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 «Система управления тренингами»

Выполнил студент Метла Полина Георгиевна

(Ф.И.О.)

Руководитель проекта асс. Бурмакова А.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. Бурмакова А.В.

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

Нормоконтролер асс. Бурмакова А.В.

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

Курсовой проект защищен с оценкой

Минск 2020

**Реферат**

Пояснительная записка курсового проекта содержит 34 страницы пояснительной записки, 20 иллюстраций, 9 источников литературы, 3 приложения.

NODE.JS, HTTPS, WEBSOCKETS, MVC, MONGODB, MONGOOSE, AJAX, HTML, CSS, JAVASCRIPT

Основная цель курсового проекта: разработка кроссплатформенного программного средства для управления тренингами.

В первой главе проводится аналитический обзор литературы по тематике курсовой работы и содержит описание технологий, использованных во время выполнения проекта.

Вторая глава посвящена процессу проектирования системы.

В третьей главе описывается процесс разработки и принципы функционирования проекта.

В четвёртой главе описано тестирование используемых технологий и программного средства.

В пятой главе описано руководство пользователя, позволяющее подробно понять интерфейс программного средства.

В заключении приведены результаты проделанной работы.

**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**

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.

На сайте системы обычно представлен подробный каталог тренингов с информацией о нем, а также информация о тренерах, которые будут вести тренинг. На основе такой информации пользователь решает, стоит ли ему записаться на курс.

Преимущества, которые даёт веб-приложение:

* 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 сотрудник компании, 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, которые проводятся внутри компании, и о тренерах, которые эти курсы ведут.

Основными задачами курсовой работы являются:

* провести аналитический обзор литературы;
* 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 MongoDB.

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, предоставляющие 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 становится все менее популярными. На смену им приходят онлайн-сервисы.

## **1.1 Обзор прототипов**

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

*Pluralsight* [1] - это довольно популярный онлайн-ресурс, на котором можно научиться программированию смотря видео. Не важно новичок вы или эксперт, который хочет улучшить свои способности, на Pluralsight каждый может найти обучающий видеокурс для себя. Главная страница сайта представлена на рисунке 1.1.

