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 Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

WEB-приложение «IT курсы»

Выполнил студент Почиковская Юлия Сергеевна

(Ф.И.О.)

Руководитель проекта асс. 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 асс. Дубовик М.В.

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

Нормоконтролер асс. Дубовик М.В.

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

Курсовой проект защищен с оценкой

Минск 2022

**Содержание**

[Введение 5](#_Toc41642411)

[1 Аналитический обзор литературы и постановка задачи 6](#_Toc41642412)

[1.1 Обзор прототипов 6](#_Toc41642413)

[1.2 Постановка задачи 8](#_Toc41642414)

[1.3 Описание используемых технологий 8](#_Toc41642415)

[2 Разработка архитектуры проекта 10](#_Toc41642416)

[2.1 Проектирование модели базы данных 10](#_Toc41642417)

[2.2 Проектирование структуры web-приложения 10](#_Toc41642418)

[3 Разработка программного средства 12](#_Toc41642419)

[3.1 Реализация модели базы данных 12](#_Toc41642420)

[3.2 Проектирование структуры сервера 13](#_Toc41642421)

[3.3 Проектирование структуры клиента 16](#_Toc41642422)

[4 Тестирование 20](#_Toc41642423)

[4.1 Тестирование формы логина 20](#_Toc41642424)

[4.2 Тестирование работы WebSockets 21](#_Toc41642425)

[4.3 Тестирование работы поиска курсов 21](#_Toc41642426)

[5 Руководство пользователя 22](#_Toc41642427)

[Заключение 27](#_Toc41642428)

[Список литературных источников 28](#_Toc41642429)

[Приложение А 29](#_Toc41642430)

[Приложение Б 30](#_Toc41642431)

[Приложение В 32](#_Toc41642432)

# **Введение**

В последнее время все более популярным становиться онлайн обучение. Многие IT-компании, предоставляют свободное обучение своим сотрудникам, имеют свою внутреннюю закрытую систему, в которой представлена информация о тренингах, которые может предложить компания. Также существуют независимые сервисы, находящиеся в свободном доступе, предназначенные для предоставления доступа к треннингам всех желающим на платной или бесплатной основе.

А в связи с событиями последних лет в мире, онлайн обучение становиться актуальным не только для обычных пользователей, но и для университетов, и школ.

Преимущества, которые предоставляет такой вид обучения способствует развитию новых веб-приложений.

Преимущества

* система работает 24 часа в сутки, 365 дней в году, без перерыва на обед, без выходных и праздничных дней;
* доступ к тренингам может получить любой пользователь, находящийся в любой точке планеты на любом континенте;
* срок и стоимость создания системы несоизмеримо ниже, чем ведение документации и учета всей информации вручную.

Целью моего курсового проекта является создание web-приложения для системы управления курсами, которое предоставляет возможность просмотра информации о курсах, и о тренерах, которые эти курсы ведут.

Основными задачами курсовой работы являются:

* провести анализ прототипов и литературных источников;
* разработать модель базы данных;
* разработать web-приложение с функциональностью, соответствующей теме курсового проекта;
* провести тестирование разработанного web-приложения;
* написать руководство пользователя.

Приложение должно быть разработано с помощью языка JavaScript и программной платформы Node.js, реализуя архитектуру MVC.

Для проектирования базы данных используется система управления базами данных MySQL.

1. **Постановка задачи**

## **1.1 Обзор прототипов**

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

Skillbox [1] – онлайн-университет, в котором обучают 90+ программам для получения востребованных профессий во всем земном шаре. И всё это в режиме online. Страница сайта представлена на рисунке 1.1.

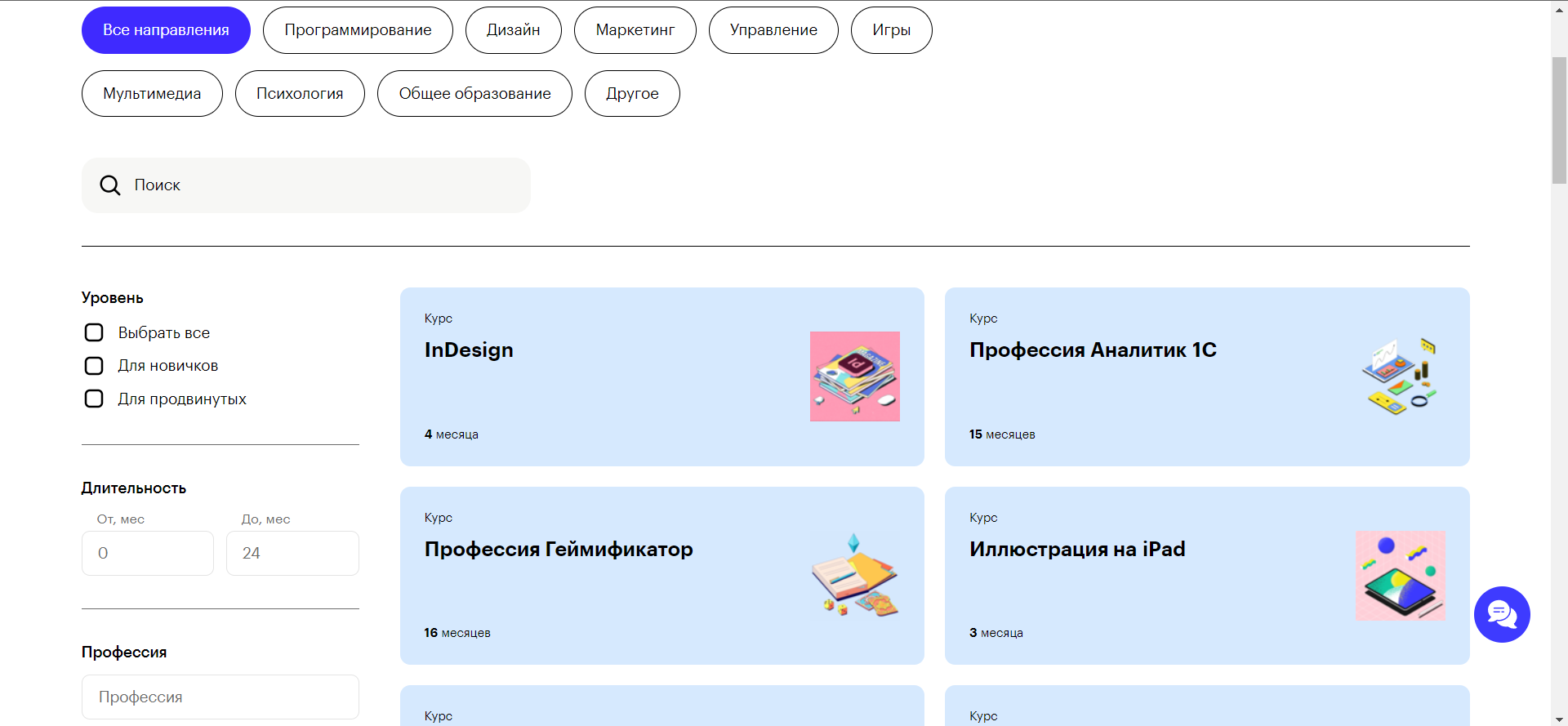


Рисунок 1.1 – Онлайн-университет Skillbox

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.

