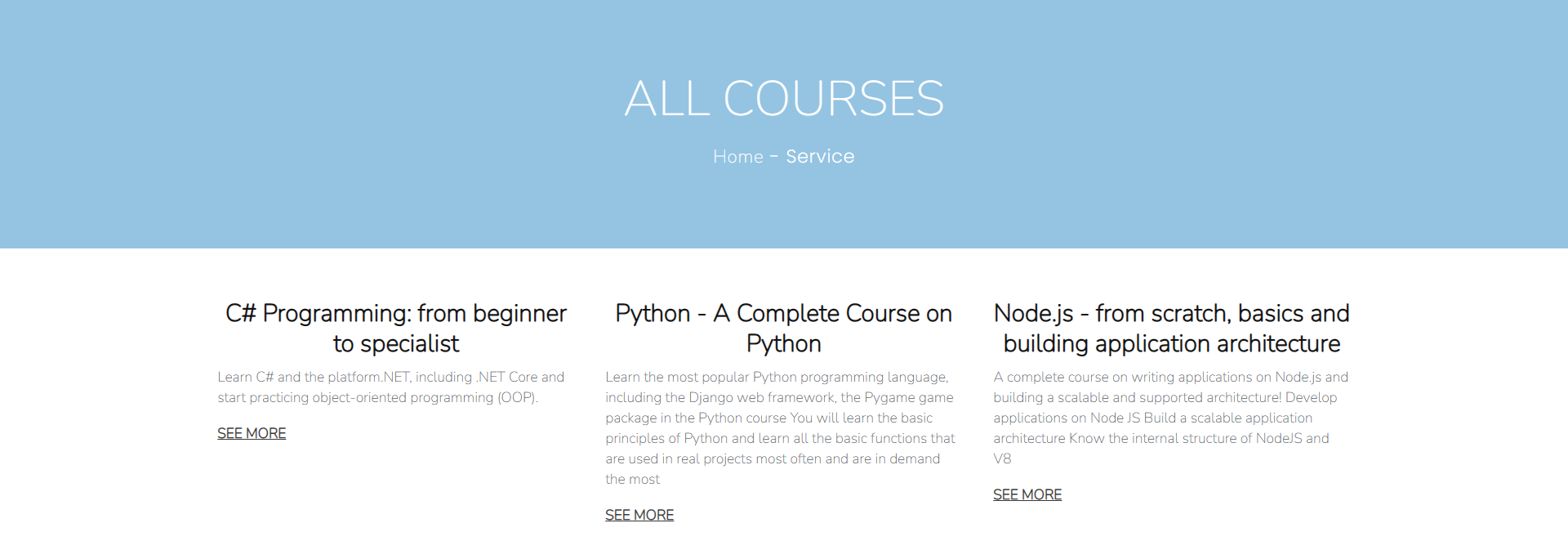


Рисунок 5.3 – Просмотр списка курсов

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).

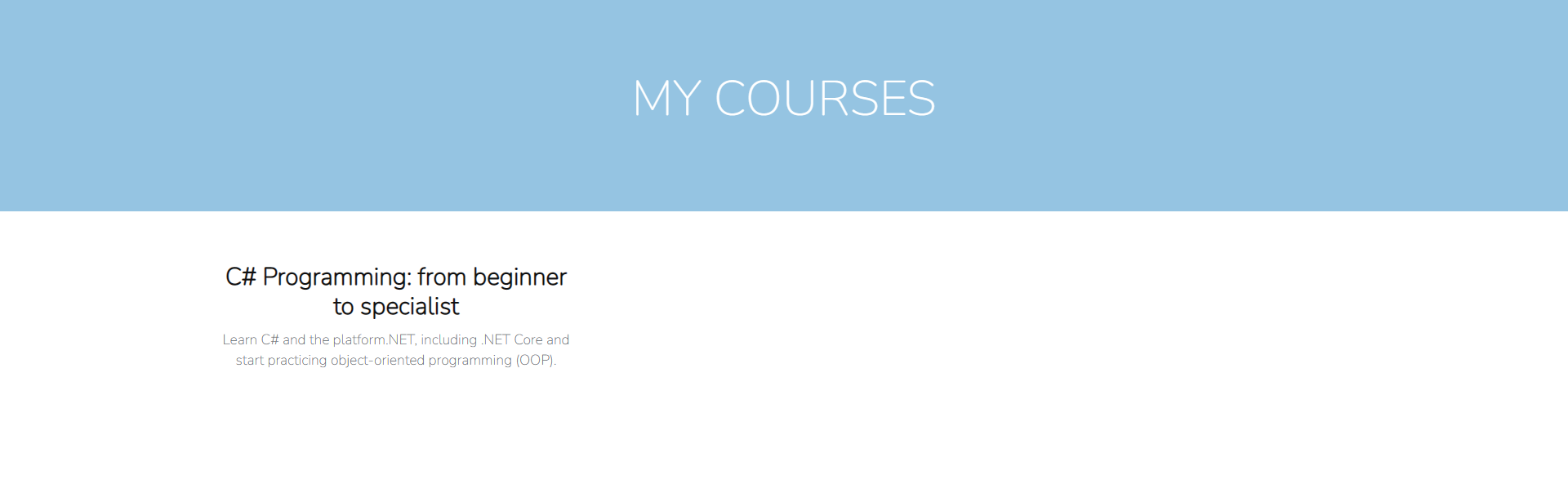


Рисунок 5.4 – Просмотр курсов, на которые записан студент

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.

