МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждения 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 «CodeSchool»

Выполнил студент Агапкина Диана Сергеевна

(Ф.И.О.)

Руководитель проекта преп.-стаж. 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 преп.-стаж. Дубовик М.В.

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

Нормоконтролер преп.-стаж. Дубовик М.В.

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

Курсовой проект защищен с оценкой

Минск 2020

# **Реферат**

Пояснительная записка курсового проекта состоит из 43 страниц, 46 рисунков, 2 приложения, 4 источника литературы.

Основная цель курсового проекта: разработка платформы CodeSchool для изучения программирования.

В первом разделе рассматриваются прототипы приложения, литературные источники, а также формируются требования к проектируемому программному средству.

Во втором разделе описаны алгоритмы решения и разрабатываемая функциональность программного средства.

В третьем разделе представлена модель базы данных, обобщенная структура проекта и проектирование архитектуры проекта.

В четвертом разделе описана реализация программного средства.

В пятом разделе представлены результаты тестирования приложения.

Шестой раздел содержит руководство пользователя для разработанного клиентского приложения.

Седьмой раздел содержит руководство программиста.

В заключении описывается результат курсового проектирования и задачи, которые были решены в ходе разработки приложения.

**Оглавление**

[Реферат](#_cvtl5c7yjjvg) 2

[ВВЕДЕНИЕ](#_30j0zll) 5

[1 Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству](#_1fob9te) 6

[1.1 Обзор прототипов](#_3znysh7) 6

[1.1.1 SkillBox](#_h5hwyyuvghfi) 6

[1.1.2 GeekBrains](#_m7rzn6wb7bbq) 7

[1.2 Формирование требований](#_2et92p0) 8

[2 Анализ требований к проектируемому программному средству и разработка функциональных требований](#_tyjcwt) 9

[2.1 Алгоритмы решения](#_3dy6vkm) 9

[2.2 Описание разрабатываемой функциональности программного средства](#_1t3h5sf) 10

[3 Проектирование программного средства](#_4d34og8) 13

[3.1 Проектирование архитектуры](#_beil8j92tlsd) 13

[3.2 Обобщенная структура проекта](#_d4cgtj4rnp2w) 13

[3.3 Модель базы данных](#_17dp8vu) 14

[4 Создание (реализация) программного средства](#_3rdcrjn) 16

[4.1 Физическая структура программного средства](#_26in1rg) 16

[4.2 Разработка базы данных](#_dsfqpkx75i5r) 19

[4.3 Авторизация](#_bq315s8vvfcd) 23

[5 Тестирование](#_1ksv4uv) 25

[6 Методика использования программного средства](#_15aulpijgo1u) 28

[6.1 Преподаватель](#_lcttb73gntgh) 28

[6.2 Студент](#_b0brs95tzsxd) 34

[6.3 Администратор](#_wrp6j9h7ss3l) 35

[7 Руководство программиста](#_44sinio) 38

[ЗАКЛЮЧЕНИЕ](#_2jxsxqh) 40

[СПИСОК ЛИТЕРАТУРЫ](#_z337ya) 42

[ПРИЛОЖЕНИЕ А](#_1ci93xb) 43

[ПРИЛОЖЕНИЕ Б](#_3whwml4) 44

# **ВВЕДЕНИЕ**

Электронное обучение или e-learning развивается, практикуется и совершенствуется уже много лет. Тема актуальна как в СМИ, так и среди научного и образовательного сообществ. С начала 2020 года дистанционная учёба стала еще более актуальной.

Дистанционное образование и до карантина с самоизоляцией было очень популярным по многим причинам. На сегодняшний день неизвестно, сколько еще времени продлится пандемия, а учиться надо всегда. Поэтому дистанционный формат будет актуален еще долгое время.

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 — никто не заставляет вас учиться 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 онлайн-платформы для обучения программированию CodeSchool.

# **1 Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству**

## **1.1 Обзор прототипов**

### 1.1.1 SkillBox

Skillbox – онлайн-университет, в котором обучают 90+ программам для получения востребованных профессий во всем земном шаре. И всё это в режиме online!

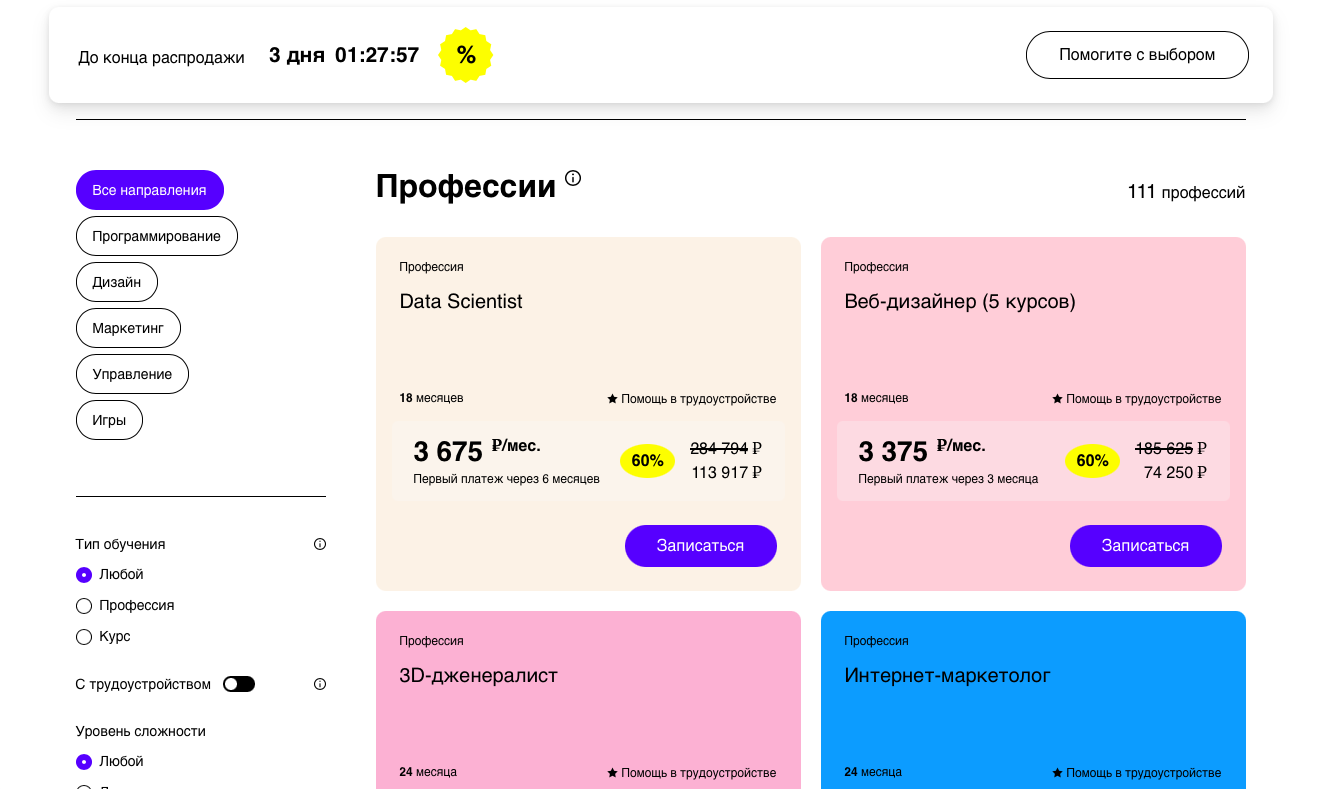
****

Рисунок 1.1.1 – Онлайн-университет Skillbox

Здесь обучают профессиям, которые будут всегда востребованы в мире IT. Если осваивать программы больше года, можно не только получить диплом, но и наработать достойное портфолио, составить CV и устроиться на работу ещё ДО завершения курсов.

В Skillbox разноформатное обучение. Самым основным являются видеолекции, после которых обязательным порядком дают практические домашние задания. Проверяют д/з и консультируют по ним педагоги экспертного уровня в мессенджерах. Но помимо этого есть онлайн-семинары и «живые» встречи офлайн-формата.

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.

Если нет возможности уделять время обучению, а основы хочется знать прямо здесь и сейчас, тогда имеет смысл записаться на курсы, которые по времени занимают меньше года.

### 1.1.2 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.

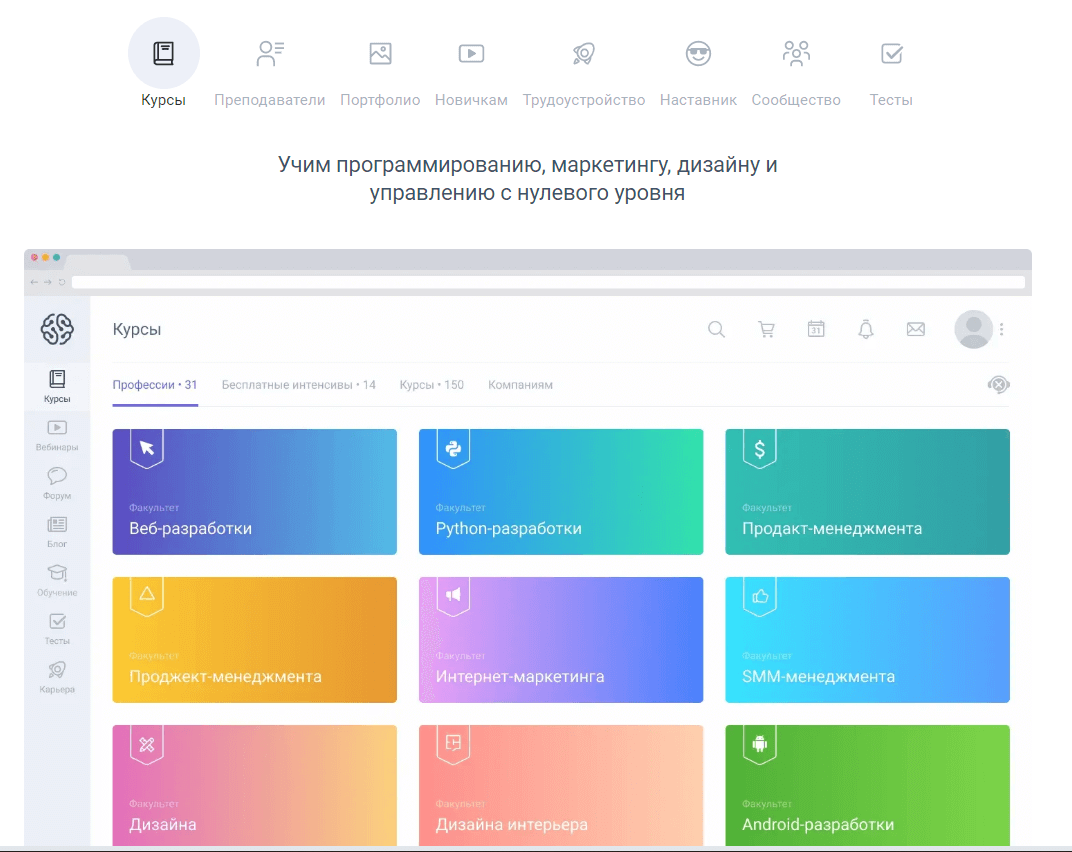
****

Рисунок 1.1.2 – Онлайн-университет GeekBrains

Преимущество этого образовательного портала в доступе к 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.

## **1.2 Формирование требований**

Исходя из проведенного анализа и с учетом требований, указанных в задании на курсовое проектирование, основной задачей является проектирование и разработка веб-приложения онлайн-платформы для обучения программированию CodeSchool, которое ориентировано абсолютно для любого человека, который хочет делиться своими знаниями с другими или обучаться чему-то новому у других. Благодаря возможности просмотра абсолютно всех курсов, которые добавили преподаватели, пользователь может выбрать именно то, что ему нравится. Также вы имеете возможность побывать в роли преподавателя и самому добавлять курсы и лекции к ним!

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

– Сохранять пользовательскую информацию в базе данных;

– Получать информацию из базы данных;

– Создание пользовательского интерфейса для взаимодействия с базой данных;

– Регистрация и авторизация пользователей;

– Удаление при необходимости информации из базы данных;

– Просмотр уроков / лекций;

– Добавление материалов в урок;

– Добавление уроков.

# **2 Анализ требований к проектируемому программному средству и разработка функциональных требований**

## **2.1 Алгоритмы решения**

Серверная часть приложения реализована с помощью Node.js. Node.js представляет собой среду выполнения JavaScript, построенную на базе JS-движка V8, разработанного Google и применяемого в Google Chrome. 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 опенсорсный 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.

Node.js — это низкоуровневая платформа. Для того чтобы упростить разработку для неё и облегчить жизнь программистам, было создано огромное количество библиотек. Некоторые из них со временем стали весьма популярными. В проекте использовалась библиотека Express, которая предоставляет предельно простой, но мощный инструмент для создания веб-серверов. Ключом к успеху Express стал минималистический подход и ориентация на базовые серверные механизмы без попытки навязать некое видение «единственно правильной» серверной архитектуры.

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.

– Использует виртуальный DOM, который является объектом JavaScript. Это улучшает производительность приложений, поскольку виртуальный JavaScript DOM быстрее, чем обычный DOM.

– Может использоваться как на стороне клиента, так и на стороне сервера, а также на других платформах.

– Компоненты и модели данных улучшают читаемость, что помогает поддерживать более крупные приложения.

В качестве СУБД использовалась MongoDB. MongoDB реализует новый подход к построению баз данных, где нет таблиц, схем, запросов SQL, внешних ключей и многих других вещей, которые присущи объектно-реляционным базам данных. В отличие от реляционных баз данных MongoDB предлагает документо-ориентированную модель данных, благодаря чему MongoDB работает быстрее, обладает лучшей масштабируемостью, ее легче использовать.