TeachMeSkills [2] *-* 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.

Сайт обладает динамическими стилями и информативным слайдером на главной странице.

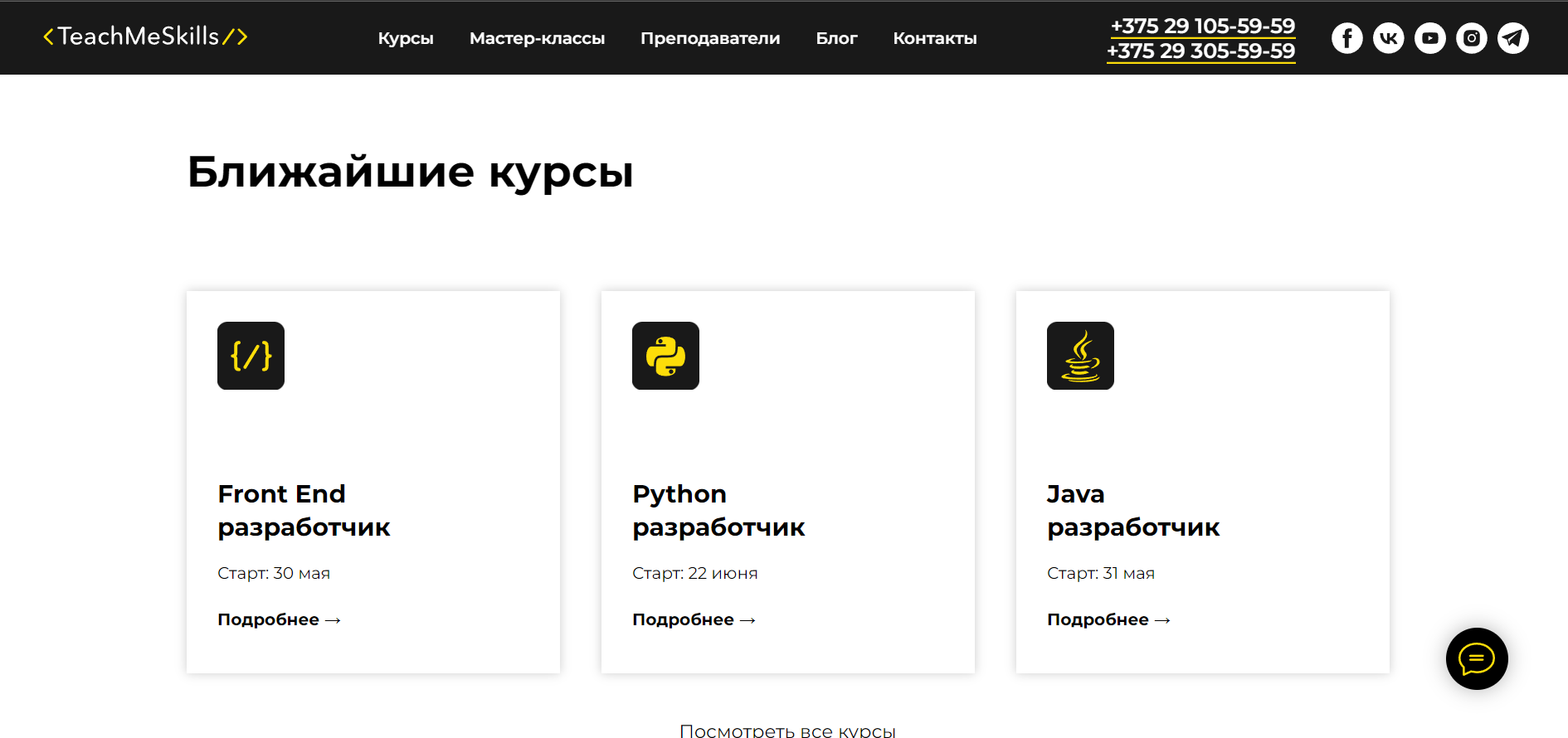


Рисунок 1.2 – Главная страница TeachMeSkills

GeekBrains [3] 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.