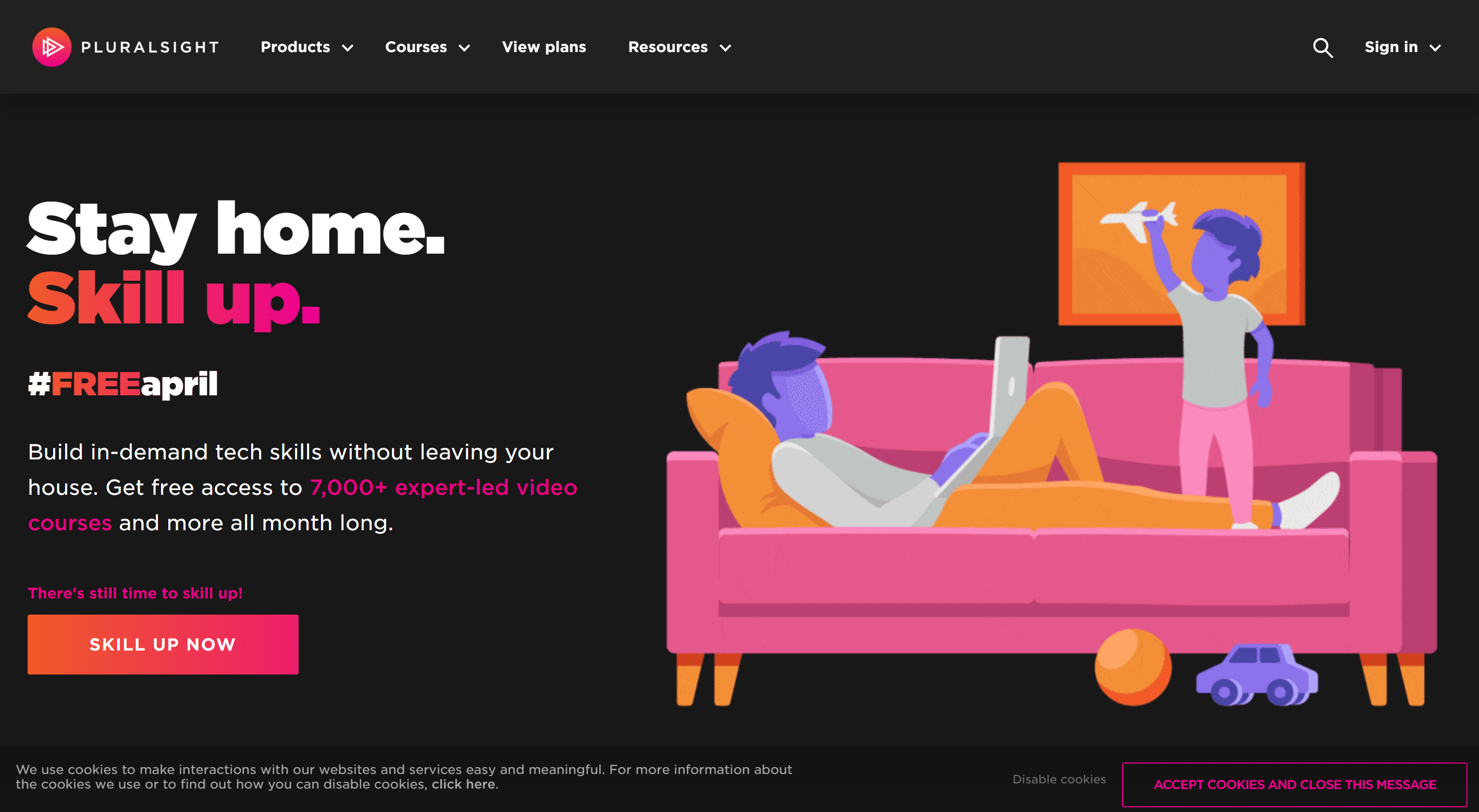


Рисунок 1.1 – Главная страница Pluralsight

Сайт имеет интуитивно-понятный интерфейс, приятный дизайн и обладает высокой функциональностью.

*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.