Для хранения в MongoDB применяется формат, который называется BSON (БиСон) или сокращение от binary JSON. BSON позволяет работать с данными быстрее: быстрее выполняется поиск и обработка.

Отсутствие жесткой схемы базы данных и в связи с этим потребности при малейшем изменении концепции хранения данных пересоздавать эту схему значительно облегчают работу с базами данных MongoDB и дальнейшим их масштабированием. Кроме того, экономится время разработчиков. Им больше не надо думать о пересоздании базы данных и тратить время на построение сложных запросов.

## **2.2 Описание разрабатываемой функциональности программного средства**

Программное средство предусматривает 4 группы пользователей. Рассмотрим функциональные возможности для каждой из этих групп.

Незарегистрированный пользователь:

* Зарегистрироваться как студент;
* Зарегистрироваться как преподаватель.

Преподаватель:

* Добавить курс;
* Добавить лекцию;
* Просмотр своих курсов;
* Просмотр всех доступных курсов;
* Просмотр лекций;
* Заполнить профиль;
* Просмотр своего профиля;
* Просмотр профилей всех преподавателей;
* Выйти из системы.

Студент:

* Просмотр всех доступных курсов;
* Добавить курс;
* Просмотр добавленных курсов;
* Просмотр лекций из курса;
* Выйти из системы.

Администратор:

* Просмотр статистики;
* Просмотр списка пользователей;
* Редактирование пользователей;
* Просмотр списка всех курсов;
* Редактирование курса;
* Просмотр категорий курсов;
* Создание категории;
* Редактирование категории;
* Просмотр студентов, записанных на курсы;
* Записать студента на курс;
* Удалить студента с курса;
* Выйти из системы.

На рисунке 2.2.1 представлена UML-диаграмма вариантов использования, которая отражает функциональность программного средства с точки зрения получения значимого результата для различных пользователей.

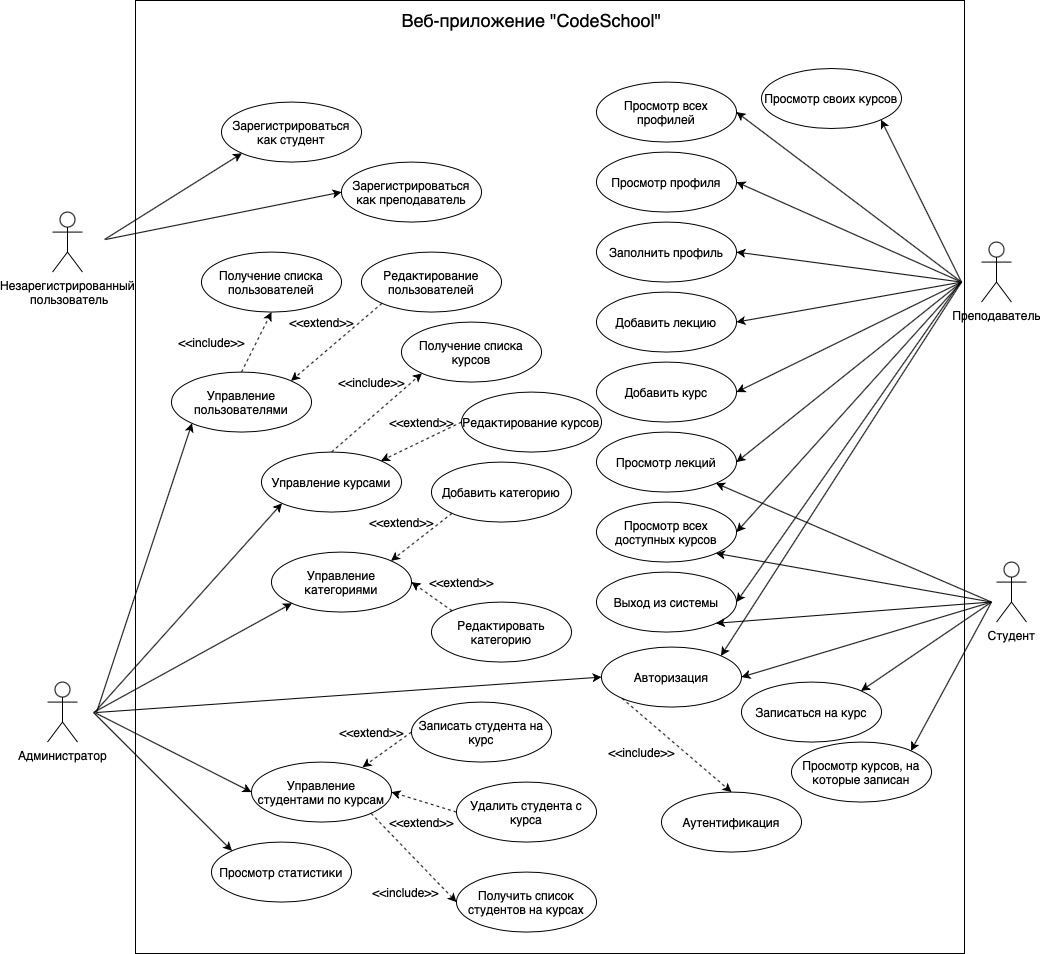


Рисунок 2.2.1 – Диаграмма вариантов использования

Таким образом, в ходе работы над этим разделом были сформулированы основные функциональные требования для проектирования программного средства.

# **3 Проектирование программного средства**

## **3.1 Проектирование архитектуры**

Архитектура проекта – это его строение как оно видно (или должно быть видно) извне его, т.е. представление программного средства как системы, состоящей из некоторой совокупности взаимодействующих подсистем. В качестве таких подсистем выступают обычно отдельные программы. Разработка архитектуры является первым этапом борьбы со сложностью программного средства, на котором реализуется принцип выделения относительно независимых компонент.

Основные задачи разработки архитектуры проекта:

* выделение программных подсистем и отображение на них внешних функций (заданных по внешнему описанию) программного средства;
* определение способов взаимодействия между выделенными программными подсистемами.

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

## **3.2 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 клиент-серверной архитектуры представлена на рисунке 3.2.1.

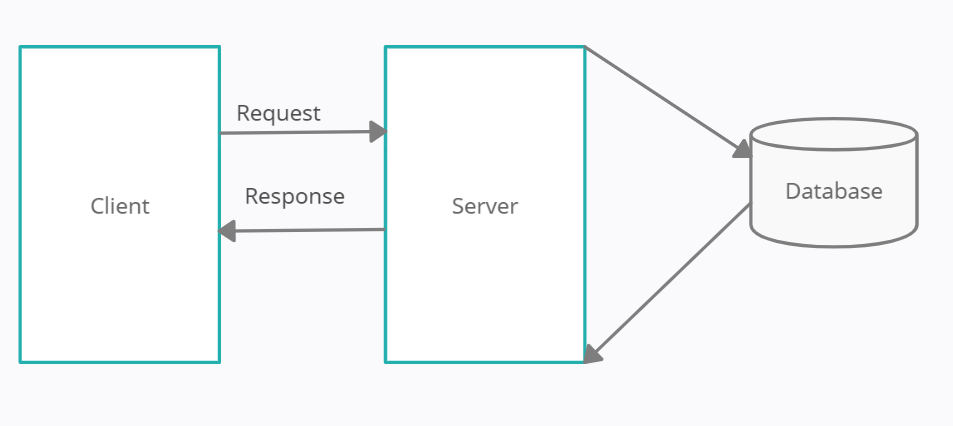


Рисунок 3.2.1 – Клиент-серверная архитектура

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.

Однако не стоит забывать, что с возрастанием числа клиентов, возможно, понадобится увеличение вычислительной мощи серверной части приложения. При разработке данного приложения использована двухуровневая архитектура. В трехуровневой архитектуре добавляется сервер базы данных, на котором работает какая-либо база данных. В данном случае MongoDB – удалённая база данных. На рисунке 3.2.2 изображена трехуровневая архитектура.

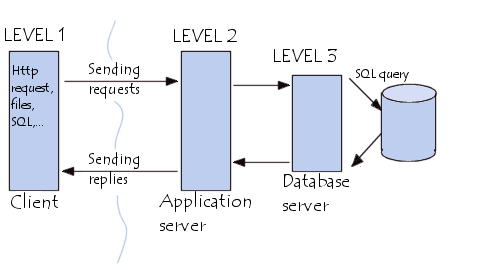


Рисунок 3.2.2 – Трехуровневая архитектура

Таким образом, приложение должно быть выполнено в клиент-серверной архитектуре. Сервер принимает запрос клиента, обрабатывает запрос, работает с базой данных и возвращает клиенту ответ. Клиент будет являться инициатором соединения с сервером.

## **3.3 Модель базы данных**

Для реализации поставленной в курсовом проектировании задачи была создана база данных CODESCHOOL. Для её создания использовалась система управления реляционными базами данных MongoDB. Взаимосвязь всех таблиц проектируемой базы данных представлена на рисунке 3.3.1.

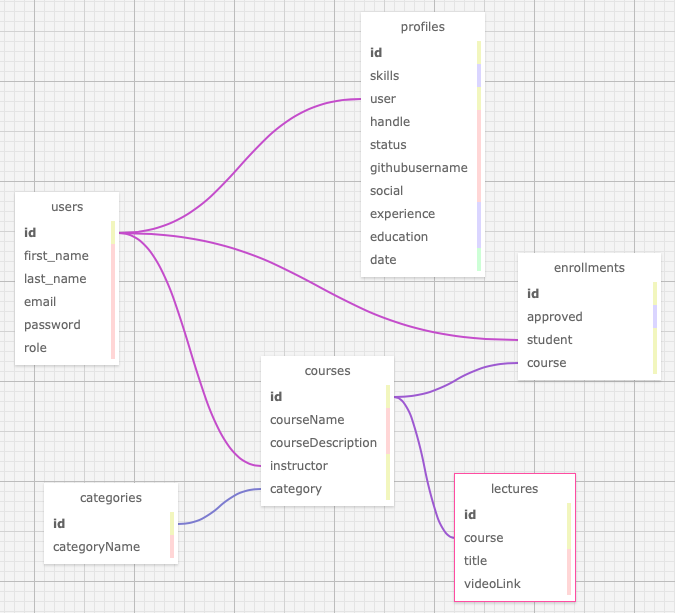


Рисунок 3.3.1 – Взаимосвязь таблиц базы данных

Созданная база данных содержит в себе 6 документов:

* Users – для хранения информации о пользователе;
* Profiles – данные для профиля преподавателя;
* Categories – содержатся названия категорий для курсов;
* Courses – информация о курсе;
* Lectures – данные из лекции;
* Enrollments – информация о студентах, записанных на определенные курсы;

Более подробная структура будет рассмотрена в следующей главе.

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

## **4.1 Физическая структура программного средства**

Решение представлено двумя проектами: server (серверная часть на Node.js) и client (клиентская часть – ReactJS). Для старта сначала запускается сервер, затем – клиент.

Структура проекта backend представлена на рисунке 3.1.1.

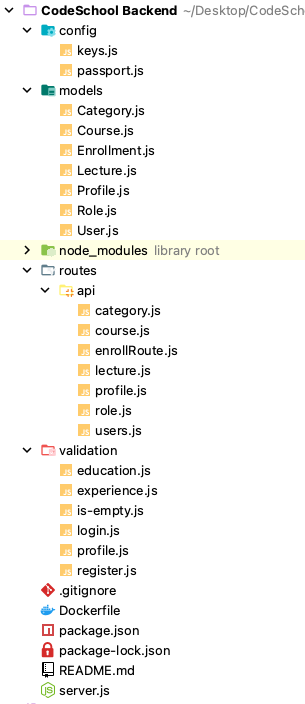


Рисунок 3.1.1 – Структура проекта server

Описание приведенной структуры проекта представлено в таблице 3.1.1.

Таблица 3.1.1 – Описание структуры проекта

|  |  |
| --- | --- |
| Имя файла | Содержание |
| Config | Конфигурационные данные, такие как строка подключения к базе данных и секрет JWT. |
| Model | Подпространство имён, содержащее описание сущностей для работы с БД. |
| Validation | Различные промежуточные проверки на корректный ввод данных. Например, чтобы поля были непустыми, проверка на количество символов в пароле и т.д. |
| Routes | Определение маршрутов и их обработчиков. Также там содержится и логика по обработке запроса по каждому из маршрутов. |
| Util | Дополнительная логика. |
| server.js | Основной файл приложений – главный, он же и запускается. |
| Node\_modules | Все скачанные модули проекта |

На рисунке 3.1.2 показана структура файлов и папок в корне проекта client.

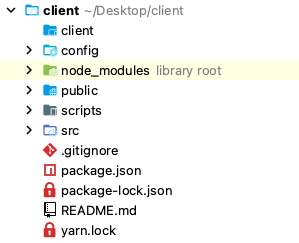


Рисунок 3.1.1 – Структура корня проекта client

Описание приведенной структуры представлено в таблице 3.1.2.

Таблица 3.1.2 – Описание файлов и папок в корне проекта client.

|  |  |
| --- | --- |
| Имя файла | Содержание |
| node\_modules | В этой папке хранятся все пакеты, которые используются для создания приложения. |
| src | В этой папке хранится весь исходный код нашего приложения. |
| сonfig | Конфигурационные данные. |
| public | Общедоступные файлы, такие как скрипты, стили, изображения, шрифты и т.д. |
| scripts | Скрипты для сборки и запуска приложения. |
| package.json | Данный файл устанавливает пакеты и зависимости, которые будут использоваться проектом. |

Структура директории src того же проекта представлена на рисунке 3.1.3.