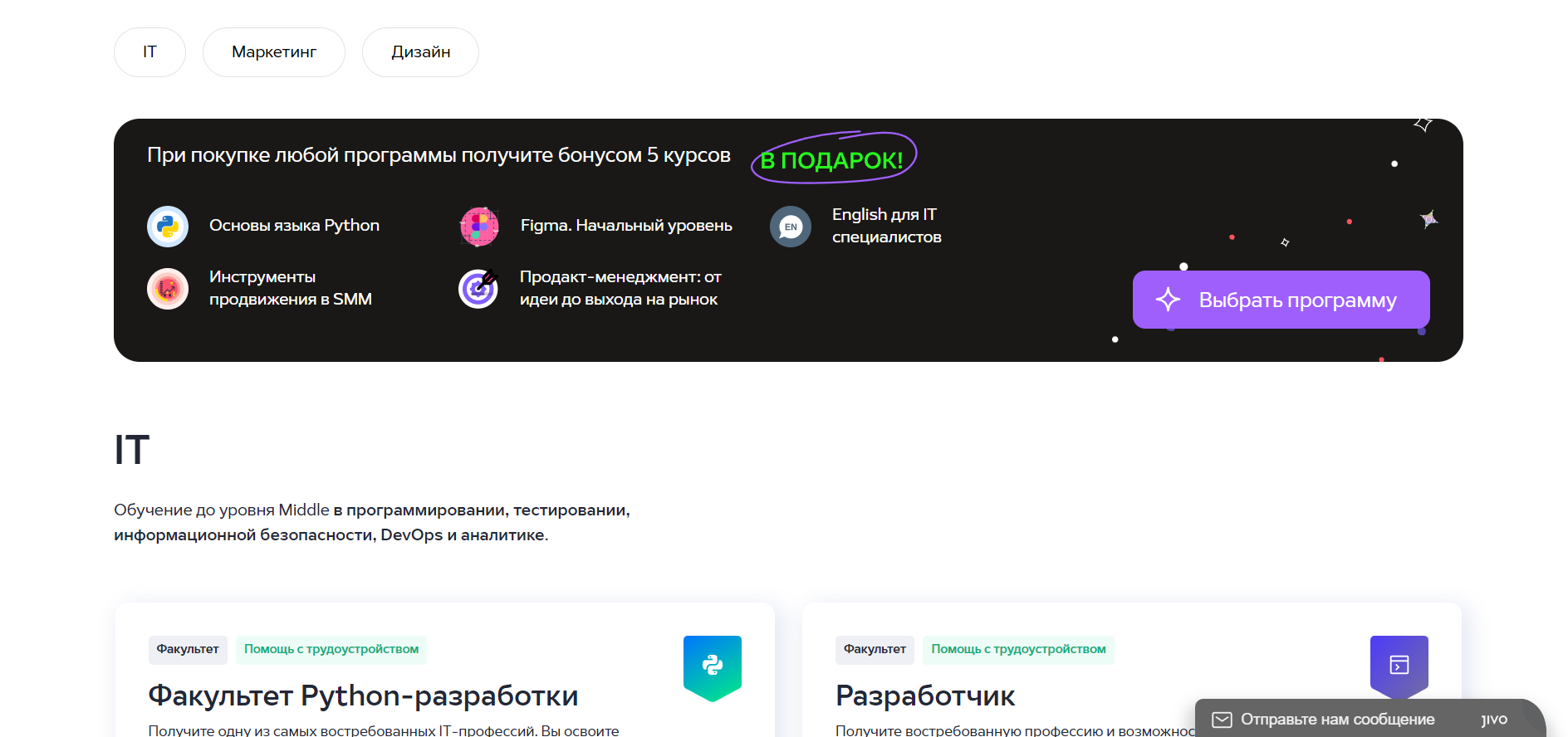


Рисунок 1.3 – Главная страница GeekBrains

Преимущество этого образовательного портала в доступе к множеству бесплатного контента, но главное преимущество – 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.

## **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 сервисом, просмотра информации о тренерах ведущих курс 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, тренеров, а также детальную информацию о каждом из них.

Интерфейс администратора должен предусматривать всевозможные операции с содержимым сайта, описаниями.

Также можно обозначить функциональные требования курсовой работы:

* авторизация администратора;
* возможность просмотра данных о курсах и тренерах для всех пользователей;
* возможность добавления, изменения и удаления данных о курсах и тренерах администратором.

Данные требования будут реализованы в web-приложении и описаны в главе 3 данной пояснительной записки.

## **1.3 Описание используемых технологий**

При проектировании программного средства в качестве программной платформы для сервера был выбран Node.js [4]. 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.

Приложение построено таким образом, чтобы соответствовать архитектуре MVC [5]. Оно разделено на три части Model, View, Controller. Такая архитектура используется для того, чтобы упростить большой по объёму код. MVC применима к разным видам приложений: серверным веб-приложениям, десктопным приложениям.

Для проектирования базы данных используется система управления базами данных MySQL. Эта серверная система способна эффективно функционировать во взаимодействии с интернет-сайтами и веб-приложениями.

Для удобного взаимодействия приложения с базой данных используется ORM-библиотека Sequelize[6] – применяется для приложений на Node.js, осуществляет сопоставление таблиц в базе данных и отношений между ними с классами. При использовании Sequelize мы можем не писать SQL-запросы, а раоъбоать с данными как с обычными объектами. Причем Sequelize может работать с рядом СУБД – MySQL, Postgres, MariaDB, SQLite, MS SQL Server.

Протокол WebSocket тоже будет реализован в приложении. WebSocket [7] является абстракцией над протоколом TCP и используется для обмена данными между клиентом и сервером в режиме реального времени.

1. **Проектирование приложения**

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

## **2.1 Проектирование модели базы данных**

В данной курсовой работе для проектирования базы данных используется система управления базами данных MySQL. MySQL представляет систему управления реляционными базами данных (СУБД). На сегодняшний день это одна из самых популярных систем управления базами данных.

В приложении А представлены коллекции разработанной базы данных ITCOURSES и связи между ними.

Таблица *users* хранит данные об пользователях.

Таблица *courses* содержит информацию о проводимых курсах.

Таблица *enrollments* содержит информацию о пользователях и выбранных ими курсах.

Таблица *teacher* предназначена для информации о преподавателях

## **2.2 Проектирование структуры web-приложения**

В проекте имеется следующие директории. Структура проекта представлена на рисунке 2.1

Директория *config* содержит в себе файл models.js, которые описывает данные, а также логику управления этими данными.

Директория *static* предназначен для хранения статических файлов, представленных в виде картинок, html-файлов являющихся представлениями, css и js-файлов.

Директория *routers* содержит файлы, которые отвечают за обработку логики запроса пользователя.

Директория safety содержит файл, который используется для хеширования паролей пользователей, для безопасного хранения их в базе данных.

Директория *util* содержит файлы, которые отвечают за проверку привилегий пользователей.

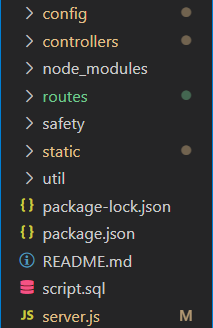


Рисунок 2.1 – Структура программного средства

1. **Разработка приложения**

## **3.1 Реализация модели базы данных**

Для удобной работы с данными из базы данных ITCOURSES используется модуль *mysql2*. Чтобы работать с данными, необходимо разработать модель данных. Листинг создания всех составляющих модели данных представлен в приложении Б.

В процессе создания таблиц нужно также их связать. В листинге 3.1 продемонстрирована связь между таблицей Users и таблицей Enrollments.

**Листинг 3.1 – Связь между таблицами**

Users.hasMany(Enroll, {as:'enroll\_users', foreignKey:'student', sourceKey:'id'})

База данных к приложению подключается так, как показано в листинге 3.2.

**Листинг 3.2 – Подключение базы данных**

global.sequelize = new Sequelize('ITCOURSES', 'root', 'Zalesse2015!', {host:'localhost', dialect:'mysql'})

Отдельно можно рассмотреть некоторые дополнительные функции в файлах модели базы данных. Ниже (листинг 3.3) представлены виртуальные методы для удобного возврата значений из базы данных, а именно, для вывода фотографии и возврата полного имени пользователя (сотрудника).

**Листинг 3.3 – Реализация виртуальных методов**

userSchema.virtual('fullName').get(function() {

    return `${this.firstName} ${this.lastName}`;

});

## **3.2 Проектирование структуры сервера**

Маршрутизация в приложении организована с помощью роутеров. Для этого используется Express. Ниже в листинге 3.4 представлен код добавления роутеров с учётом их расположения в директориях проекта. Также мы настраиваем приложение на использование роутеров по определённому пути.

**Листинг 3.4 – Добавление роутеров для определения маршрутизации запросов**

*let authRouter = require('./routes/auth.route')*

*let coursesRouter = require('./routes/courses.route')*

*let teacherRouter = require('./routes/teacher.route')*

*let enrollmentsRouter = require('./routes/enrollments.route')*

*app.use(authRouter)*

*app.use(coursesRouter)*

*app.use(teacherRouter)*

*app.use(enrollmentsRouter)*

В следующем листинге, представленном в 3.5 можно увидеть реализацию метода GET роутера *courses*.

Изначально запрос просто ищет все курсы, но если в поисковой строке хотя бы частично введено название курса, то запрос строит список по введённым критериям. После этого происходит перенаправление на страницу courses/index с заданными параметрами.