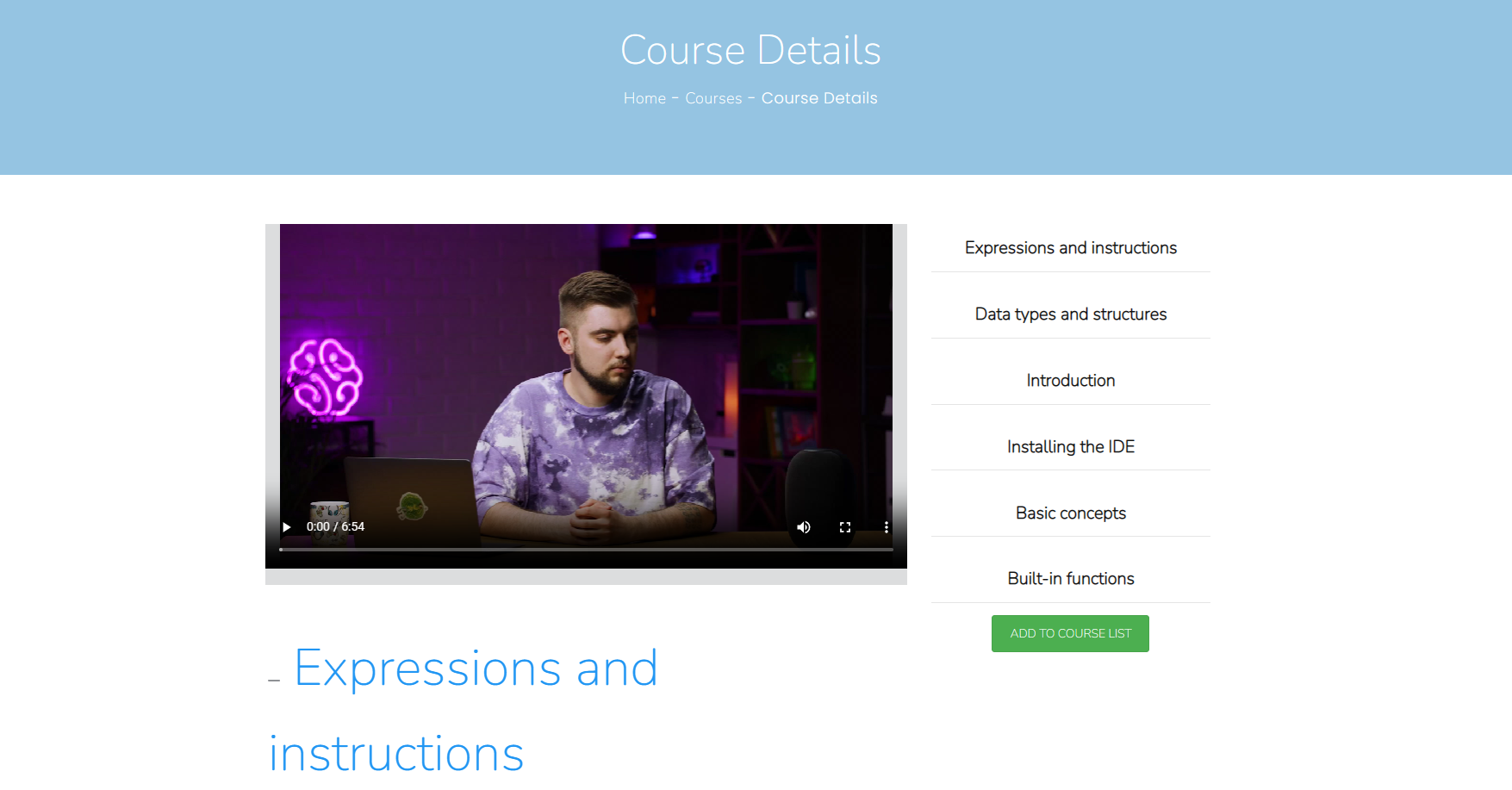


Рисунок 5.5 – Просмотр лекций

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

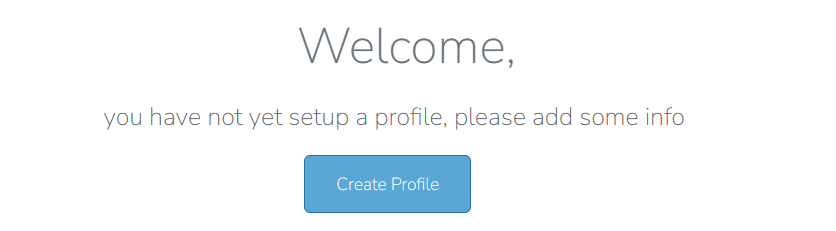


Рисунок 5.6 – Создание профиля

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.