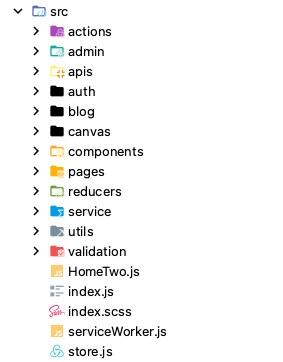


Рисунок 3.1.3 – Структура директории src проекта client

Описание приведенной структуры представлено в таблице 3.1.3.

Таблица 3.1.3 – Описание файлов и папок в корне проекта frontend.

|  |  |
| --- | --- |
| Имя файла | Содержание |
| actions | Запросы на сервер, связанные с авторизацией и профилем. |
| admin | Это основная html-страница для нашего проекта. Тут можно добавить/изменить мета-теги title, description, изменить кодировку документа, viewport и т.д. |
| apis | Параметры для обращения к YouTube API. |
| auth | Компоненты для логина и регистрации. |
| blog | Компоненты для управления курсами и лециями. |
| canvas | Подключение библиотеки для работы с диаграммами, отображающими статистику для администратора. |
| components | Компоненты для работы с профилем преподавателя: заполнение, редактирование, отображение, просмотр всех профилей. |
| pages | Страница для вывода ошибки. |
| reducers | Редьюсеры, т.е функция, которая принимает предыдущее состояние и экшен (state и action) и возвращает следующее состояние (новую версию предыдущего). |
| service | Компоненты для просмотр своих курсов для студентов и преподавателей, а также просмотра всех доступных курсов. |
| utils | Дополнительная функция для добавления токена к заголовку. |
| validation | Проверка на непустые данные. |
| HomeTwo,js | Компонент стартовой страницы. |
| index.js | Сопоставление маршрутов и компонентов. |
| index.scss | Объединение всех импортированных scss и sass файлов. |
| serviceWorker.js | Посредник между клиентом и сервером, пропускающий через себя все запросы к серверу. С его помощью можно перехватывать все запросы “на лету”. |
| store.js | Глобальное хранилище приложения. |

Следует отметить, что весь пользовательский интерфейс строится из компонентов. React разработан вокруг концепции многоразовых компонентов. Мы определяем небольшие компоненты, и объединяем их, чтобы сформировать более крупные компоненты. Все компоненты, маленькие или большие, могут использоваться повторно, даже в разных проектах.

Каждый компонент — это JavaScript-функция, которая возвращает кусок кода, представляющего фрагмент страницы. Для формирования страницы мы вызываем эти функции в определённом порядке, собираем вместе результаты вызовов и показываем их пользователю.

Таким образом, сформированные таблицы помогают понять общую структура проектов проектируемого программного средства.

## **4.2 Разработка базы данных**

Для работы с базой данных использовалась специальная ODM-библиотека (Object Data Modelling) Mongoose, которая позволяет сопоставлять объекты классов и документы коллекций из базы данных. Грубо говоря, Mongoose работает подобно инструментам ORM.

Данные, которые используются в Mongoose, описываются определенной схемой. Схема содержит метаданные объектов. В частности, здесь устанавливаем, какие свойства будет иметь объект и какой у них будет тип данных. То есть это схема, которая описывает объект пользователя.

Созданная база данных содержит в себе 6 документов: users, profiles, categories, courses, lectures, enrollments. Рассмотрим их структуру.

Схема модели users представлена на рисунке 3.2.2.



Рисунок 3.2.2 – Модель user

Документ users содержит информацию о пользователях:

* \_id – генерируется автоматически, однозначно идентифицирует объект;
* firstname – имя пользователя;
* lastname – фамилия пользователя;
* email – электронный адрес пользователя / логин;
* password – пароль;
* \_v – добавляется автоматически, указывает на версию документа;

Схема модели profile представлена на рисунке 3.2.3.

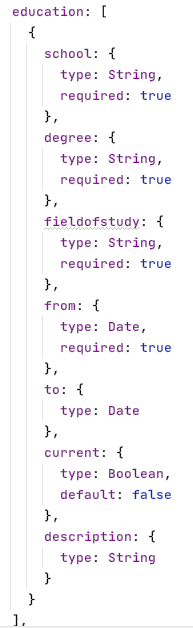
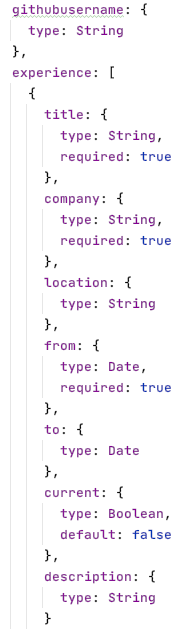
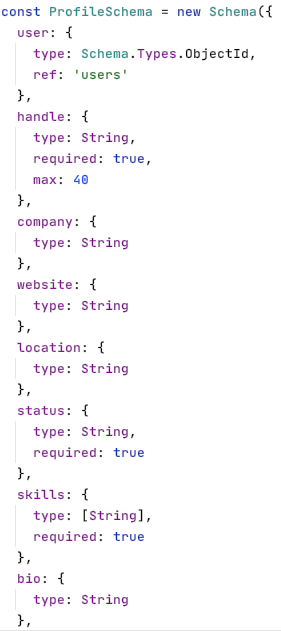


Рисунок 3.2.3 – Модель profile

Документ posts содержит информацию о всех опубликованных постах:

* \_id – генерируется автоматически, однозначно идентифицирует объект;
* user – идентификатор пользователя, заполнившего профиль;
* handle – имя пользователя;
* company – название компании;
* website – веб-сайт;
* locations – месторасположение;
* status – название позиции;
* skills – навыки;
* bio – биография или сведения о пользователе;
* githubusername – имя пользователя в GitHub;
* experience – вложенный объект:
  + \_id – генерируется автоматически, однозначно идентифицирует объект;
  + title – название позиции;
  + company – название компании;
  + from – с какого периода;
  + to – по какой период;
  + current – является ли текущей на данный момент;
  + description – описание работы;
* education – вложенный объект:
  + \_id – генерируется автоматически, однозначно идентифицирует объект;
  + school – название учебного заведения;
  + degree – ученая степень;
  + fieldofstudy – область обучения;
  + from – с какого периода;
  + to – по какой период;
  + current – является ли текущей на данный момент;
  + description – описание;
* social – вложенный объект, включающий ссылки на социальные сети;
* date – дата регистрации;
* createdAt – создается автоматически, т.к при создании схемы указано {timestamps:true}, время создания объекта;
* updatedAt – создается автоматически, т.к при создании схемы указано {timestamps:true}, время изменения объекта;
* \_v – добавляется автоматически, указывает на версию документа.

Схема модели categories представлена на рисунке 3.2.4.

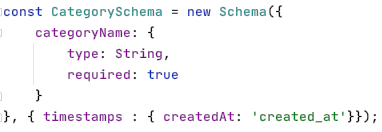


Рисунок 3.2.4 – Модель categories

Документ categories содержит информацию о категориях курсов:

* \_id – генерируется автоматически, однозначно идентифицирует объект;
* categoryName – название категории;

Схема модели courses представлена на рисунке 3.2.5.

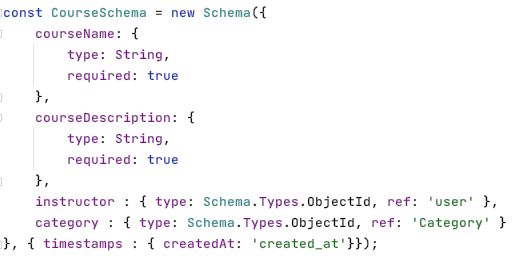


Рисунок 3.2.5 – Модель courses

Документ courses содержит информацию о курсах:

* \_id – генерируется автоматически, однозначно идентифицирует объект;
* courseName – название категории;
* courseDescription – описание курса;
* instructor – ссылка на таблицу users, инструктор, который добавил курс;
* category – ссылка на таблицу categories, к какой категории относится курс;

Схема модели lectures представлена на рисунке 3.2.6.

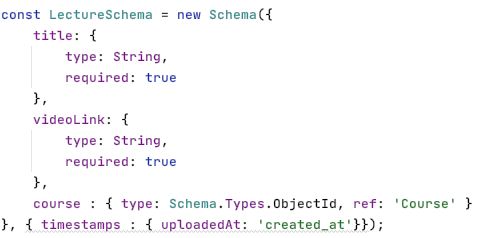


Рисунок 3.2.6 – Модель lectures

Документ lectures содержит информацию о лекциях:

* \_id – генерируется автоматически, однозначно идентифицирует объект;
* title – название лекции;
* videoLink – ссылка на видеоматериал;
* course – ссылка на таблицу courses, к какому курсу относится лекция;

Схема модели enrollment представлена на рисунке 3.2.7.

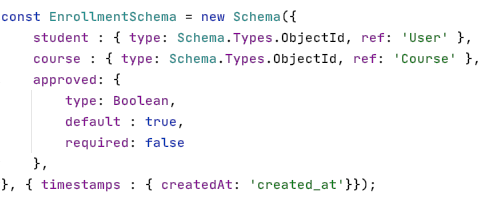


Рисунок 3.2.7 – Модель enrollment

Документ enrollment содержит информацию о записанных на курсы студентах:

* \_id – генерируется автоматически, однозначно идентифицирует объект;
* student – ссылка на таблицу users, студент, который добавил курс;
* course – ссылка на таблицу courses, к какому курсу относится студент;

Первый параметр в методе mongoose.model указывает на название модели. Mongoose затем будет автоматически искать в базе данных коллекцию, название которой соответствует названию модели во множественном числе. Например, в данном случае название модели "User". Во множественном числе в соответствии с правилами английского языка это "users". Поэтому при работе с данными модели User (добавлении, удалении, редактировании и получении объектов) mongoose будет обращаться к коллекции "users". Если такая коллекция есть в бд, то с ней будет идти взаимодействие. Если такой коллекции в базе данных нет, то она будет создана автоматически. Второй параметр функции mongoose.model - собственно схема.

В отличие от реляционных баз данных MongoDB не использует табличное устройство с четко заданным количеством столбцов и типов данных. MongoDB является документо-ориентированной системой, в которой центральным понятием является документ.

Документ представляет набор пар ключ-значение. Ключи представляют строки. Значения же могут различаться по типу данных. В моем случае почти все значения в таблицах представляют какой-то тип, и лишь некоторые из таблицы profile ссылаются на отдельный объект.

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

## **4.3 Авторизация**

В приложении предусмотрена авторизация и регистрация пользователей. Также в приложении присутствуют роли администратора и пользователя. Аутентификация – процедура проверки подлинности идентификации пользователя. В данном случае происходит проверка путем сравнения введенного пароля с паролем, который сохранен в базе данных и соответствует логину. Функция, производящая аутентификацию пользователя представлена в приложении А.

Эта функция принимает логин и пароль, который вводит пользователь, дальше идет поиск пользователя с введенным логином 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. JWT состоит из трех частей: заголовок header, полезные данные payload и подпись signature.

Для формирования JWT использовалась библиотека Passport.js.

Исходный код конфигурации JWT представлен в листинге 4.3.1.

|  |
| --- |
| *//Create jt payload*  const payload = {  id: user.id,  first\_name: user.first\_name,  last\_name: user.last\_name,  avatar: user.avatar,  role: user.role  };  *//Sign token*  jwt.sign(  payload,  keys.secretOrKey,  { expiresIn: 3600 },  (err, token) => {  res.json({  success: true,  token: "Bearer " + token,  first\_name: user.first\_name,  last\_name: user.last\_name  });  }  ); |

Листинг 4.3.1 – Конфигурация JWT на сервере

На стороне клиента, при получении токена, он сохраняется в local storage и устанавливается в заголовок Authorization. После чего мы можем декодировать токен и получить оттуда информацию о пользователе. Методы, выполняющие эти функции, представлены в Приложении А.

# **5 Тестирование**

Для обеспечения корректности работы программы обрабатываются различные ошибки, возникающие в процессе работы. Данное программное средство использует подключение к базе данных, следовательно, неправильно введенные данные или же их отсутствие может повлечь за собой неработоспособность приложения.

На рисунке 5.1 представлена валидация полей при входе в аккаунт пользователем. Т.е если пользователь напишет неверные данные, то получит сообщение об ошибке.