Как только сервер получает данный запрос, он прогружает в браузере страницу с использованием файла index.ejs. Где динамически заполняет её содержимое.

**Листинг 3.5 – Подключение роутеров и обработка GET запроса по адресу '/courses'**

*const* router = express.Router()

router.get('/', async(*req*, *res*) => {

    let query = Course.find();

    if (req.query.title != null && req.query.title != '') {

        query = query.regex('title', new RegExp(req.query.title, 'i'));

    }

    try {

*const* courses = await query.populate('group').exec();

        res.render('courses/index', {

            courses: courses,

            searchOptions: req.query

        });

    } catch {

        res.redirect('/');

    }

});

Листинг всего файла courses.js приведён в приложении В. Остальные роутеры построены по аналогии.

Веб-сокеты также являются одним из пунктов реализации приложения. WebSocket - это протокол передачи данных, основанный на протоколе TCP обеспечивающий обмен сообщениями между клиентом и сервером в режиме реального времени. Для реализации этой технологии был подключен модуль *socket.io*.

Socket.IO — JavaScript-библиотека для веб-приложений и обмена данными в реальном времени. Состоит из двух частей: клиентской, которая запускается в браузере и серверной для node.js. Оба компонента имеют похожее API.

В листинге 3.7 показано подключение модуля *socket.io* для работы с веб сокетами.

**Листинг 3.7 – Подключение веб-сокетов**

*const* io = require('socket.io')(server);

app.use(function(*req*, *res*, *next*) {

    req.io = io;

    next();

});

В данном приложении веб-сокеты помогают обновлять количество отметок «Like» для каждого курса в реальном времени и без перегрузки страницы. Реализация на стороне клиента представлена в листинге 3.8.

**Листинг 3.8 – Реализация веб-сокетов на стороне клиента**

*const* socket = io('https://localhost:4000');

*const* count = document.getElementById('like-count');

*const* form = document.getElementById('like-course');

socket.on('like-course', (*value*) => {

    count.innerText = value;

})

form.addEventListener('submit', *e* => {

    e.preventDefault();

    let liked = Number.parseInt(count.innerText);

    socket.emit('liked', ++liked);

    count.innerText = liked;

})

Код листинга 3.9 представляет собой обработчик GET запроса на стороне сервера к странице с информацией об определенном курсе, где и применяется веб-сокеты.

**Листинг 3.9 – Реализация веб-сокетов на стороне сервера**

router.get('/:id', async(*req*, *res*) => {

    try {

        var course = await (await Course.findById(req.params.id).populate('trainer'))

            .populate('trainer.user')

            .populate('group')

            .execPopulate();

*const* io = req.io;

        io.on('connection', *socket* => {

            socket.on('liked', async(count) => {

                try {

                    course.likeCount = Number.parseInt(count);

                    await course.save();

                    socket.broadcast.emit('like-course', count);

                } catch (*err*) {

                    console.log(err);

                }

            });

        })

        res.render('courses/show', { course: course });

    } catch {

        res.redirect('/');

    }

});

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

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 используется библиотека jsonwebtoken.

## **3.3 Проектирование структуры клиента**

На стороне клиента используются представления. В приложении есть отдельные директории с представлениями для каждого роутера.

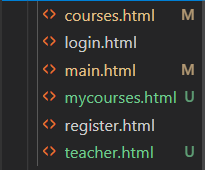


Рисунок 3.1 – Содержимое директория views

В директории static также хранится дополнительный функционал для клиентской части. В нём расположены клиентские стили, файлы клиентской части кода на JavaScript, картинки используемые на страницах приложения

Описанный директорий показан на рисунке 3.14.

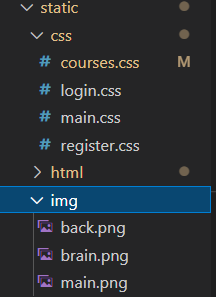


Рисунок 3.2 – Содержимое директория static

В листинге 3.15 представлен пример одной из страниц приложения.

**Листинг 3.15 – Пример страницы Login.html**

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<link rel="preconnect" href="https://fonts.googleapis.com">

<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>

<link href="https://fonts.googleapis.com/css2?family=Montserrat:wght@500&display=swap" rel="stylesheet">

<link rel="stylesheet" href="style/css/login.css">

<title>Login</title>

</head>

<body>

<div>

<h4>Login</h4>

<form action="/login" method="POST">

<input class ="text-field\_\_input" type="text" placeholder="Enter Username" name="username" required/>

</br>

<input class ="text-field\_\_input" type="password" placeholder="Enter Password" name="password" required/>

</br>

<button class="button button1" type="submit">Login</button>

<button class="button button1" type="reset" class="cancelbtn">Cancel</button>

</form>

<h4> Don't have an account? Do you want to register?</h4>

<form action = "/register">

<button class="button button1" type="submit" action = "/register" method ="get">Registration</button>

</form>

</div>

</body>

</html>

# **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.

## **4.1 Тестирование формы логина**

При заполнении формы логина и любой другой формы в приложении пользователь может ввести данные не во все обязательные поля. Пример обработки таких ошибок представлен на рисунке 4.1.

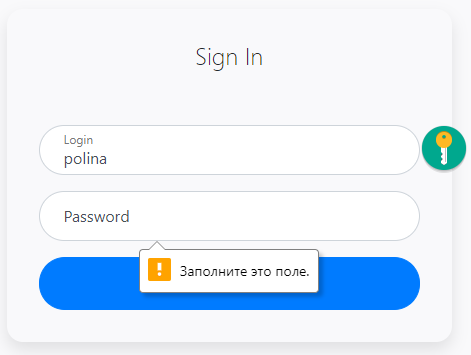


Рисунок 4.1 – Обработка ввода пустых данных

В момент авторизации возможна ситуация, когда пользователь вводит некорректные данные. Обработка таких ошибок происходит с помощью добавления текста под заголовком с описанием ошибки. Пример обработки неверных данных показан на рисунке 4.2.

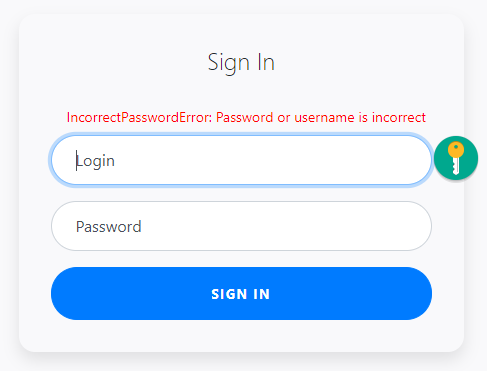


Рисунок 4.2 – Обработка некорректного логина или пароля

## **4.2 Тестирование работы WebSockets**

Используя WebSockets, предполагается, что некоторые действия будут происходить в реальном времени. С помощью данной технологии в приложении реализовано обновление отметок «Like». Попробуем продемонстрировать работу WebSockets на рисунке 4.3.