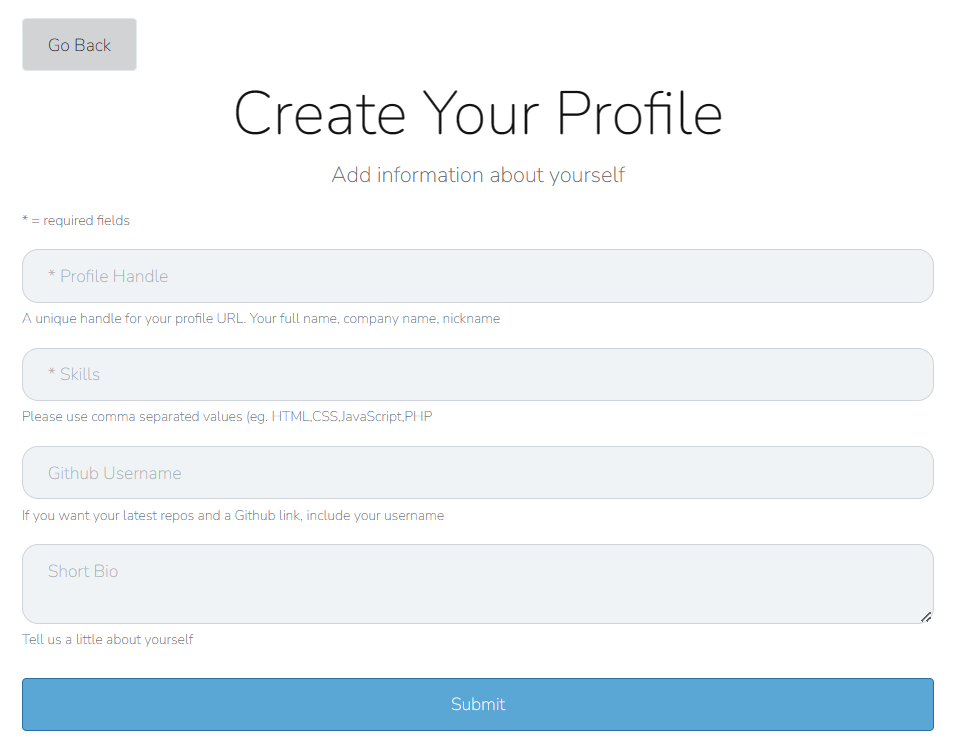


Рисунок 5.7 – Создание профиля

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

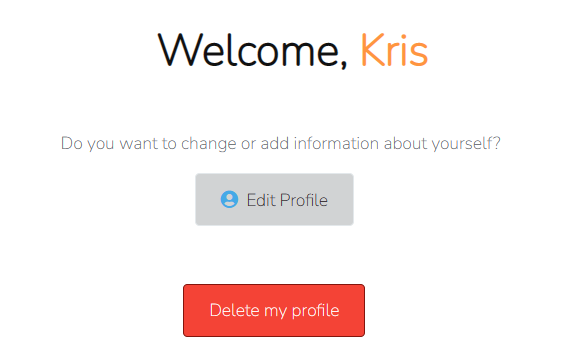


Рисунок 5.8 – Профиль пользователя

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

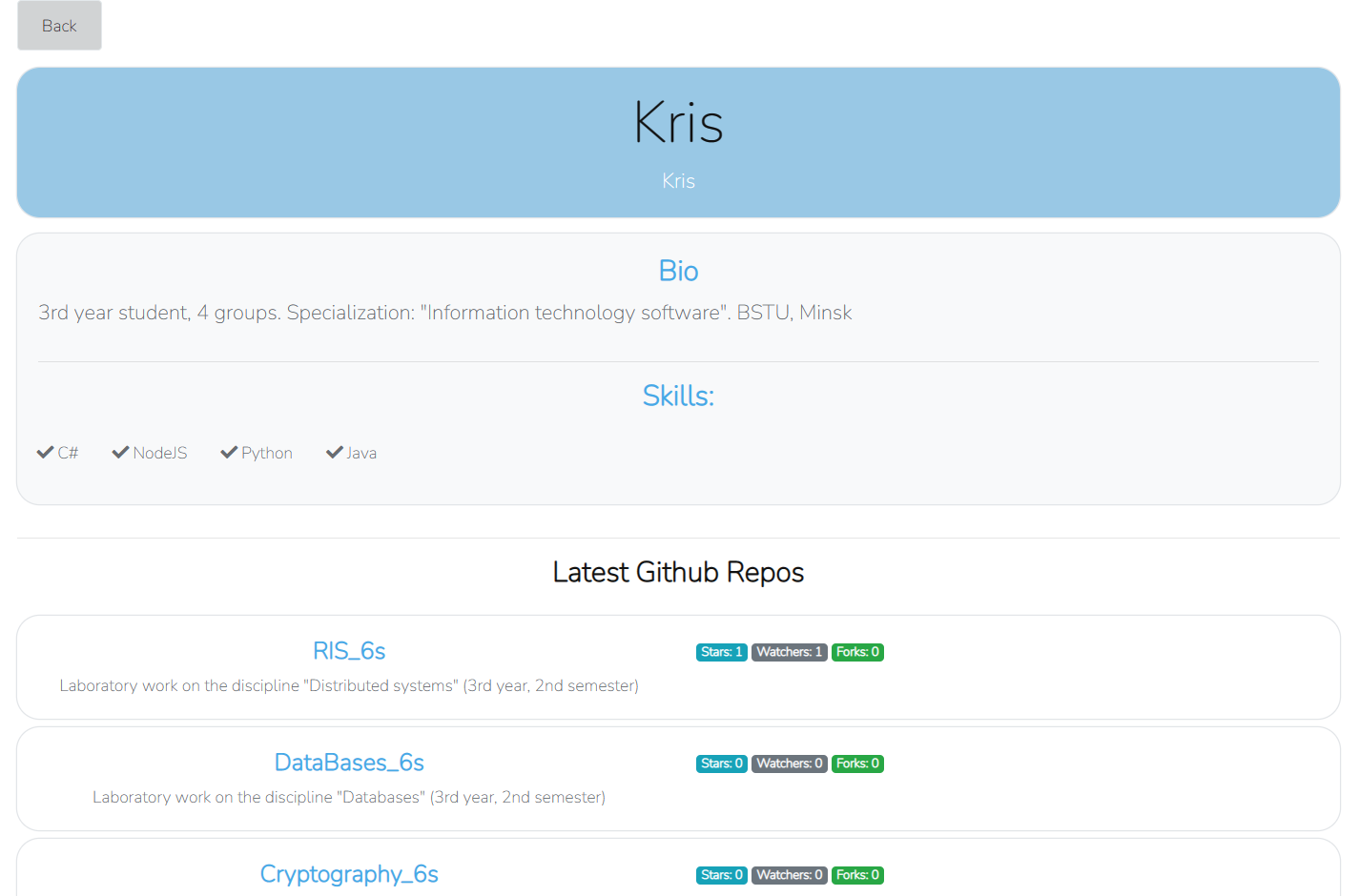


Рисунок 5.9 – Просмотр профиля

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.