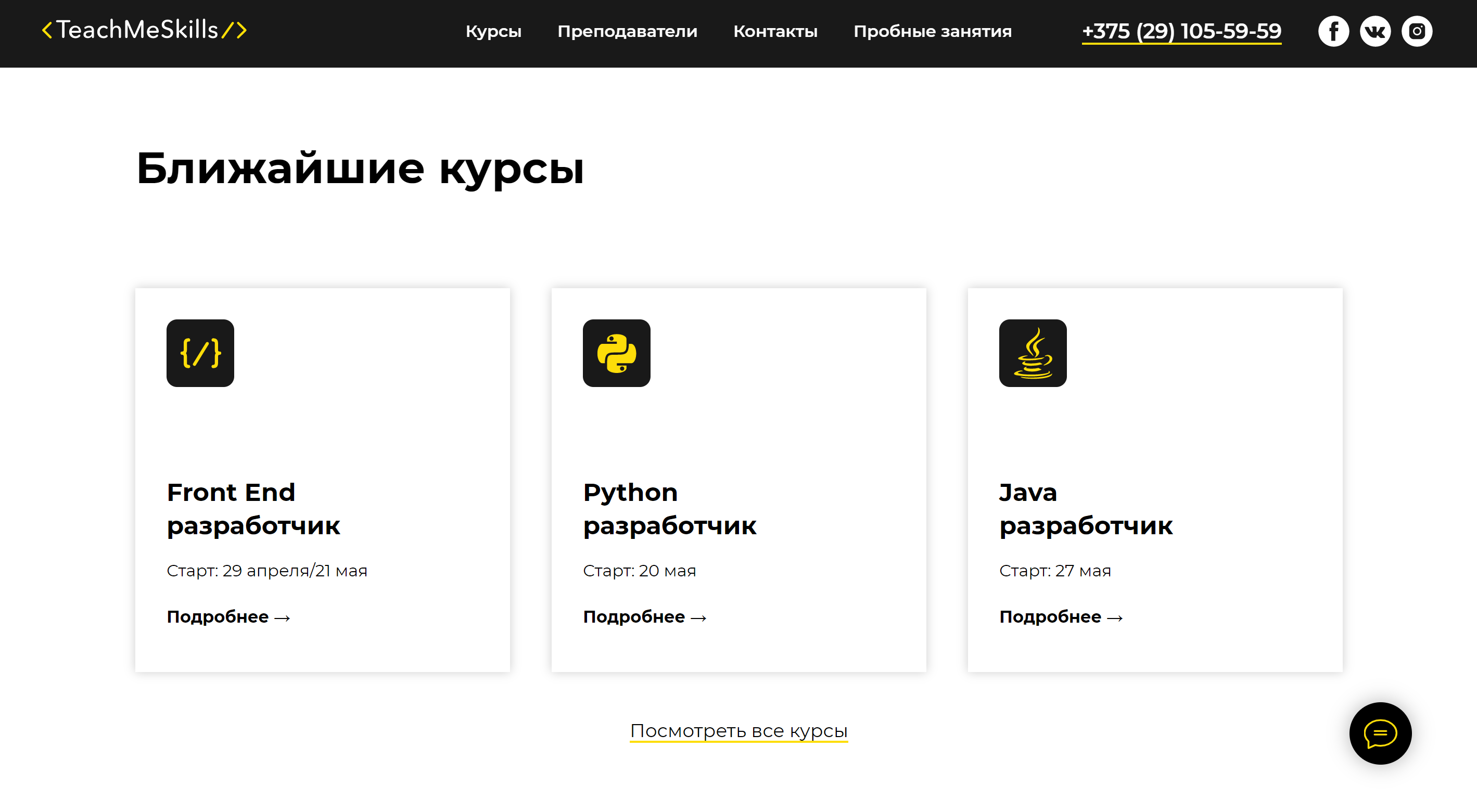


Рисунок 1.2 – Главная страница TeachMeSkills

Сайт обладает динамическими стилями и информативным слайдером на главной странице.

*BitDegree* [3] предлагает массу бесплатных курсов, которые варьируются от программирования до разработки игр.

Они предлагают стандартные онлайн-курсы программирования и курсы в игровой форме. Изученные курсы помогают достичь успеха в обучении и наладить взаимодействие в процессе. Всё, что вам нужно сделать, это выбрать свой язык и начать обучение. Главная страница сайта представлена на рисунке 1.3.