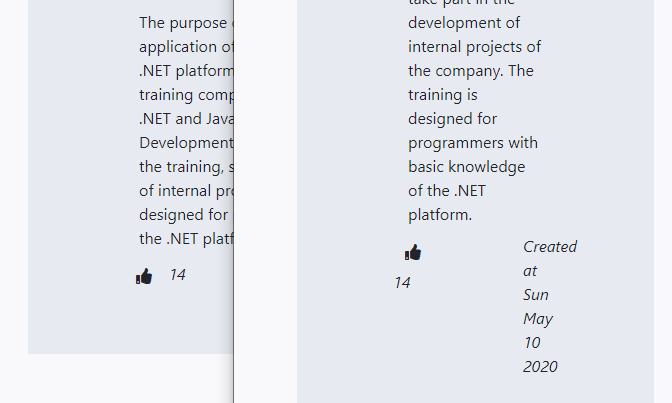


Рисунок 4.3 – Тестирование WebSockets

## **4.3 Тестирование работы поиска курсов**

На странице курсов имеется поисковая строка по названию курсов, с помощью которой можно найти курсы даже по неполному названию. Тестирование данной функции показана на рисунке 4.4.

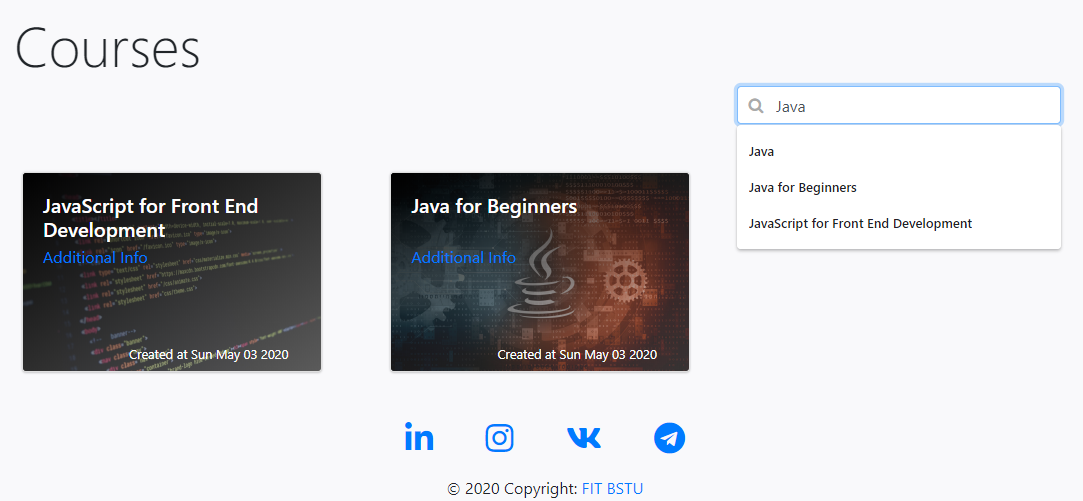


Рисунок 4.4 – Тестироние поиска по названиям курсов

# **Руководство пользователя**

Начальной страницей приложения является страница с последними добавленными курсами (рис. 5.1). Здесь пользователь может перейти по ссылкам «View Course» для перехода на страницу с подробной информацией о курсе.

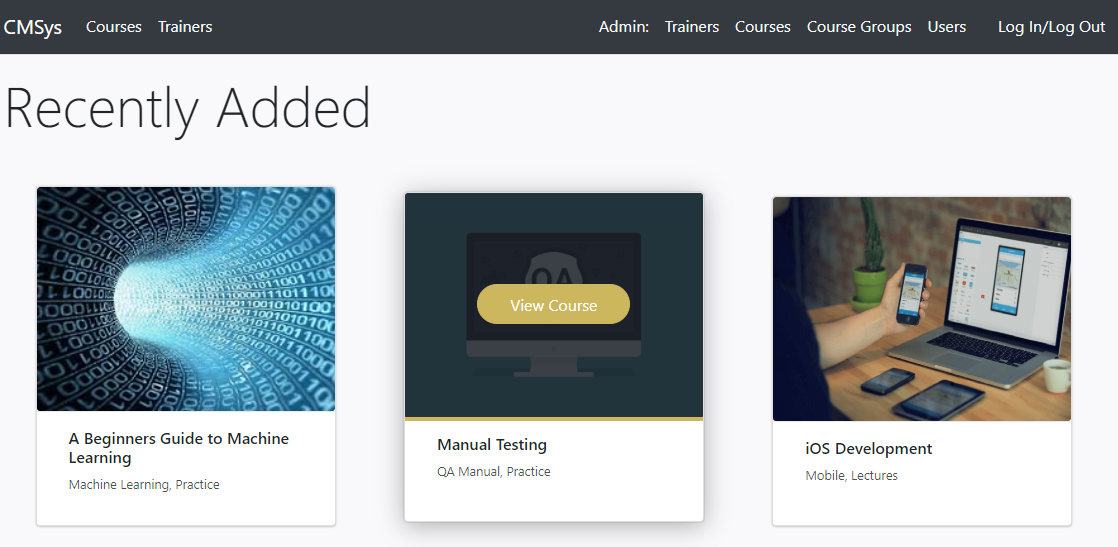


Рисунок 5.1 – Начальная страница web-приложения

Далее пользователь может перейти по вкладкам меню «Courses» или «Trainers» для предоставления соответствующей информации.

Если пользователь перешел на страницу с курсами, то ему предоставляется интерфейс, представленный на рисунке 5.2. На странице имеется поле поиска.

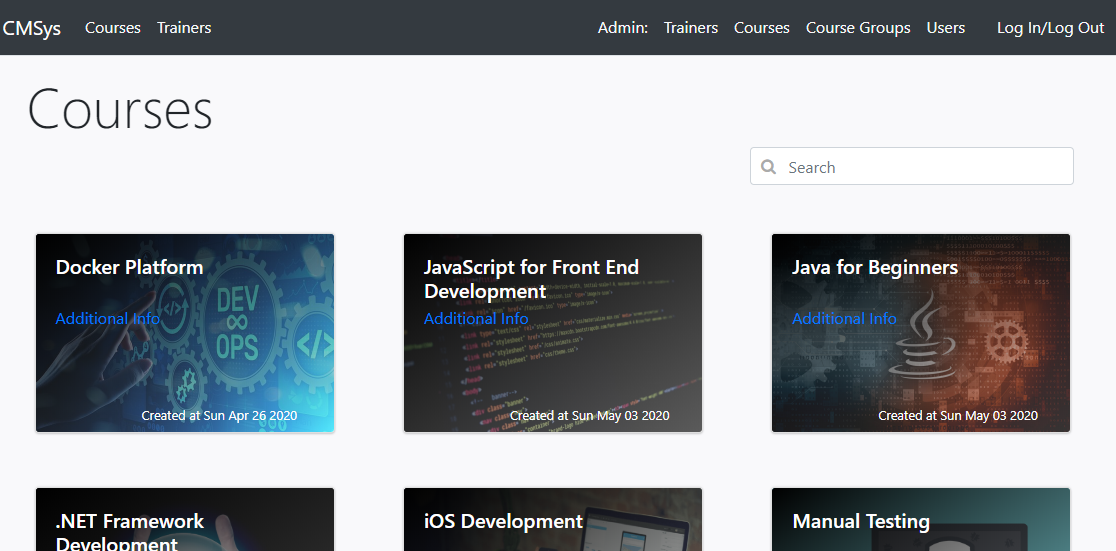


Рисунок 5.2 – Страница со курсами

На странице курсов представлены курсы в виде карт с ссылками на страницы с более подробной информацией о них (рис. 5.3).