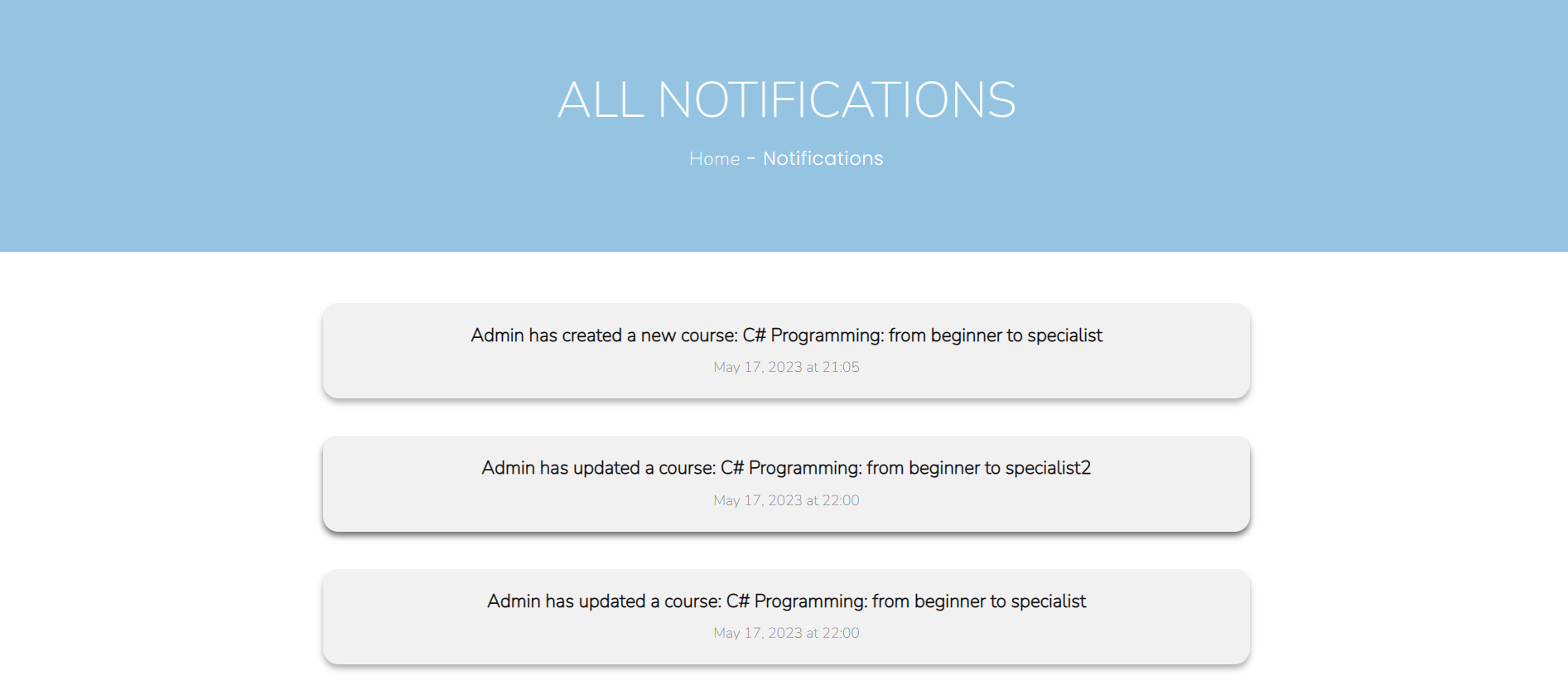


Рисунок 5.10 – Страница уведомлений пользователя

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.