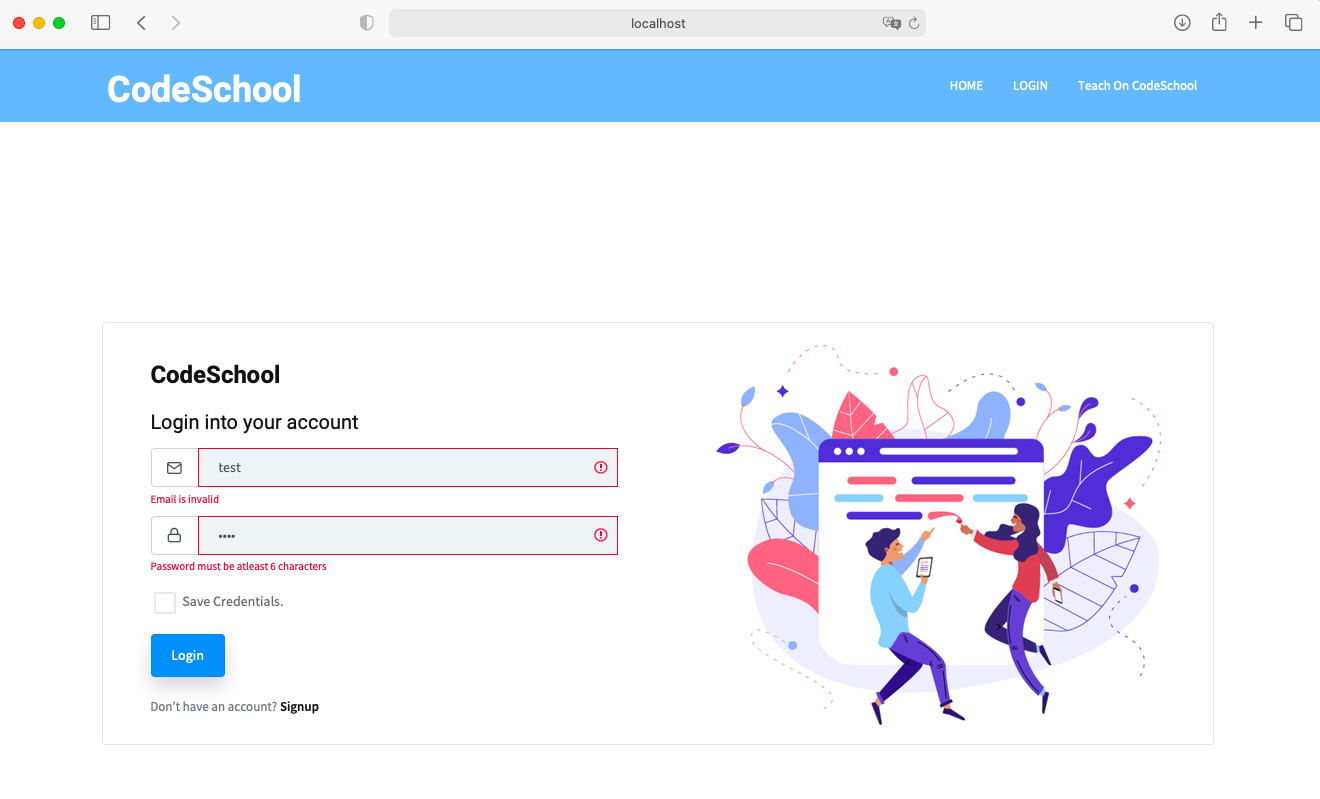


Рисунок 5.1 – Проверка на заполнение всех полей на форме Login

При попытке добавить в лекцию материал неподдерживаемого формата появляется предупреждение как на Рисунке 5.2.

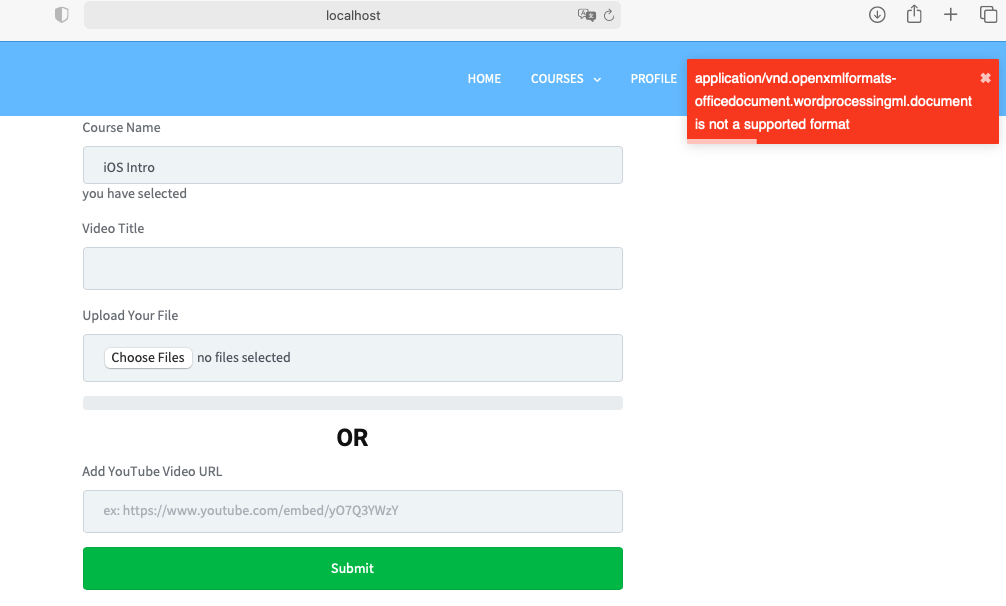


Рисунок 5.2 – Некорректный формат файла

Если пользователь не заполнил все обязательные поля при заполнении профиля, то информация не будет сохранена, а пользователь будет предупрежден, что нужно заполнить поля (Рисунок 5.3).

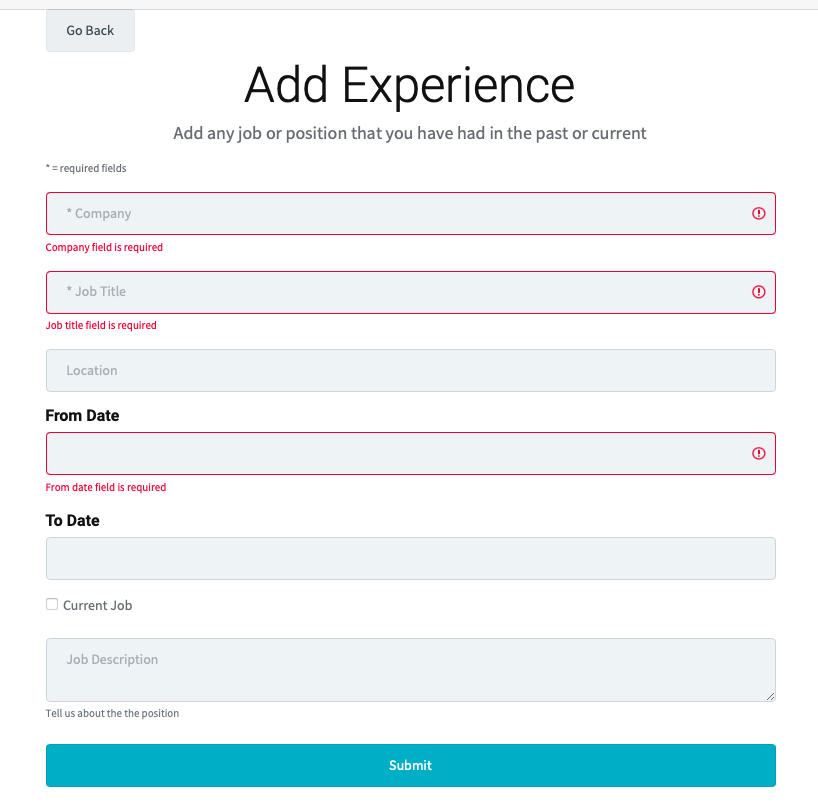


Рисунок 5.3 – Корректно заполненная форма Experience

Тоже самое с добавлением информации об образовании. (Рисунок 5.4).

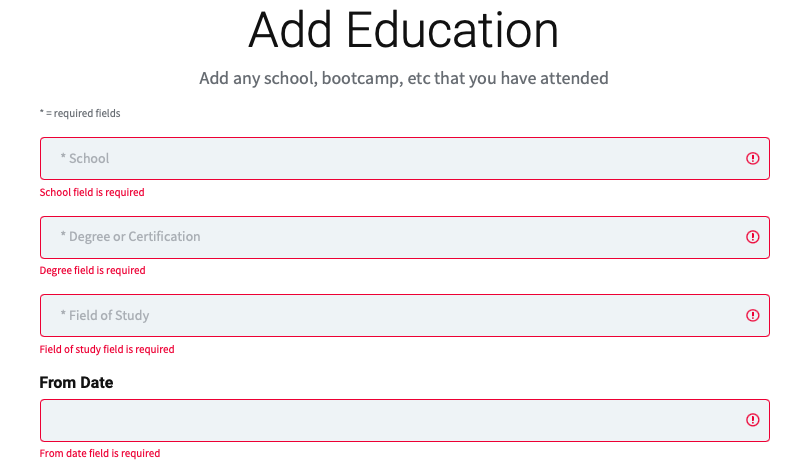


Рисунок 5.4 – Проверка на заполнение всех полей на форме Education

Также при редактировании и создании профиля. Например, если мы хотим удалить текст из поля, которое является обязательным, но мы не сможет обновить информацию и система попросит заполнить обязательные поля (Рисунок 5.5).

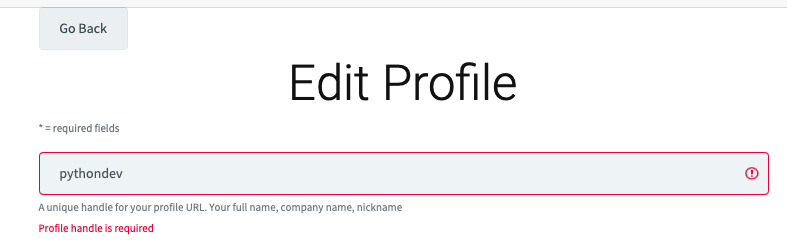


Рисунок 5.5 – Обработка пустых данных в обязательных полях

Если преподаватель попытается добавить пустой курс, то ничего не произойдет, и окно не закроется (Рисунок 5.6)

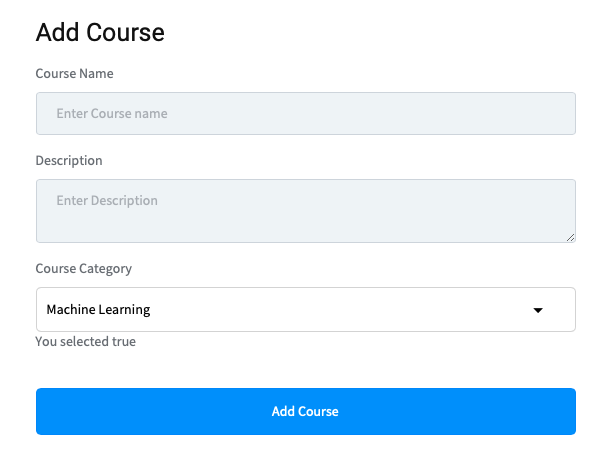


Рисунок 5.6 – Обработка пустых полей

Это значит, что если мы ничего не ввели, то поле не является валидным. Мы не можем добавить такой курс.

# **6 Методика использования программного средства**

При запуске приложения открывается начальная страница, представленная на рисунке 6.1.

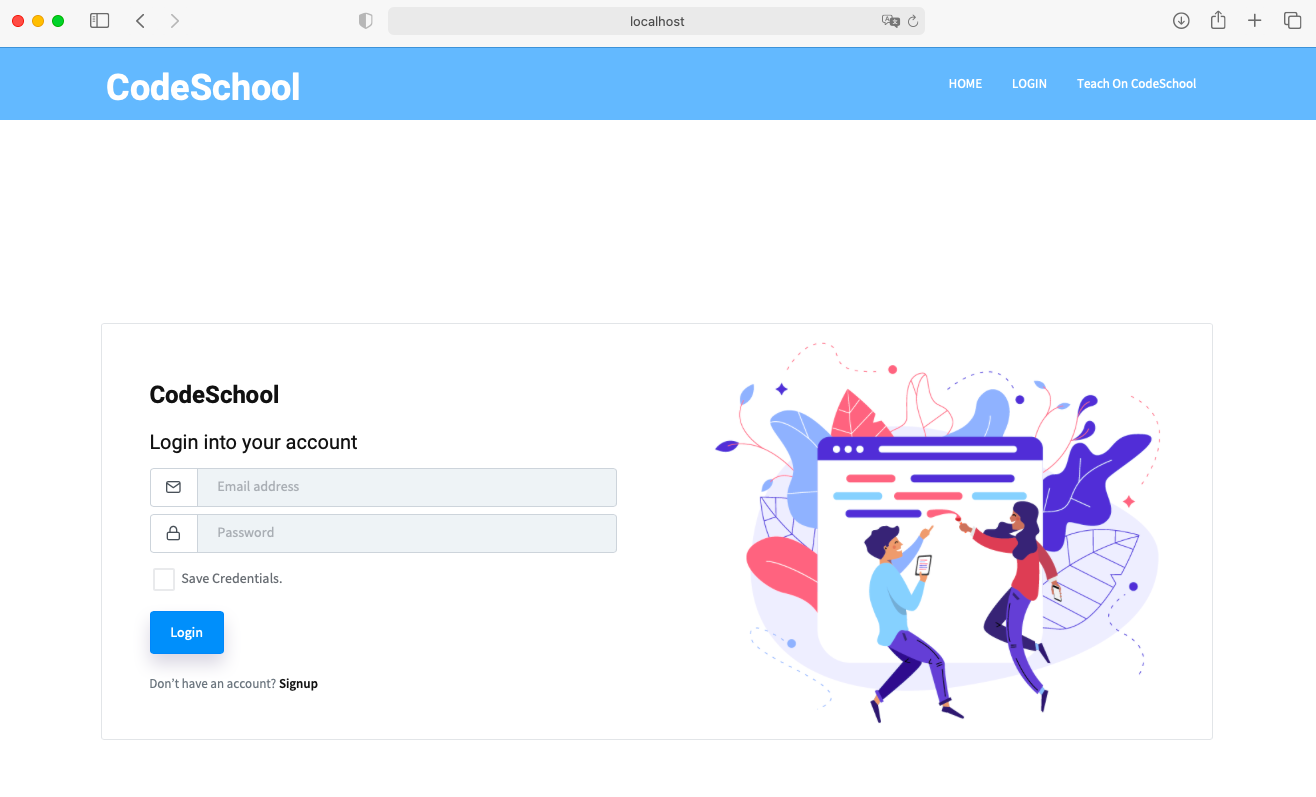


Рисунок 6.1 – Начальная страница

На этом этапе незарегистрированный пользователь может выбрать в качестве кого он будет использовать приложение: Студента или Преподавателя. В зависимости от этого он выбирает для регистрации кнопку Login или Teach On CodeSchool соответственно.

Рассмотрим методику использования приложения для каждой из групп пользователей.

## **6.1 Преподаватель**

После нажатия на Teach On CodeSchool пользователю предлагается войти в систему. (Рисунок 6.2).