Рисунок 5.3 – Страница информации о выбранном курсе

На данной странице имеется ссылка на тренера, который проводит выбранный курс. Нажав на ссылку, открывается модальное окно с информацией о тренере (рис 5.4).

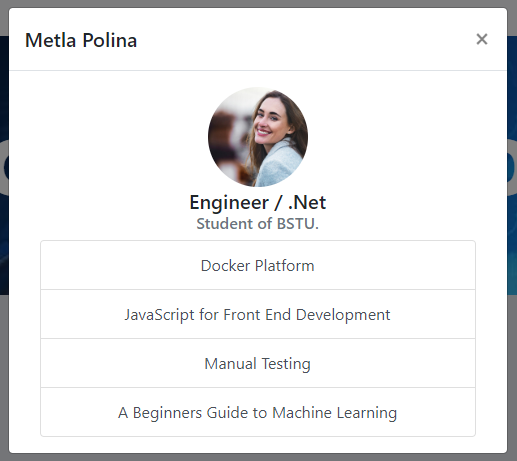


Рисунок 5.4 – Модальное окно с информацией о тренере

На модальном окне также имеются ссылки и на другие тренинги, которые они проводят.

Когда пользователь нажмет на вкладку «Trainers», то откроется страница, представленная на рисунке 5.5.

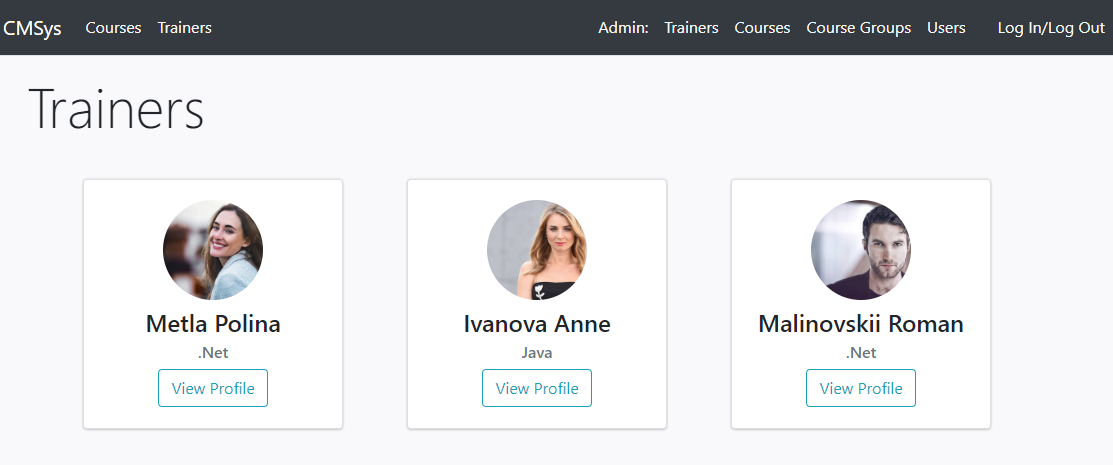


Рисунок 5.5 – Страница тренеров

На данной странице показаны тренера компании, также имеются ссылки, которые открывают модальные окна с более подробной информацией о них.

Для аутентификации необходимо ввести пароль, логин и нажать на кнопку «Войти», что позволит проверить, существует ли пользователь с введенными данными в системе. Все поля являются обязательными для заполнения.

Если пользователь нажмет любую вкладку в меню в правом углу, то его переадресует на форму входа, т.к. та информация доступна только для администратора приложения.

Если пользователь ввел корректные данные администратора, и аутентификация прошла успешно, то происходит перенаправление на начальную страницу (рис. 5.1). Скриншот формы авторизации представлен на рисунке 5.6.

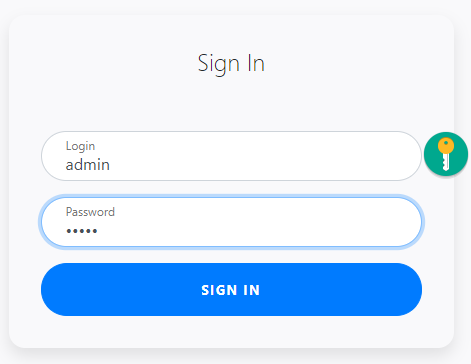


Рисунок 5.6 – Фрагмент экрана с формой для авторизации админа

Администратор имеет доступ к вкладкам «Trainers», «Courses», «Course Groups» и «Users», расположенным справа в меню. Если он захочет разлогониться, то необходимо нажать на «Log In/Log Out» в меню, после чего произойдет перенаправление на страницу логина.

Страницы для администратора выглядят подобными друг на друга. Страница имеет таблицу со всеми данными объекта, который мы выбрали, а также имеет кнопки для добавления, изменения и удаления данных. Пример одной из страниц представлен на рисунке 5.7.

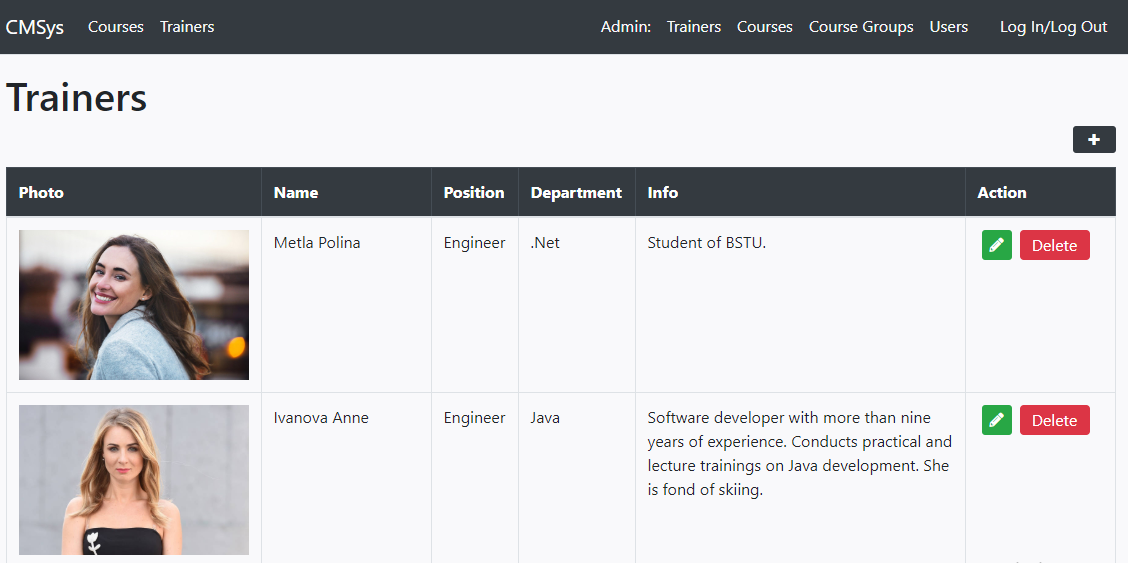


Рисунок 5.7 – Страница тренеров для администратора

Для добавления данных необходимо нажать на кнопку «+», после чего откроется модальное окно, представленное на рис. 5.8.