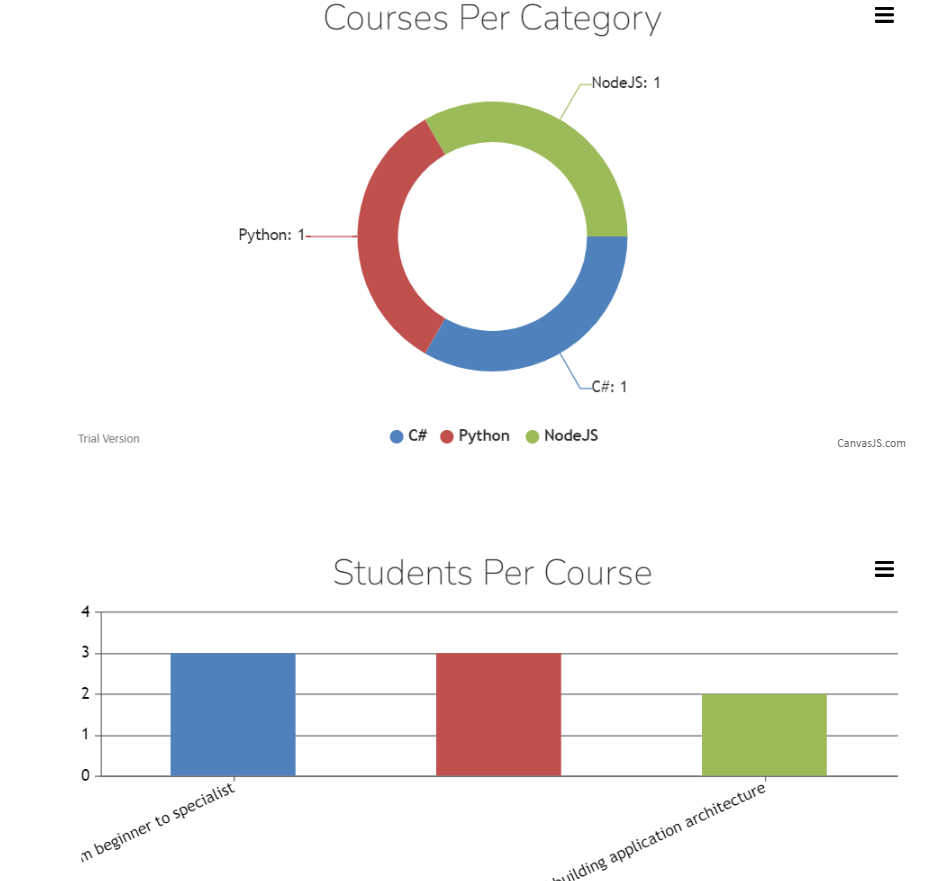


Рисунок 5.12 – Статистика в приложении

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.

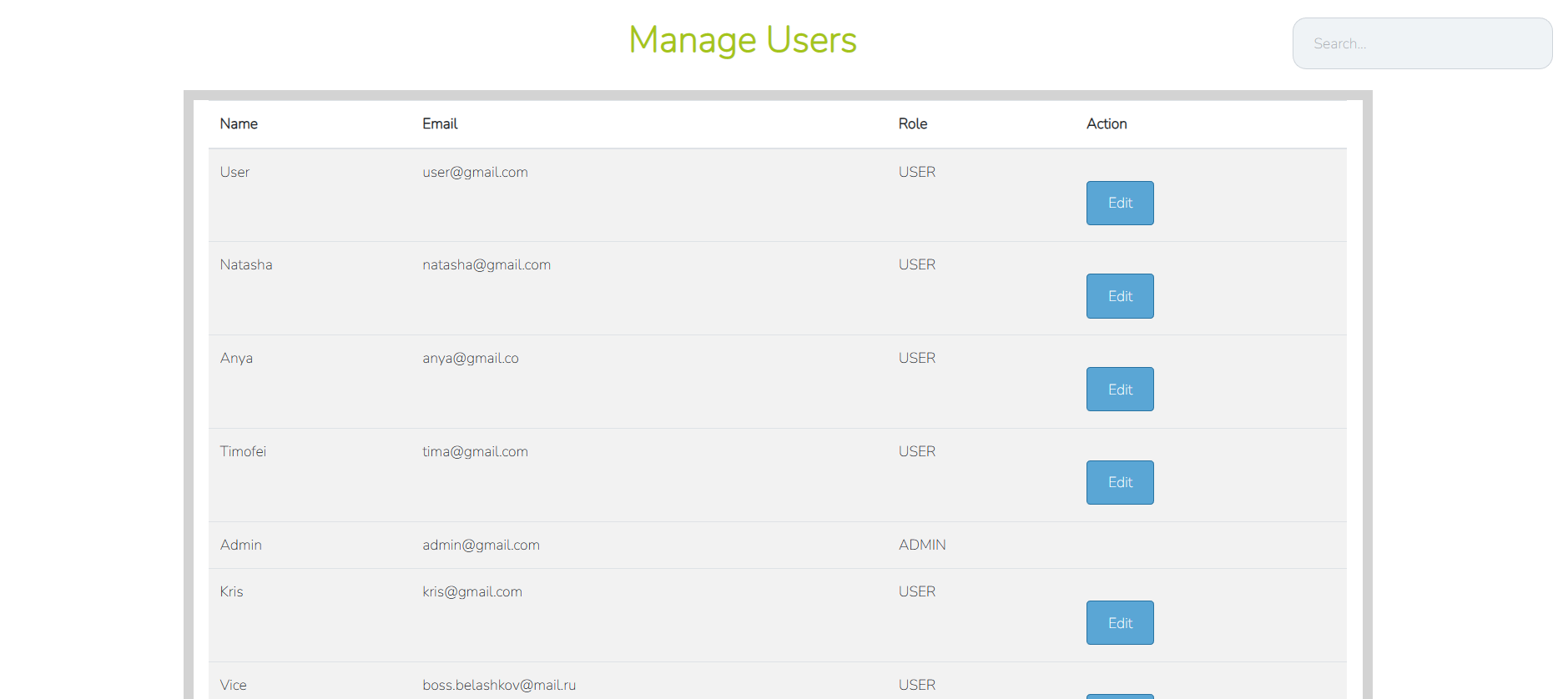


Рисунок 5.13 – Управление пользователями

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.