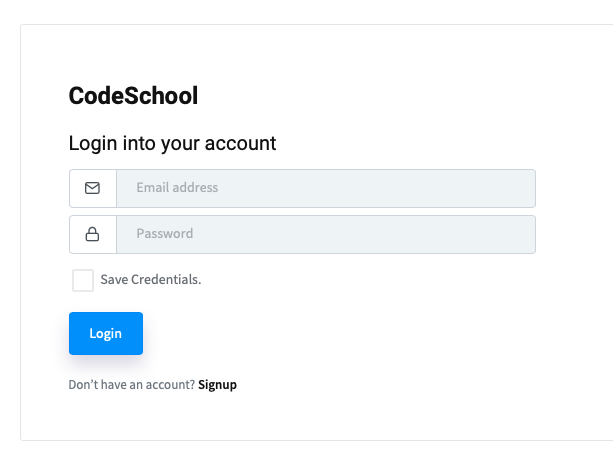


Рисунок 6.2 – Форма входа в систему

Если у пользователя нет аккаунта, то он может создать его, нажав на Signup. После чего появится форма регистрации. (Рисунок 6.3).

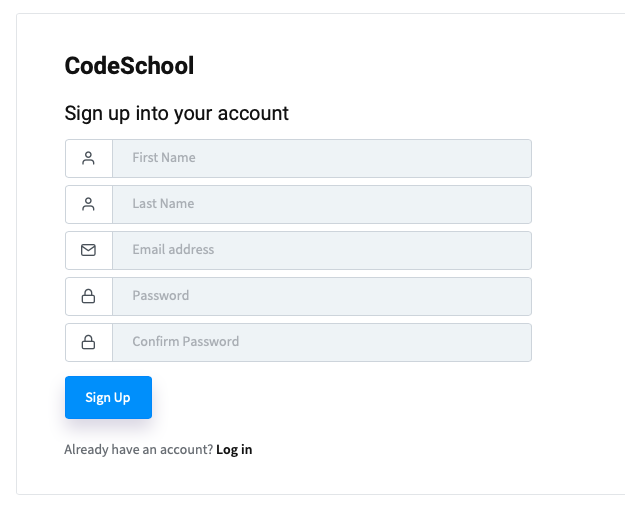


Рисунок 6.3 – Форма регистрации

В случае неверного ввода данных на одной из форм выведется соответствующее сообщение. Также стоит отметить, что все поля являются обязательным к заполнению. Как только пользователь зарегистрируется в системе он сразу же будет перенаправлен на страницу входа в систему, где после ввода данных, он будет перенаправлен на главную страницу, представленную на Рисунке 6.4.

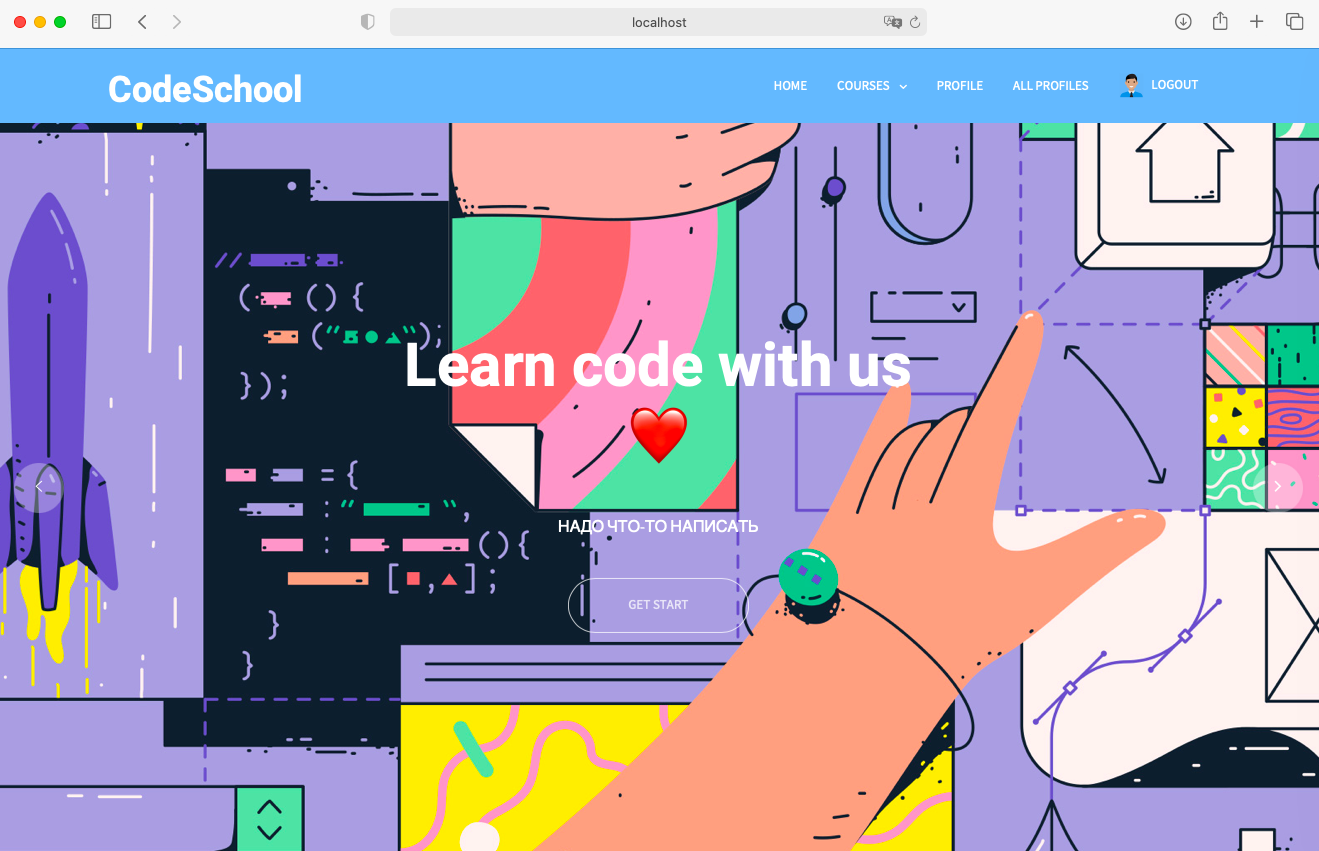


Рисунок 6.4 – Главная страница

В меню представлены все функциональные возможности для пользователя. Для того чтобы добавить курс наводим на панель в меню Courses и выбираем Add Courses. После чего появится форма, для добавления курса, представленная на Рисунке 6.5.

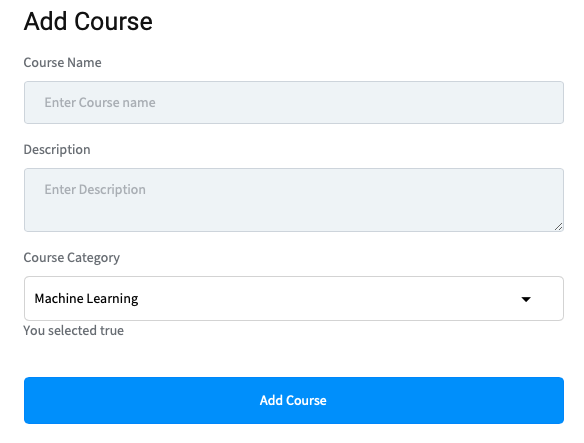


Рисунок 6.5 – Создание курса

Все поля должны быть 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 6.6.

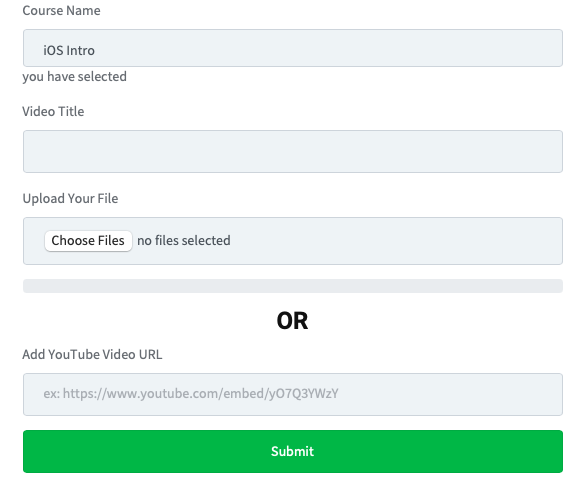


Рисунок 6.6 – Добавление лекции

Тут также все поля являются обязательными. Выбираем курс, к которому хотим добавить лекцию, добавляем название и можем прикрепить видео материалы к лекции. Есть возможность либо загрузить видео с устройства, либо добавить видео с YouTube. После заполнения всех полей нажимаем Submit, и лекция будет добавлена к курсу.

Если прикрепить файл в формате, отличном от видео, то курс не будет добавлен, а пользователь увидить сообщение об ошибке.

Преподаватель также может просматривать свои добавленные курсы и лекции, а также добавленные другими преподавателями.

Для просмотра всех курсов – из этой же вкладки выбирается All Courses, для просмотра своих - My Courses. Выглядят эти вкладки одинаково, как представлено на Рисунке 6.7.

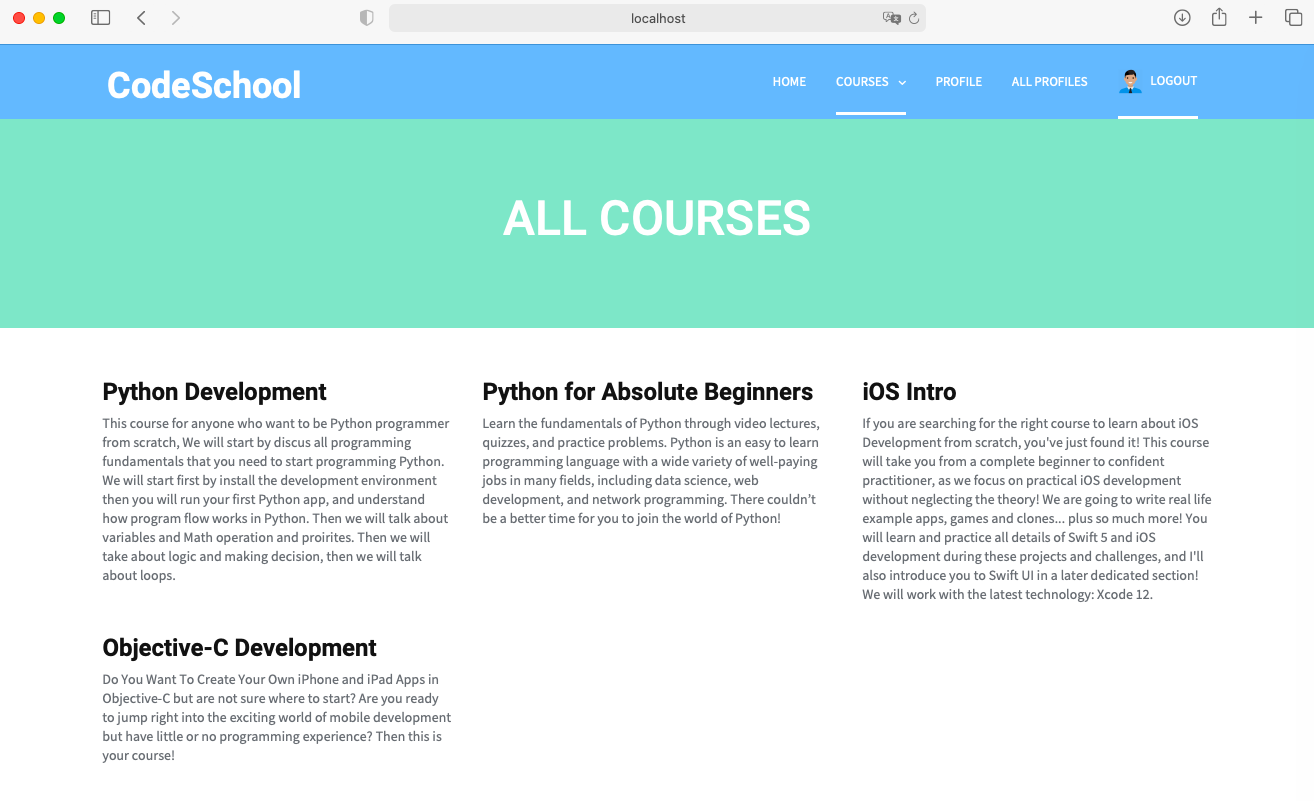


Рисунок 6.7 – Просмотр списка курсов

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

Также пользователь может заполнить свой профиль. Для этого нажимаем на вкладку Profile и нажимаем Create Profile, как на Рисунке 6.8.

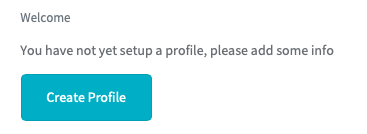


Рисунок 6.8 – Создание профиля

После нажатии появится форма для создания профиля, представленная на Рисунке 6.9.