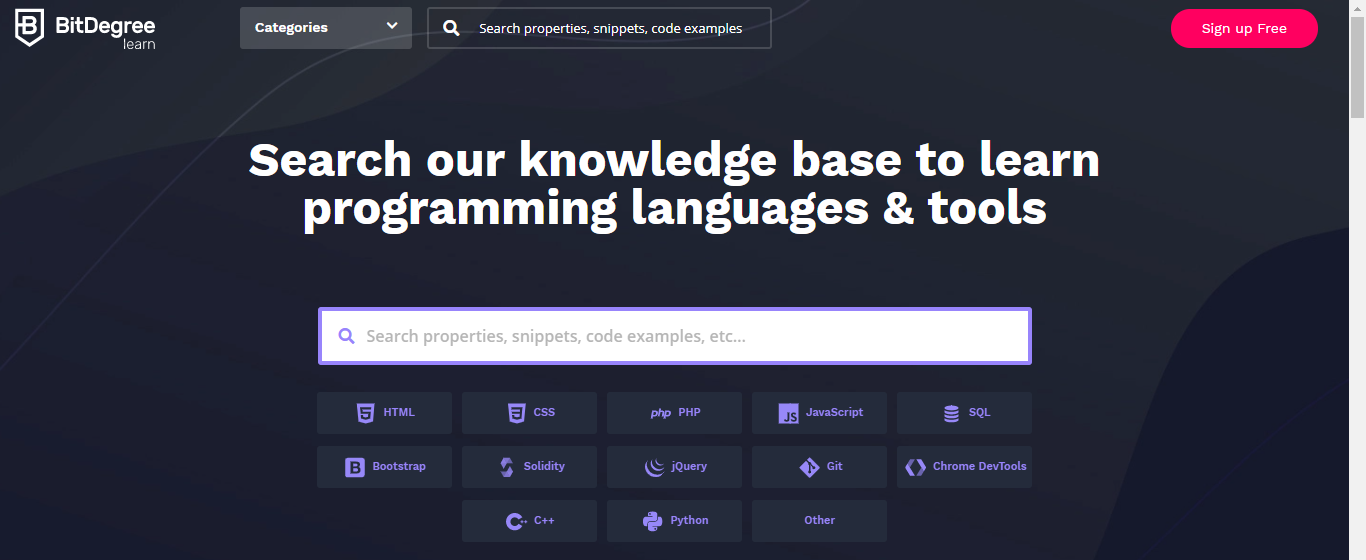


Рисунок 1.3 – Главная страница BitDegree

В BitDegree существуют прозрачные вознаграждения и система достижений, которые помогают компаниям подбирать технически талантливых специалистов и оценивать успешность курса.

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

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

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, тренерах и группах курсов 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 MongoDB. Она имеет открытый исходный код, не требует описания схемы таблиц. Также MongoDB является NoSQL базой данных, т.е. не реляционной базой данных [6].

Mongoose [7] представляет специальную ODM-библиотеку (Object Data Modelling) для работы с MongoDB, которая позволяет определять объекты со строго-типизированной схемой, соответствующей документу MongoDB. Это библиотека, часто используемая в приложении Node.js с базой данных MongoDB.

Для представлений в приложении используется движок EJS. Данный движок использует синтаксис, который позаимствовал из Web Forms на платформе ASP.NET, который разрабатывался в компании Microsoft. EJS будет использоваться как шаблонизатор – программа, выполняющая генерацию представления на основе готового шаблона. Для стилизации приложения использует Bootstrap 4.

Для получения данных с сервера и заполнения ими страниц разрабатываемого приложения используются AJAX-запросы. AJAX является подходом для интерфейсов веб-приложений при котором осуществляется фоновый обмен данными между сервером и клиентом. При использовании асинхронного Javascript страница не перезагружается при обновлении данных.

Также в приложении использовано расширение протокола HTTP – HTTPS [8]. Оно применено для повышения безопасности передачи данных. Передача данных будет происходить поверх криптографического протокола SSL. Он использует асимметричную криптографию для аутентификации ключей обмена, симметричное шифрование для сохранения конфиденциальности, коды аутентификации сообщений для целостности сообщений.

На стороне клиента используется jQuery – быстрая, небольшая и многофункциональная библиотека JavaScript. Она позволяет делать такие вещи, как обход и манипулирование HTML-документами, обработка событий, анимация, намного проще с помощью простого в использовании API, который работает во множестве браузеров.

Протокол WebSocket тоже будет реализован в приложении. WebSocket [9] является абстракцией над протоколом TCP и используется для обмена данными между клиентом и сервером в режиме реального времени.

1. **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.

## **2.1 Проектирование модели базы данных**

В данной курсовой работе для проектирования базы данных используется система управления базами данных MongoDB. Данная СУБД 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 документно-ориентированную модель данных, благодаря чему MongoDB работает быстрее, обладает лучшей масштабируемостью, ее легче использовать.

Если в традиционном мире SQL есть таблицы, то в мире MongoDB есть коллекции. И если в реляционных БД таблицы хранят однотипные жестко структурированные объекты, то в коллекции могут содержать самые разные объекты, имеющие различную структуру и различный набор свойств. Аналогией строк РБД в MongoDB являются документы. В отличие от строк документы могут хранить сложную по структуре информацию. Документ можно представить, как хранилище ключей и значений.