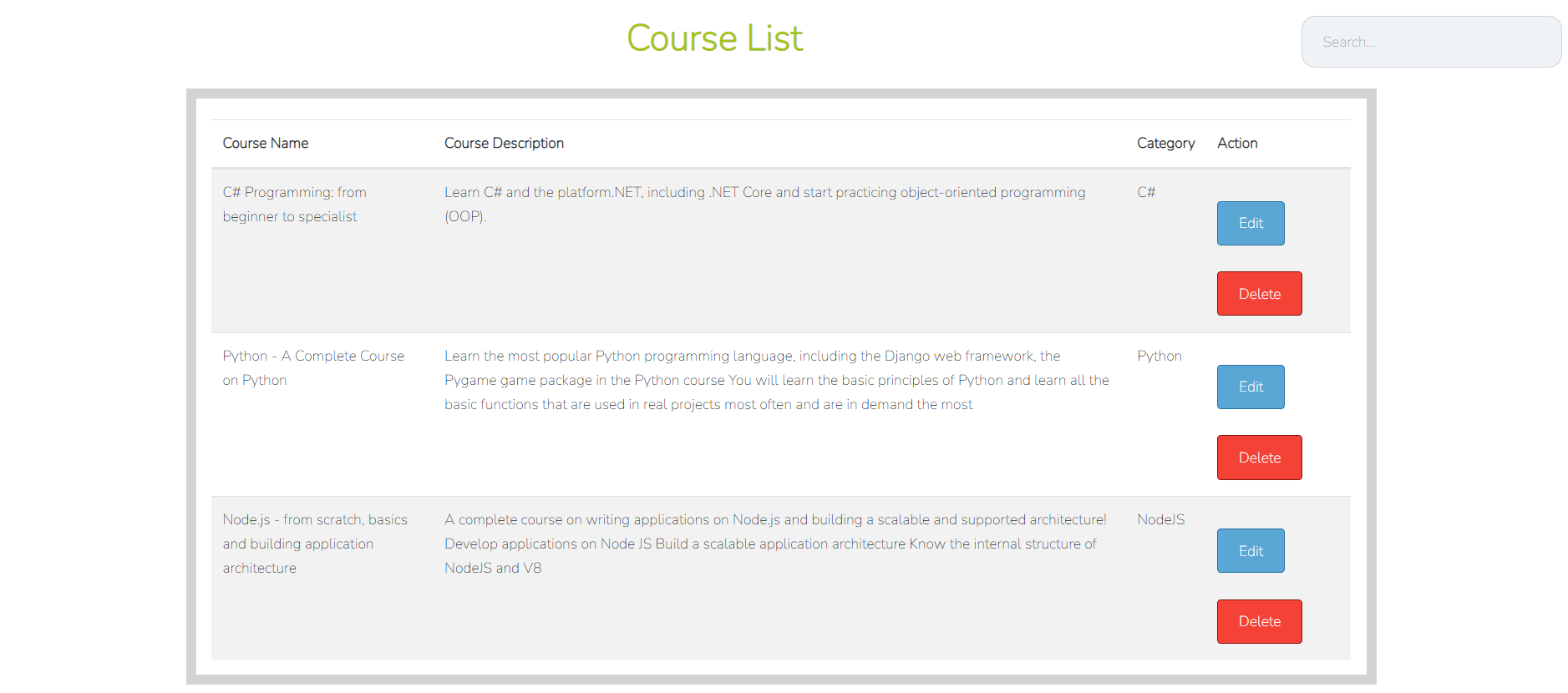


Рисунок 5.14 – Управление курсами

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».