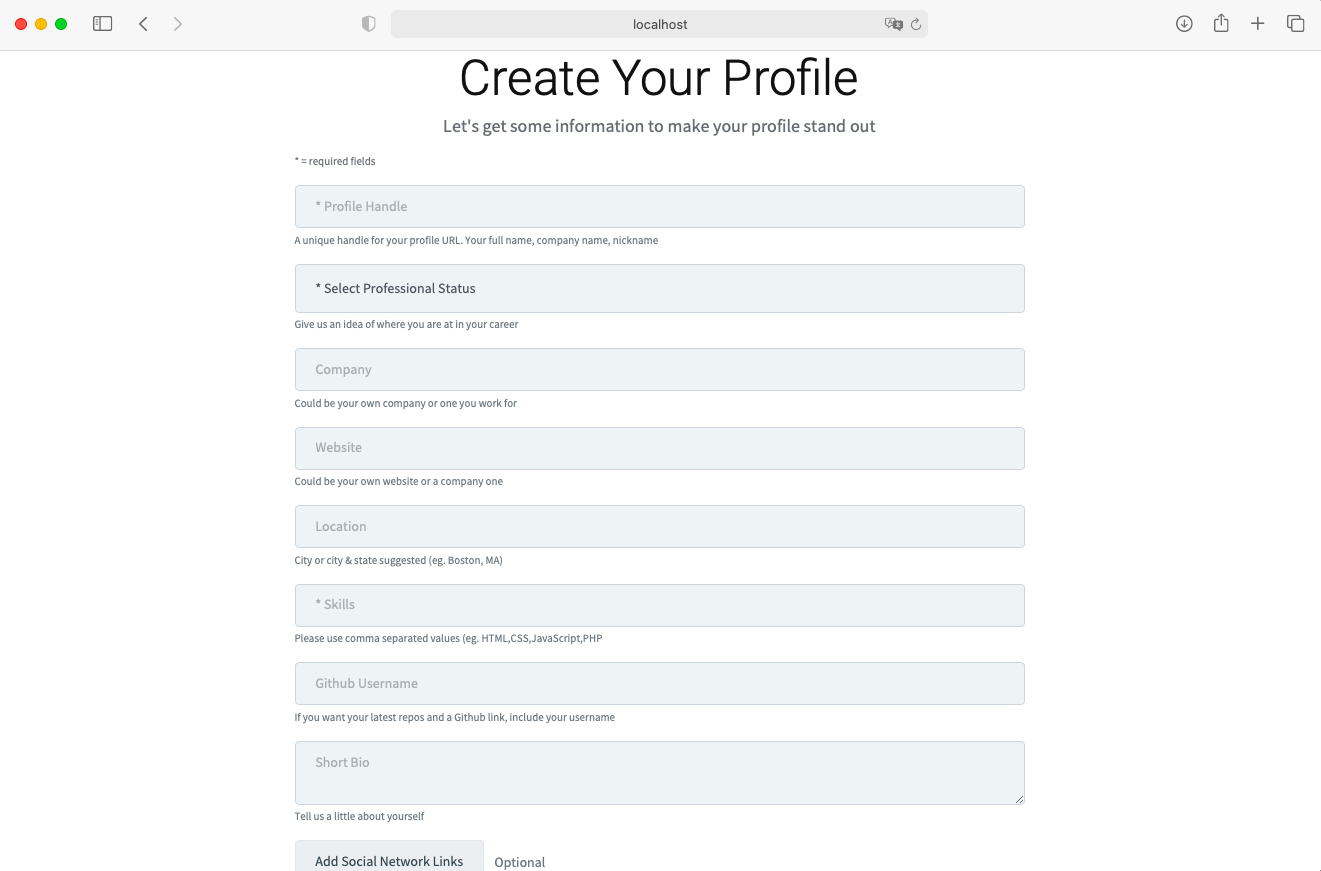


Рисунок 6.9 – Создание профиля

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. Также здесь есть возможность добавить ссылки на социальные сети. Для этого необходимо нажать Add Social Network Links и в форме, представленной на рисунке 6.10, добавить ссылки на желаемые соцсети.

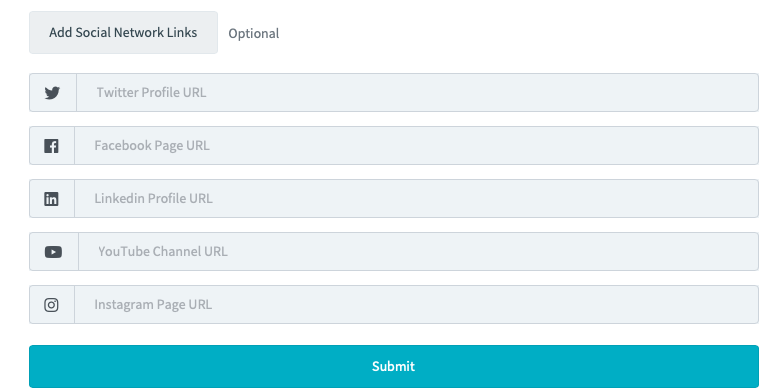


Рисунок 6.10 – Добавление ссылок на соцсети

После заполнения всех полей необходимо нажать на Submit для сохранения профиля. И появится окно, представленное на Рисунке 6.11.

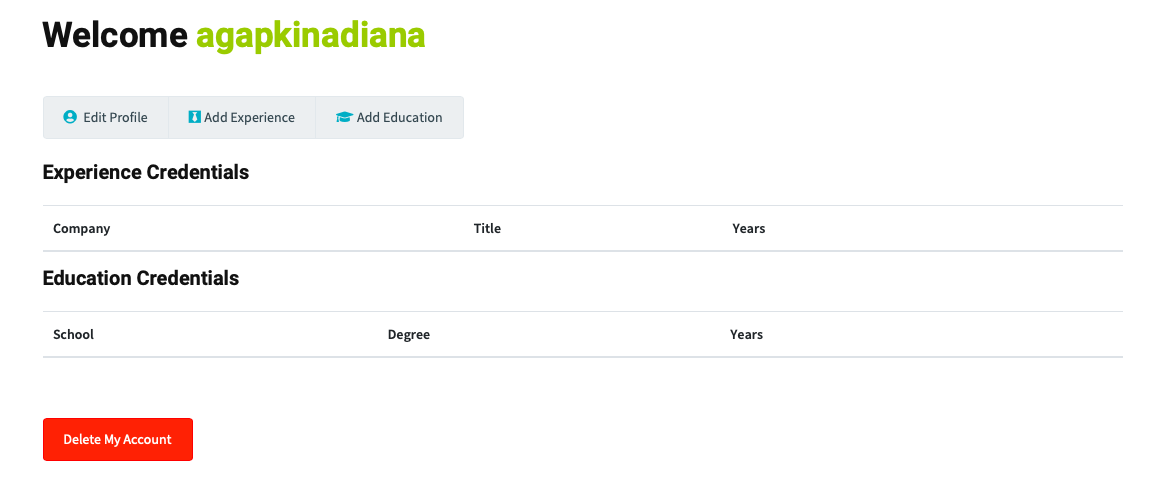


Рисунок 6.11 – Профиль пользователя

Для просмотра профиля необходимо нажать на свое имя пользователя. (Рисунок 6.12).

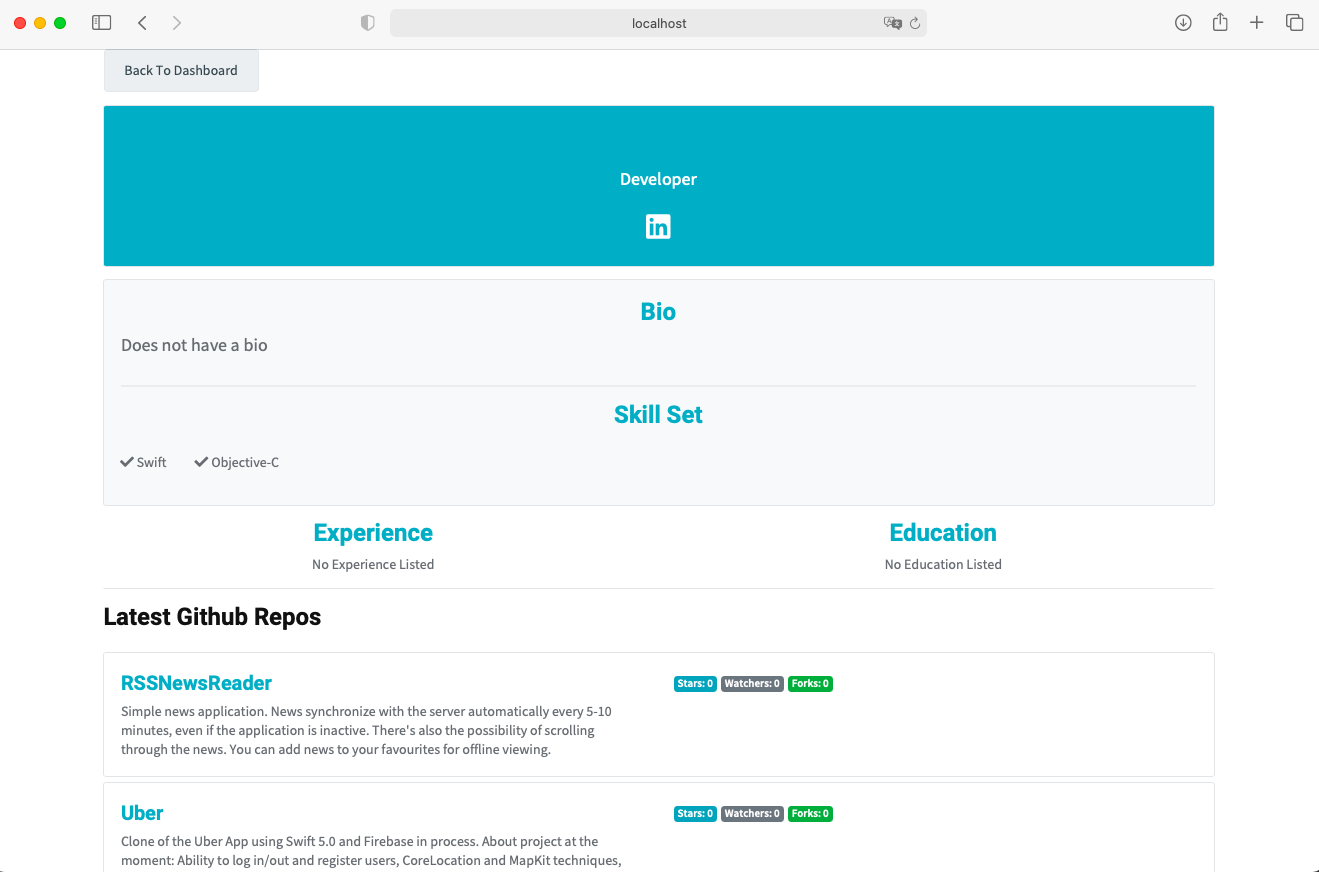


Рисунок 6.12 – Просмотр профиля

Здесь отображается информацию, указанная при добавлении профиля, а также подгружаются репозитории, из указанного GitHub аккаунта.

Также дополнительно можно добавить информацию об образовании и опыте работы. Для этого есть соответствующие кнопки на Рисунке 6.11. Формы были уже представлены в разделе 5. Также пользователь может удалить свой аккаунт, нажав Delete My Account.

Для просмотра профилей всех преподавателей, необходимо нажать на All Profiles в меню, после чего появится страница как на Рисунке 6.13.



Рисунок 6.13 – Просмотр всех профилей

Для просмотра профиля необходимо нажать View Profile и появится страница, как мы уже видели на Рисунке 6.12.

## **6.2 Студент**

При логине и регистрации студент видит все те же формы, как и преподаватель. (Рисунок 6.2, Рисунок 6.3). После входа в систему, студент видит такую же главную страницу, как и преподаватель, но с отличными пунктами меню. (Рисунок 6.14).

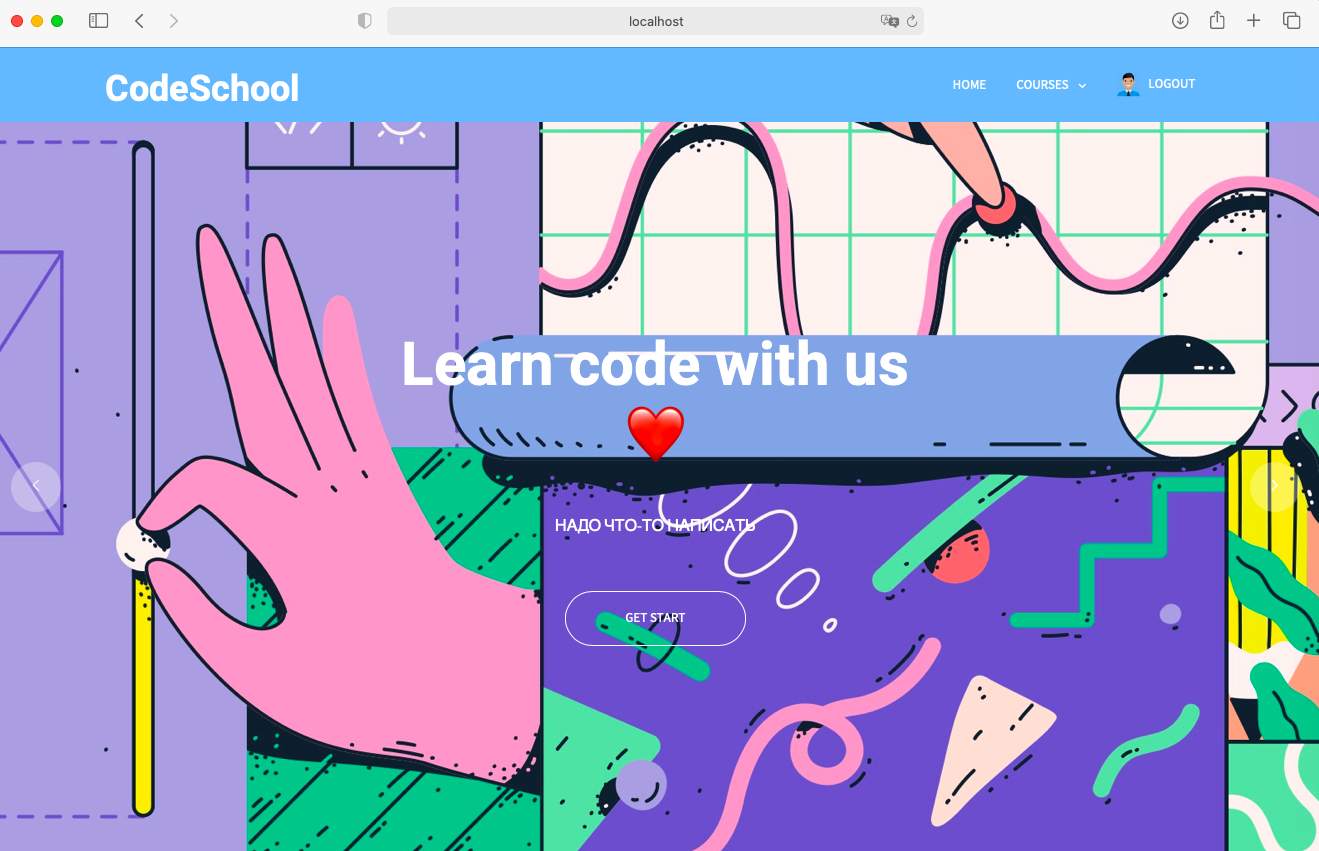


Рисунок 6.14 – Главная страница студента

Также, как и преподаватель, студент может просмотреть все курсы, а также те, которые он добавил в свои. (Рисунок 6.7).