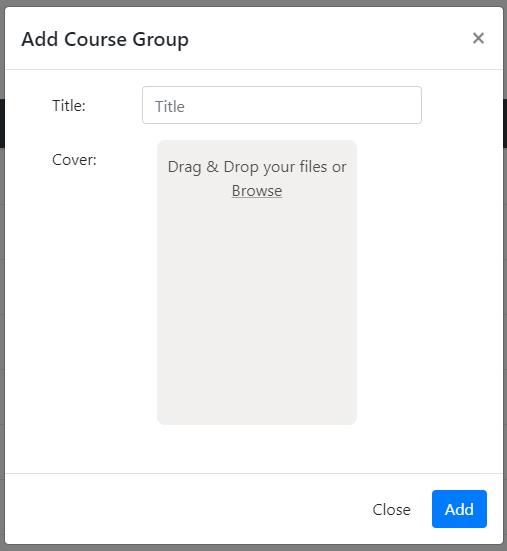


Рисунок 5.8 – Модальное окно добавления группы курсов

Для редактирования данных в таблицах для каждой строки имеется кнопка, при нажатии на которую откроется модальное окно для редактирования данных (рис. 5.9).

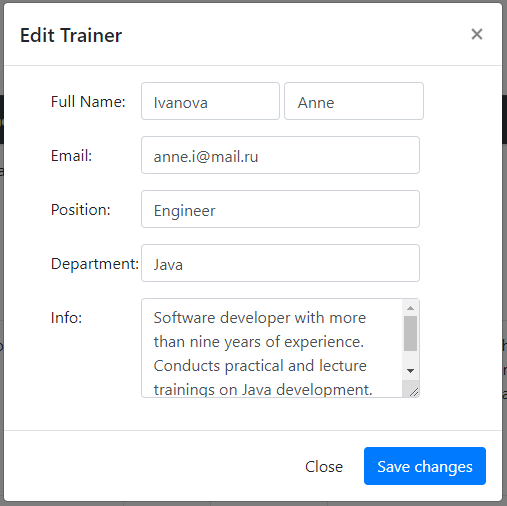


Рисунок 5.9 – Модальное окно для редактирования информации о тренере

Чтобы удалить запись, необходимо нажать на кнопку «Delete» определенной строки таблицы. После чего будет выведено модальное окно с предупреждением об удалении объекта (рис. 5.10), причем данные, зависимые от данного объекта будут также удалены.

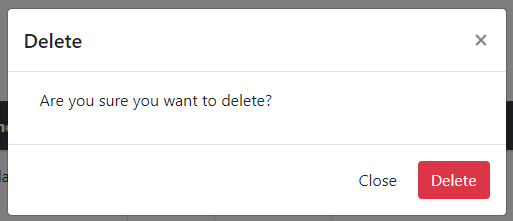


Рисунок 5.10 – Модальное окно предупреждения об удалении объекта

Таким образом реализованы все вкладки администратора, кроме «Users», т.к. подразумевается, что данные о сотрудниках будут подтягиваться с другой базы данных компании.

# **Заключение**

В результате проделанной работы был создано веб-приложение CMSys на основе MVC, с помощью которого любой сотрудник компании может узнать о курсах, которые проводятся в ней, а также получить информацию о тренерах, ведущих эти курсы.

Перед началом разработки программного средства был произведен аналитический обзор прототипов приложений подобной тематики и определение функциональных возможностей разрабатываемого приложения.

Данный проект представляет собой пользовательский интерфейс, предназначенный для просмотра и изменения данных из базы данных. В процессе выполнения курсовой работы была спроектирована база данных для хранения информации о тренерах, курсах и пользователях (сотрудниках компании). База данных была разработана с помощью документно-ориентированной системы управления базами данных «MongoDB».

Приложение написано с помощью платформы Node.js. Связь между базой данных и приложением осуществляется с помощью специальной ODM-библиотеки, которая позволяет определять объекты со строго-типизированной схемой, соответствующей документу MongoDB. Также было написано руководство пользователя для созданного приложения.

В приложении была реализована технология веб-сокетов для обмена сообщениями между клиентом и сервером. Технология AJAX также добавлена в проект для получения данных с сервера. Для повышения безопасности передачи данных в приложении используется HTTPS.

При разработке выполнены следующие пункты:

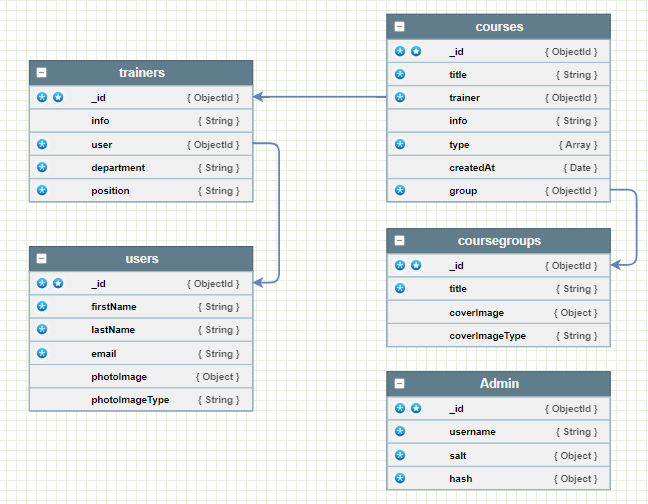
* просмотр информации о курсах и тренерах для все пользователей;
* просмотр информации о курсах, тренерах, группах курсов, а также сотрудниках компании для администратора;
* добавление курсов, тренеров и групп курсов;
* изменение курсов, тренеров и групп курсов;
* удаление курсов, тренеров и групп курсов;
* поиск курсов по названию.

В соответствии с полученным результатом, можно сказать, что разработанная приложение функционирует верно, требования технического задания реализованы в полном объеме, поэтому цель курсового проекта можно считать достигнутой.