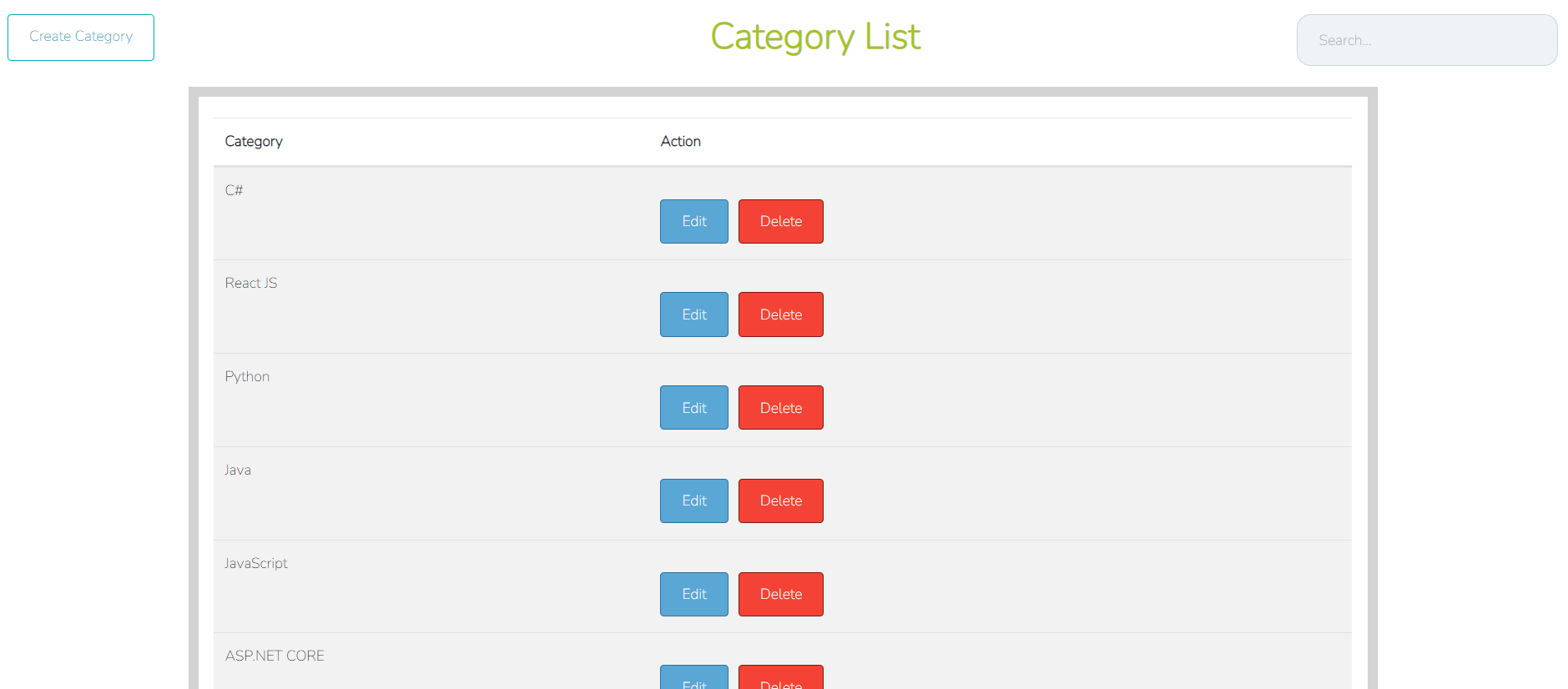


Рисунок 5.15 – Управление категориями

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

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

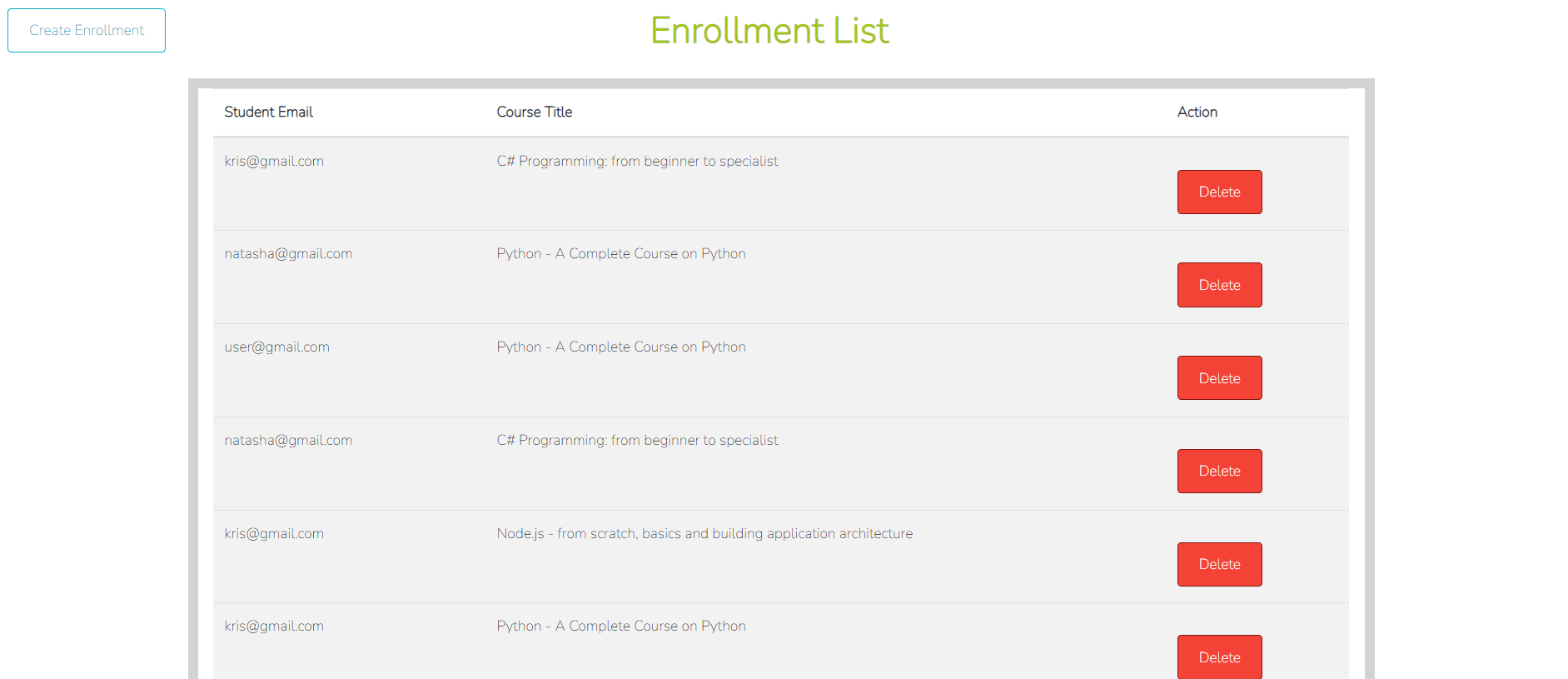


Рисунок 5.16 – Управление записями на курсы

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.