При нажатии на определенный курс, появляется список лекций с материалами, как представлено на Рисунке 6.15.

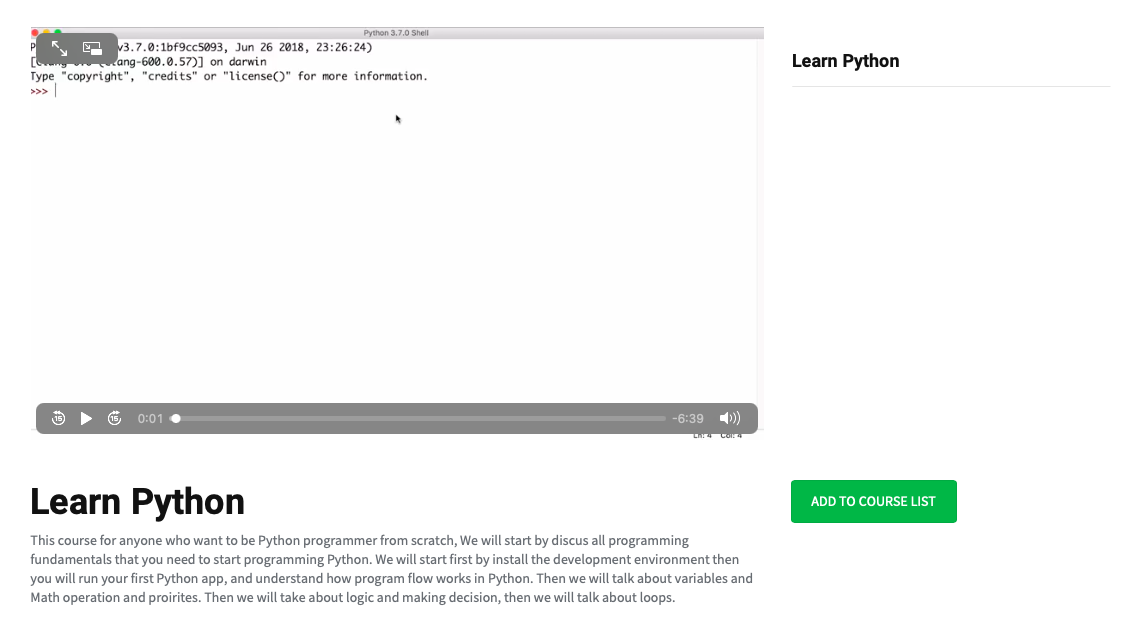


Рисунок 6.15 – Просмотр лекций

Для добавления курса необходимо нажать Add To Course List и он появится во вкладке My Courses.

Для выхода из системы – Logout.

## **6.3 Администратор**

После входа в систему главная страница администратора точно такая же, с различными пунктами меню, которые представлены на рисунке 6.16.



Рисунок 6.16 – Возможности администратора

После входа в систему главная страница администратора точно такая же, с различными пунктами меню, которые представлены на рисунке 6.16.

Администратор имеет возможность просматривать статистику в системе. Диаграммы можно посмотреть на вкладке Dashboard, страница которой представлена на Рисунке 6.17.

Диаграммы администратор может распечатать или сохранить себе в различных представленных форматах.

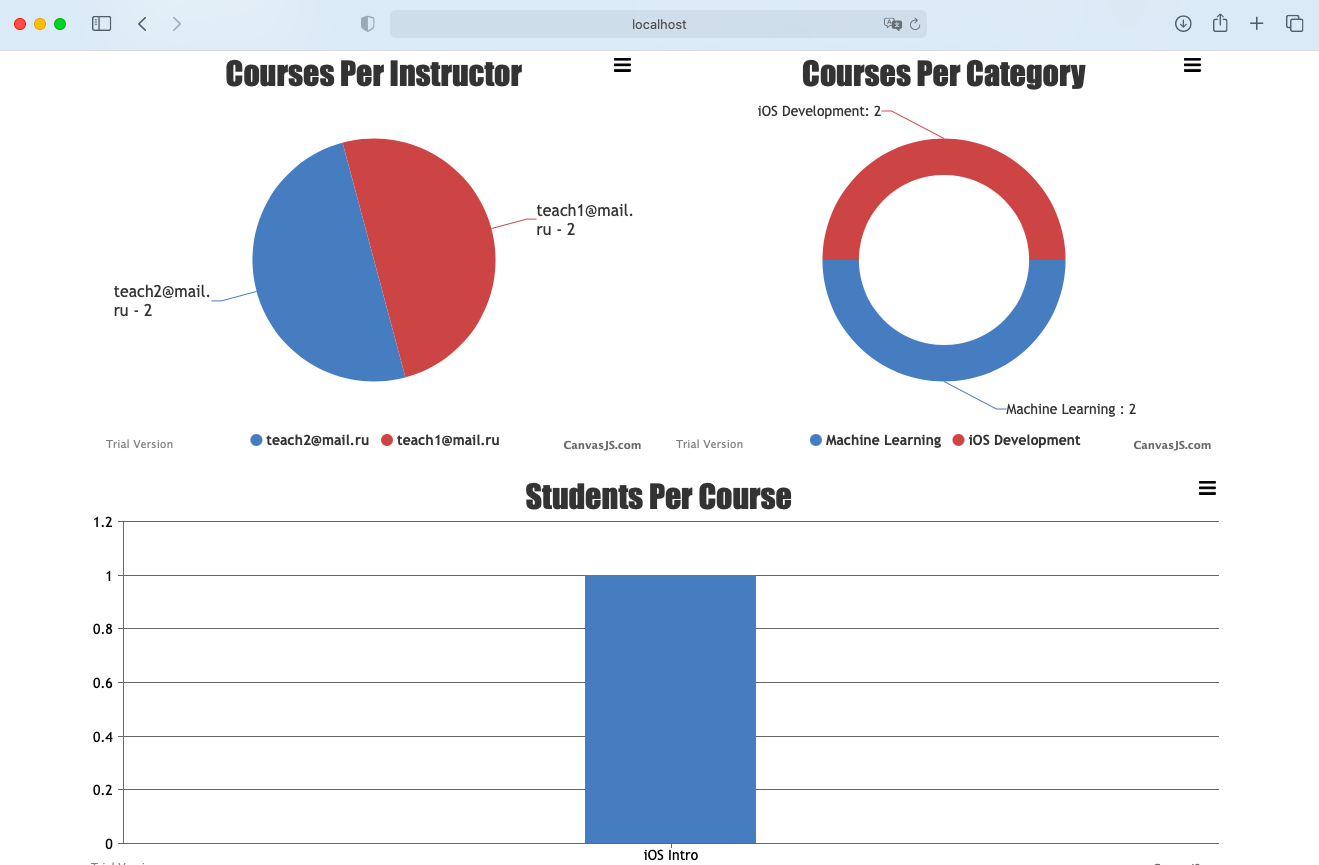


Рисунок 6.17 – Статистика в приложении

Диаграммы отображают количество курсов по инструкторам, количество курсов в различных категориях и количество студентов, которые проходят какой-то определенный курс.

Также есть возможность управления пользователями – вкладка Users (Рисунок 6.18).

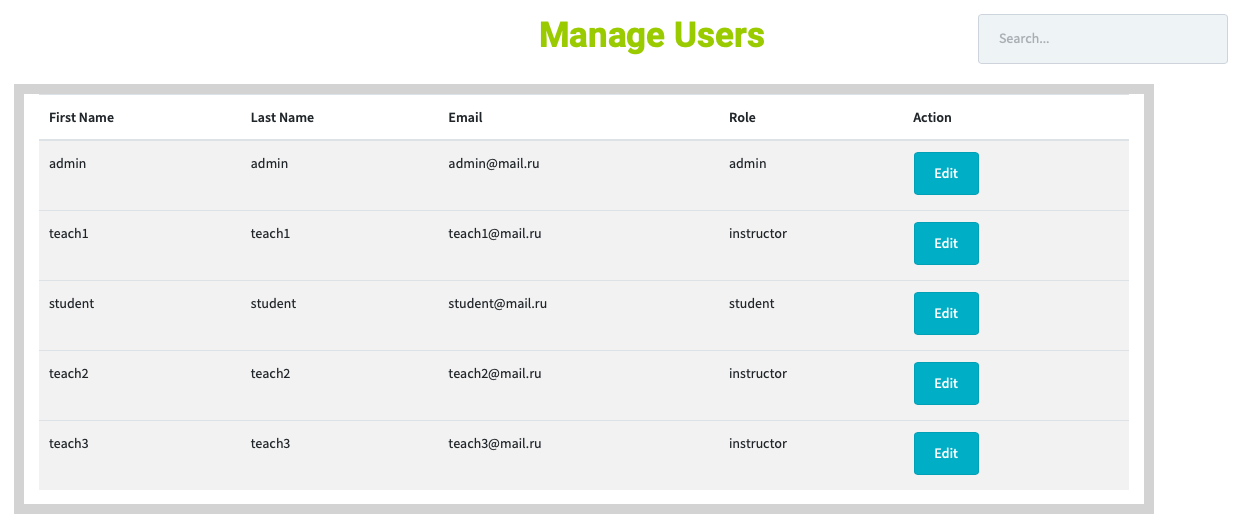


Рисунок 6.18 – Управление пользователями

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 на вкладке Courses, страница которой представлена на Рисунке 6.19.

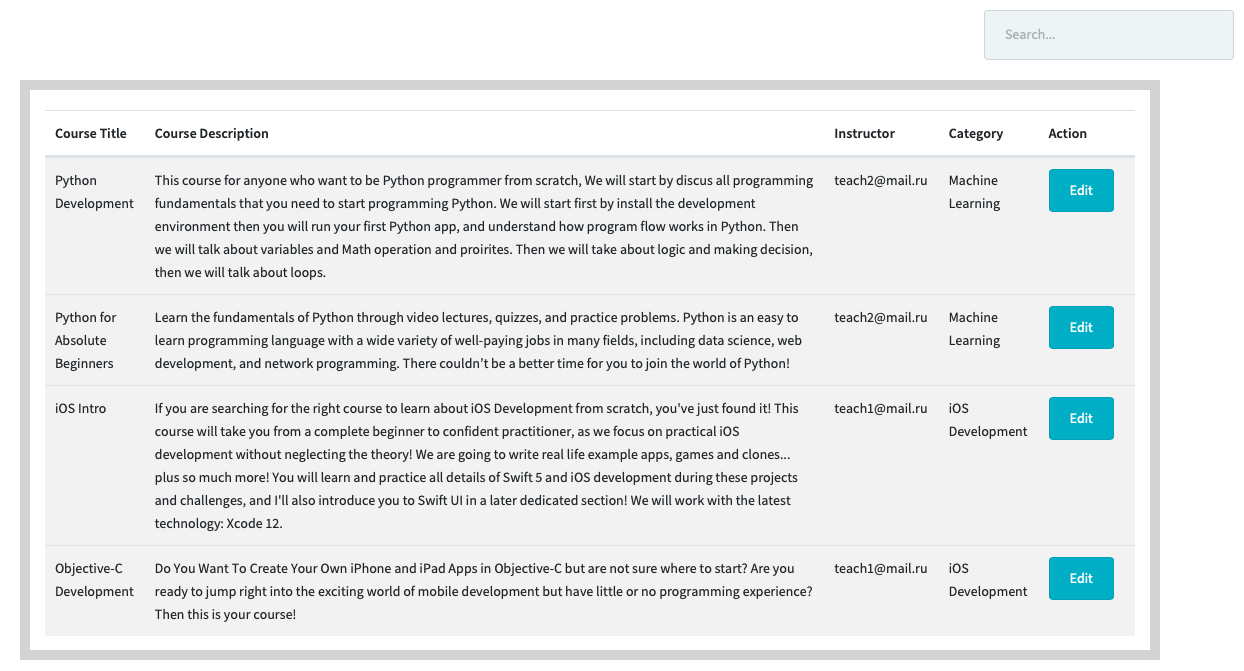


Рисунок 6.19 – Управление курсами

Точно также 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 6.20) нажать Create Category.