# **Список литературных источников**

1. Онлайн-ресурс для обучения программированию «Pluralsight» [Электронный ресурс] – Режим доступа: <https://www.pluralsight.com/> – Дата доступа: 20.05.2020.
2. Онлайн-ресурс школы программирования «TeachMeSkills» [Электронный ресурс] – Режим доступа: <https://teachmeskills.by/> – Дата доступа: 20.05.2020.
3. Онлайн-ресурс для изучения программирования «BitDegree» [Электронный ресурс] – Режим доступа: <https://www.bitdegree.org/> – Дата доступа: 20.05.2020.
4. About Node.js [Электронный ресурс] – Режим доступа: <https://nodejs.org/en/about/> – Дата доступа: 16.05.2020.
5. How to Build and Structure a Node.js MVC Application [Электронный ресурс] — Режим доступа: <https://www.sitepoint.com/node-js-mvc-application/> – Дата доступа: 23.04.2020.
6. Онлайн-руководство по MongoDB [Электронный ресурс] – Режим доступа: <https://metanit.com/nosql/mongodb/> – Дата доступа: 04.04.2020.
7. Introduction to Mongoose for MongoDB [Электронный ресурс] — Режим доступа: <https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/> – Дата доступа: 07.04.2020.
8. Easy Understanding of Web Protocols - HTTP and HTTPS [Электронный ресурс] – Режим доступа: <https://www.izooto.com/blog/understanding-http-https-protocols> – Дата доступа: 18.04.2020.
9. Протокол WebSocket [Электронный ресурс] – Режим доступа: <https://xsltdev.ru/nodejs/tutorial/websocket/> – Дата доступа: 12.05.2020.

# **Приложение А**



# **Приложение Б**

*const* mongoose = require('mongoose')

*const* adminSchema = new mongoose.Schema({

    username: String,

    password: String

});

*const* courseShema = new mongoose.Schema({

    title: {

        type: String,

        maxlength: 64,

        required: true

    },

    info: {

        type: String,

        maxlength: 2000

    },

    type: {

        type: String,

        enum: ['Practice', 'Lectures'],

        required: true

    },

    likeCount: {

        type: Number

    },

    createdAt: {

        type: Date,

        required: true,

        default: Date.now

    },

    trainer: {

        type: mongoose.Schema.Types.ObjectId,

        required: true,

        ref: 'Trainer'

    },

    group: {

        type: mongoose.Schema.Types.ObjectId,

        required: true,

        ref: 'CourseGroup'

    }

});

*const* courseGroupShema = new mongoose.Schema({

    title: {

        type: String,

        maxlength: 64,

        required: true

    },

    coverImage: {

        type: Buffer

    },

    coverImageType: {

        type: String

    }

});

*const* trainerShema = new mongoose.Schema({

    info: {

        type: String,

        maxlength: 2000

    },

    department: {

        type: String,

        maxlength: 128,

    },

    position: {

        type: String,

        maxlength: 128,

        required: true

    },

    user: {

        type: mongoose.Schema.Types.ObjectId,

        required: true,

        ref: 'User'

    }

});

*const* userSchema = new mongoose.Schema({

    firstName: {

        type: String,

        maxlength: 128,

        required: true

    },

    lastName: {

        type: String,

        maxlength: 128,

        required: true

    },

    email: {

        type: String,

        maxlength: 128,

        required: true

    },

    photoImage: {

        type: Buffer

    },

    photoImageType: {

        type: String

    }

});

# **Приложение В**

*const* router = express.Router()

const express = require('express');

const router = express.Router();

const Course = require('../models/course');

const Trainer = require('../models/trainer');

const CourseGroup = require('../models/courseGroup');

const connectEnsureLogin = require('connect-ensure-login');

// All Courses Route

router.get('/', async(req, res) => {

    let query = Course.find();

    if (req.query.title != null && req.query.title != '') {

        query = query.regex('title', new RegExp(req.query.title, 'i'));

    }

    try {

        const courses = await query.populate('group').exec();

        res.render('courses/index', {

            courses: courses,

            searchOptions: req.query

        });

    } catch {

        res.redirect('/');

    }

});

// All Courses For Admin Route

router.get('/admin', connectEnsureLogin.ensureLoggedIn(), async(req, res) => {

    try {

        res.render('courses/admin');

    } catch {

        res.redirect('/');

    }

});

// All Courses For Admin Route

router.get('/admin/api', async(req, res) => {

    try {

        const courses = await Course.find().populate('group').exec();

        const trainers = await Trainer.find().populate('user').exec();

        const groups = await CourseGroup.find();

        res.send({

            courses: courses,

            trainers: trainers,

            groups: groups

        });

    } catch {

        res.status(400);

    }

});

// Create Course Route

router.post('/', async(req, res) => {

    if (!req.body) res.status(400);

    try {

        const trainer = await Trainer.findById(req.body.trainerId);

        const group = await CourseGroup.findById(req.body.groupId);

        const course = new Course({

            title: req.body.title,

            trainer: trainer,

            group: group,

            info: req.body.info,

            type: req.body.type

        });

        await course.save();

        res.send(course);

    } catch {

        res.status(400);

    }

});

// Show Course Route

router.get('/:id', async(req, res) => {

    try {

        var course = await (await Course.findById(req.params.id).populate('trainer'))

            .populate('trainer.user')

            .populate('group')

            .execPopulate();

        const io = req.io;

        io.on('connection', socket => {

            socket.on('liked', async(count) => {

                try {

                    course.likeCount = Number.parseInt(count);

                    await course.save();

                    socket.broadcast.emit('like-course', count);

                } catch (err) {

                    console.log(err);

                }

            });

        })

        res.render('courses/show', { course: course });

    } catch {

        res.redirect('/');

    }

});

// Show Trainer Route

router.get('/api/:id', async(req, res) => {

    try {

        var course = await (await Course.findById(req.params.id).populate('trainer'))

            .populate('trainer.user')

            .populate('group')

            .execPopulate();

        res.send(course);

    } catch {

        res.status(400);

    }

});

// Edit Course Route

router.get('/:id/edit', async(req, res) => {

    try {

        const course = await Course.findById(req.params.id);

        res.send(course);

    } catch {

        res.status(400);

    }

});

// Update Course Route

router.put('/:id', async(req, res) => {

    if (!req.body) res.status(400);

    try {

        const trainer = await Trainer.findById(req.body.trainerId);

        const group = await CourseGroup.findById(req.body.groupId);

        let course = await Course.findById(req.params.id);

        course.title = req.body.title;

        course.trainer = trainer;

        course.group = group;

        course.info = req.body.info;

        course.type = req.body.type;

        await course.save();

        res.send(course);

    } catch {

        res.status(400);

    }

});

// Delete Course Route

router.delete('/:id', async(req, res) => {

    try {

        await Course.findByIdAndDelete(req.params.id, (err, course) => {

            if (err) return console.log(err);

            res.send(course);

        });

    } catch {

        res.status(400);

    }

});

module.exports = router;