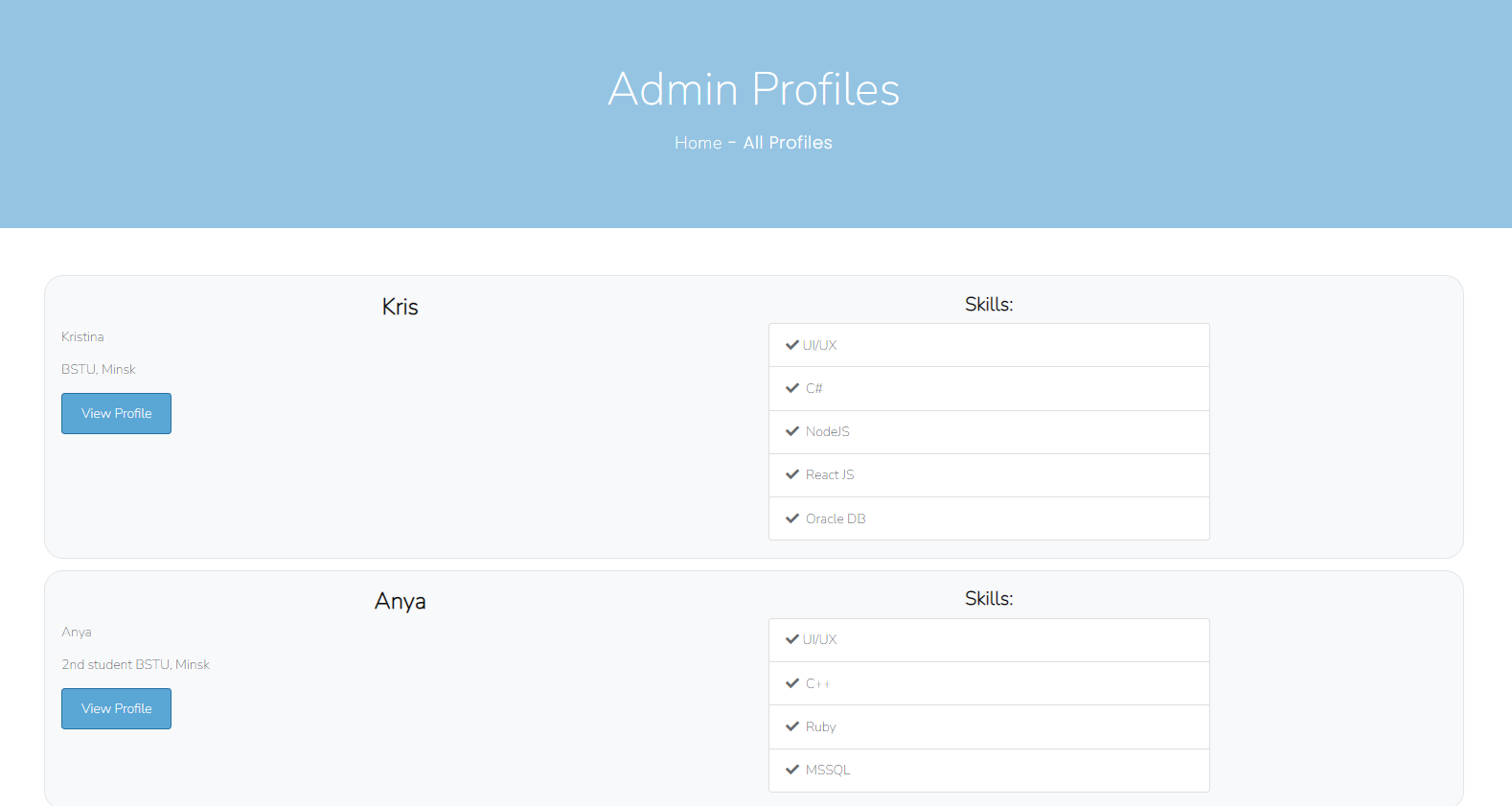


Рисунок 5.17 – Просмотр всех профилей

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.

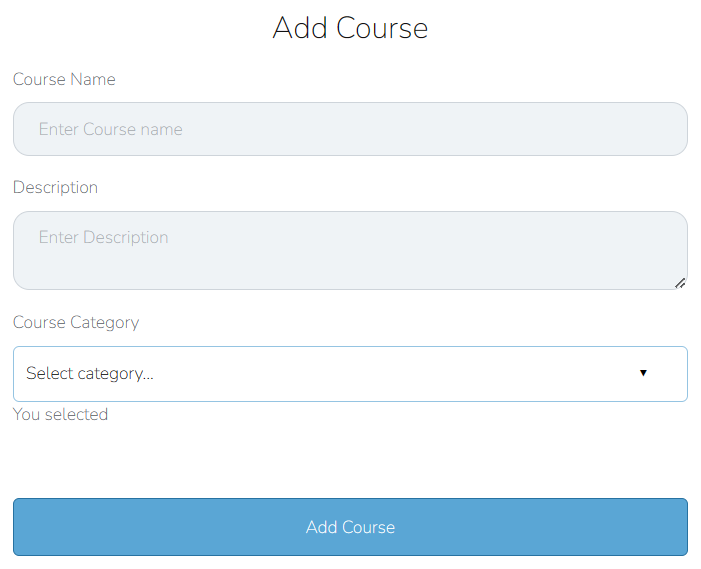


Рисунок 5.18 – Создание курса

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.

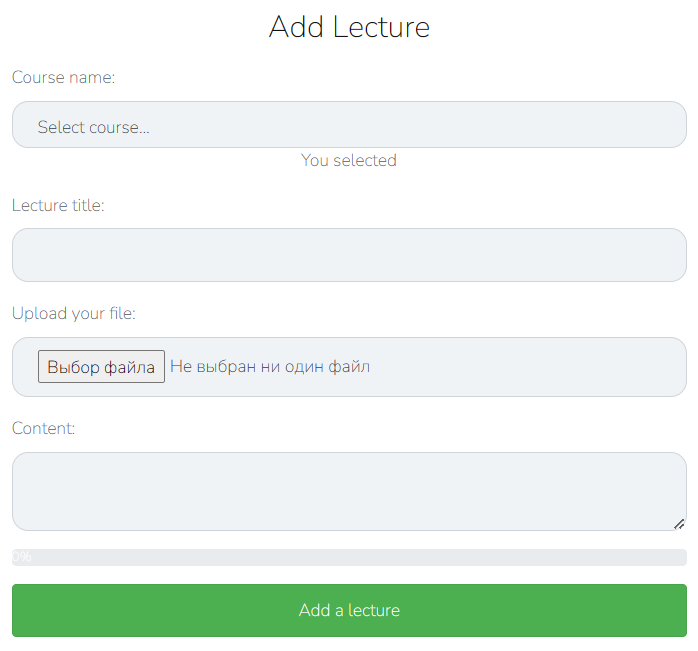


Рисунок 5.19 – Добавление лекции

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 |