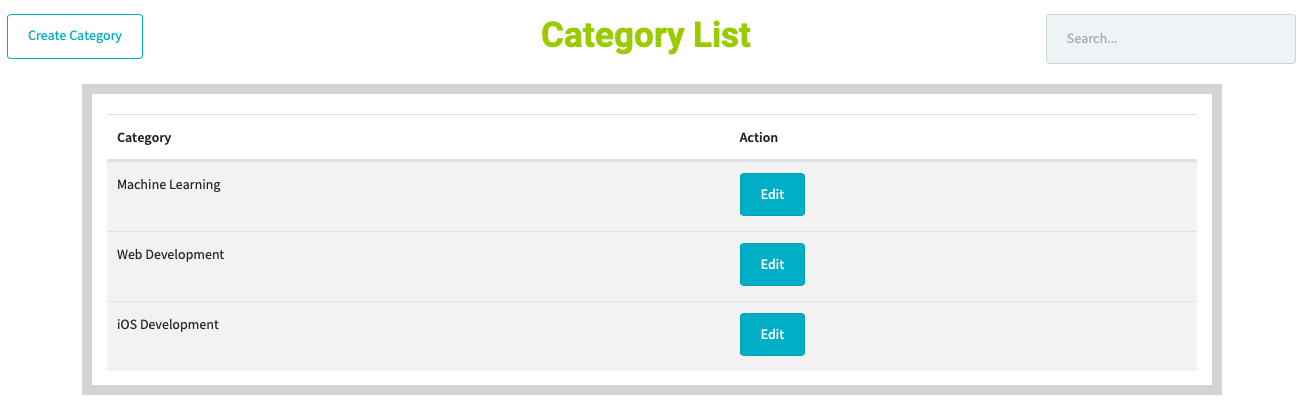


Рисунок 6.20 – Управление категориями

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

Также можно просмотреть список студентов по курсам. (Рисунок 6.21).

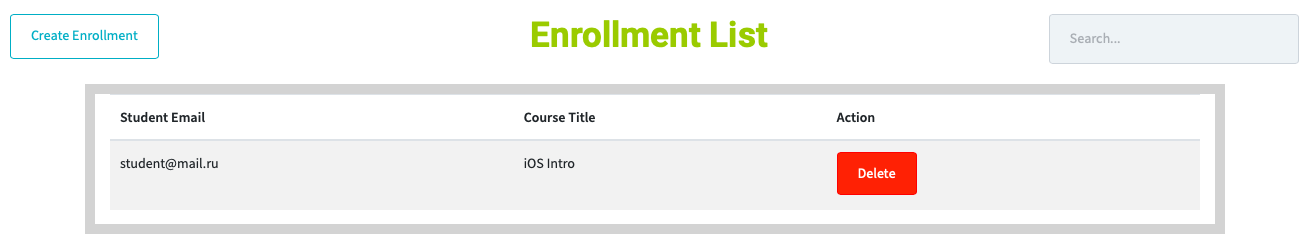


Рисунок 6.21 – Управление студентами

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

# **7 Руководство программиста**

В данной главе будут представлены шаги, описывающие как развернуть приложение при помощи Docker.

Docker – платформа для разработки, доставки и эксплуатации приложений. Основное назначение – упростить развертывание приложения.

Нам понадобится Docker последней версии. В Docker Hub заранее были загружены images, поэтому понадобится лишь загрузить их при помощи команды docker pull. На рисунке 7.1 представлена загрузка image.

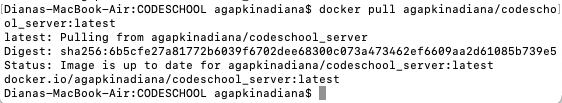


Рисунок 7.1 – Загрузка image

Для проверки, что image загружен успешно, можно использовать команду docker images. Результат работы команды представлен на рисунке 7.2.

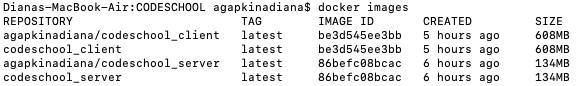


Рисунок 7.2 – Docker images

Аналогично загружаем второй image для frontend части. Далее необходимо запустить ранее загруженный image. На рисунке 7.3 виден результат запуска контейнера.

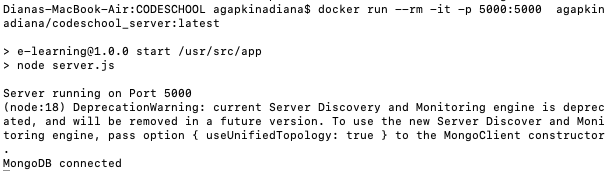
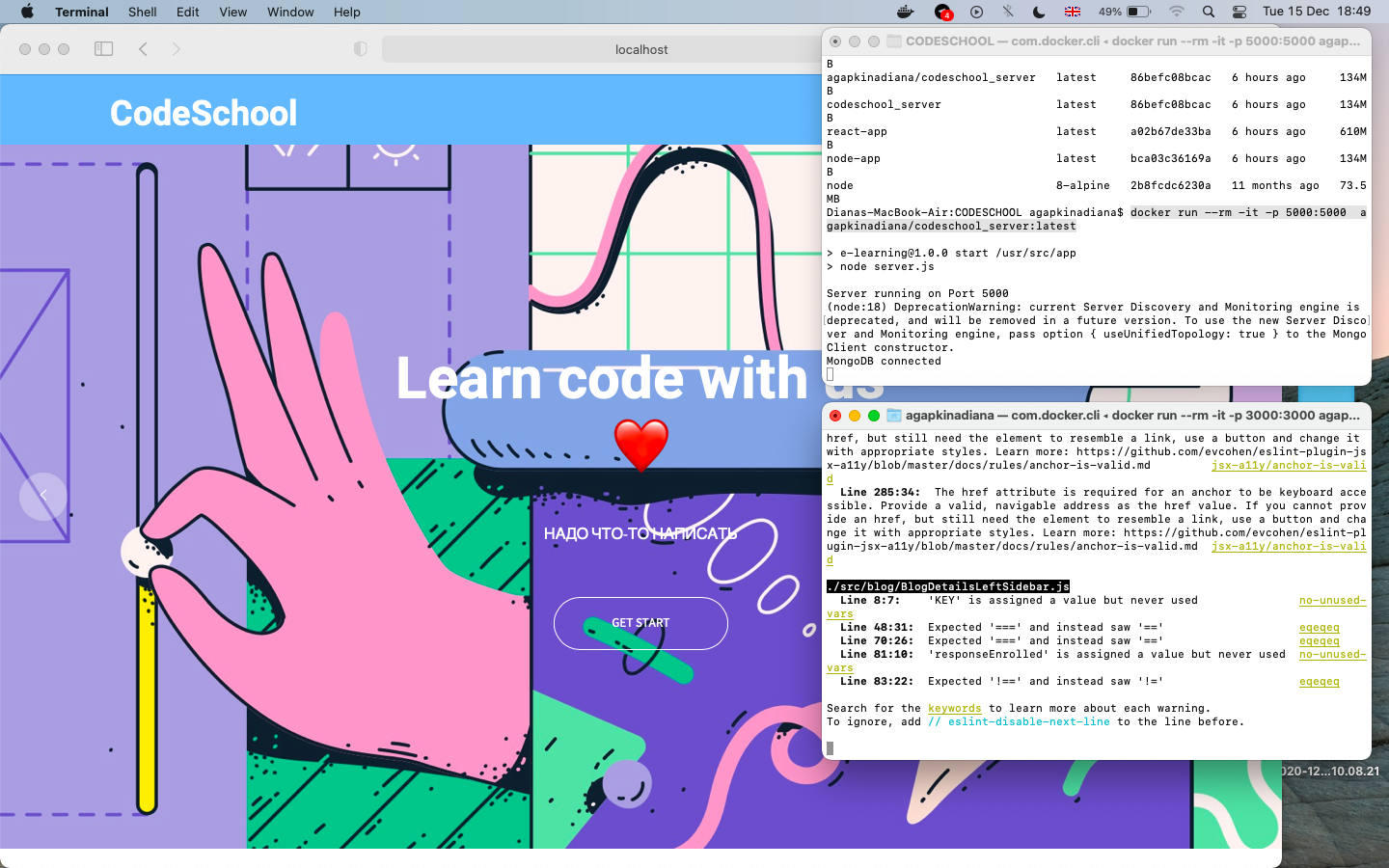


Рисунок 7.3 – Запуск контейнера

На этом рисунке ключ rm означает удаление контейнера после установки, -it нужно для взаимодействия с контейнером, ключ p отвечает за перенаправление порта с контейнера.

Далее проделываем тоже самое с frontend частью приложения, только писать нужно –p 3000:3000. В итоге получаем работающее приложение, которое было развернуто при помощи Docker. На рисунке 7.4 это можно увидеть.

Рисунок 7.4 – Развернутое приложение

Таким образом мы получили работающее приложение, которое было развернуто с помощью Docker.

Для формирования image необходимо было создать Dockerfile, который использовал следующие ключевые слова:

* FROM – «из чего» собираем наш image;
* WORKDIR – рабочая директория;
* EXPOSE – сообщает Docker о том, что контейнер прослушивает указанные сетевые порты во время выполнения;
* RUN – запуск команды;
* COPY – копирование файлов;
* ENTRYPOINT – команда, которая будет выполнена при запуске.

Листинги Dockerfile для клиента и сервера можно посмотреть в Приложении Б.

# **ЗАКЛЮЧЕНИЕ**

В результате выполнения данной курсовой работы было разработано веб-приложение онлайн-школы программирования CodeSchool, которое предназначено для онлайн-обучения программированию. Оно ориентировано абсолютно для любого человека, который хочет делиться своими знаниями с другими или обучаться чему-то новому у других. Благодаря наличию администратора можно контролировать и управлять действиями пользователей, а также отслеживать статистику.

Программное средство предусматривает 4 группы пользователей, которой предусматривает следующие функциональные возможности для каждой из них.

Незарегистрированный пользователь:

* Зарегистрироваться как студент;
* Зарегистрироваться как преподаватель.

Преподаватель:

* Добавить курс;
* Добавить лекцию;
* Просмотр своих курсов;
* Просмотр всех доступных курсов;
* Просмотр лекций;
* Заполнить профиль;
* Просмотр своего профиля;
* Просмотр профилей всех преподавателей;
* Выйти из системы.

Студент:

* Просмотр всех доступных курсов;
* Добавить курс;
* Просмотр добавленных курсов;
* Просмотр лекций из курса;
* Выйти из системы.

Администратор:

* Просмотр статистики;
* Просмотр списка пользователей;
* Редактирование пользователей;
* Просмотр списка всех курсов;
* Редактирование курса;
* 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 MongoD.

Также приложение было развернуто в Docker. Были созданы два образа: клиентской и серверной части. После чего они были опубликованы в Docker Hub для того, чтобы распространить наши образы контейнеров и в последующем протестировать работоспособность приложения на разных операционных системах.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает корректно, а требования технического задания выполнены в полном объеме.

# **СПИСОК ЛИТЕРАТУРЫ**

1. Habr [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/> – Дата доступа: 14.12.2020.
2. React - Role Based Authorization Tutorial with Example [Электронный ресурс]. – Режим доступа: https://jasonwatmore.com/post/2019/02/01/react-role-based-authorization-tutorial-with-example – Дата доступа: 14.12.2020.
3. Metanit.com. Работа с MongoDB [Электронный ресурс]. – Режим доступа: <https://metanit.com/sharp/aspnet5/27.1.php> – Дата доступа: 14.12.2020.
4. Стек MEAN. Mongo, Express, Angular, Node | Холмс Саймон, 496с

# **ПРИЛОЖЕНИЕ А**

|  |
| --- |
| *//Login - Get user token , //loginUser action creator*  export const *loginUser* = userData => dispatch => {  axios  .post("http://localhost:5000/users/login", userData)  .then(res => {  *// console.log(res);*  *//save token to local storage*  const { token } = res.data;  *//set token to local storage*  localStorage.setItem("jwtToken", token);  *//set token to auth header*  *setAuthToken*(token);  *//Decode token to get user data*  const decoded = *jwt\_decode*(token);  *//Set current user*  dispatch(*setCurrentUser*(decoded));  })  .catch(err =>  dispatch({  type: GET\_ERRORS,  payload: err.response.data  })  );  };  *//Set loggid in user , //setCurrentUser action creator*  export const *setCurrentUser* = decoded => {  return {  type: SET\_CURRENT\_USER,  payload: decoded *//actual user with all info*  };  };  *//Log user out , //logoutUser action creator*  export const *logoutUser* = () => dispatch => {  *//Remove token from lc*  localStorage.removeItem("jwtToken");  *//Remove auth header for fututre requests*  *setAuthToken*(false);  *//set current user to {} which will*  dispatch(*setCurrentUser*({}));  };  *//set default header for axios*  *//below fn always attach token to authorization header for every request*  import axios from "axios";  const *setAuthToken* = token => {  if (token) {  *//apply to every request*  axios.defaults.headers.common["Authorization"] = token;  } else {  *//If token is not there then*  *//Delete auth header*  delete axios.defaults.headers.common["Authorization"];  }  };  export default *setAuthToken*; |

# **ПРИЛОЖЕНИЕ Б**

|  |
| --- |
| *# Dockerfile for Node Express Backend*  FROM node:8-alpine  *# Create App Directory*  RUN mkdir -p /usr/src/app  WORKDIR /usr/src/app  *# Install Dependencies*  COPY package\*.json ./  RUN npm install --silent  *# Copy app source code*  COPY . .  *# Exports*  EXPOSE 5000  CMD ["npm","start"]  *# Dockerfile for React client*  *# Build react client*  FROM node:8-alpine  *# Working directory be app*  WORKDIR /usr/src/app  COPY package\*.json ./  *### Installing dependencies*  RUN npm install --silent  *# copy local files to app folder*  COPY . .  EXPOSE 3000  CMD ["npm","start"]  *#docker-compose.yml*  version: '3.7'  services:  server:  build: ./server  ports:  - "5000:5000"  client:  build: ./client  depends\_on:  - server  ports:  - "3000:3000" |