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

Коллекция *Admin* хранит данные об администраторе (ключи: username, hash, salt), с помощью которых он может залогониться в систему.

Коллекция *coursegroups* хранит данные о группах курсов (ключи: title, coverImage).

Коллекция *courses* содержит информацию о курсах, предоставляемых компанией (ключи: title, trainer, info, type, likeCount, createdAt, group).

Коллекция *trainers* предназначена для информации о тренерах (ключи: info, user, department, position).

Коллекция *users* хранит данные о сотрудниках компании (ключи: firstName, lastName, email, photoImage, photoImageType).

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

Поскольку в проекте мы используем архитектуру MVC, то проект будет разделён на три части: Model, View, Controller. Структура программного средства представлена на рисунке 2.1.

Как мы можем увидеть в проекте имеется три главных директория.

Директорий *models* содержит в себе файлы, которые описывают данные, а также логику управления этими данными.

Директорий *views* предназначен для хранения файлов представлений, там находится вся визуальная часть приложения.

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

Также есть дополнительный директорий *public*, в котором содержатся файлы css стилей и js файлы с AJAX запросами для клиентской части приложения.

Директорий *certs* с сертификатами, которые нужны для реализации HTTPS.

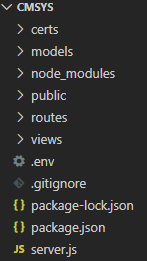


Рисунок 2.1 – Структура программного средства

1. **Разработка программного средства**

## **3.1 Реализация модели базы данных**

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

В процессе создания коллекций нужно также их связать. В листинге 3.1 продемонстрирована связь между моделью Course и моделями Trainer и CourseGroup.

**Листинг 3.1 – Связь между коллекциями**

trainer: {

        type: mongoose.Schema.Types.ObjectId,

        required: true,

        ref: 'Trainer'

    },

group: {

        type: mongoose.Schema.Types.ObjectId,

        required: true,

        ref: 'CourseGroup'

    }

База данных к приложению подключается так, как показано в листинге 3.2. Также здесь представлено подключение модуля mongoose и обработка ошибки подключения к базе данных.

**Листинг 3.2 – Подключение базы данных**

*const* mongoose = require('mongoose')

mongoose.connect(process.env.DATABASE\_URL, { useNewUrlParser: true, useUnifiedTopology: true });

*const* db = mongoose.connection;

db.on('error', *err* => *console*.error(err));

db.on('open', () *=>* *console*.log('Connected to Mongoose'));

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('photoImagePath').get(function() {

    if (this.photoImage != null && this.photoImageType != null) {

return `data:${this.photoImageType};charset=utf-8;base64,

${this.photoImage.toString('base64')}`;

    }

});

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**

*const* userRouter = require('./routes/users');

*const* indexRouter = require('./routes/index');

*const* courseRouter = require('./routes/courses');

*const* trainerRouter = require('./routes/trainers');

*const* courseGroupRouter = require('./routes/courseGroups');

*const* genresRouter = require('./routes/genre');

app.use('/', indexRouter);

app.use('/users', userRouter);

app.use('/courses', courseRouter);

app.use('/trainers', trainerRouter);

app.use('/courseGroups', courseGroupRouter);

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.

Кроме того, к приложению нужно было подключить HTTPS. Для этого сначала были созданы сертификаты с использованием криптографической библиотеки OpenSSL, затем данный сертификат был дополнительно зарегистрирован. После этого был загружен модуль https и с помощью его и сертификатов был создан сервер, использующий HTTPS.

**Листинг 3.6 – Подключение HTTPS**

*const*https = require('https');

*const* fs = require('fs');

*const* httpsOptions = {

    key: fs.readFileSync('certs/LAB.key'),

    cert: fs.readFileSync('certs/LAB.crt')

};

*const* server = https.createServer(httpsOptions, app);

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('/');

    }

});

Следующий код позволяет залогиниться администратору приложения для доступа к страницам добавления и изменения данных. Для аутентификации используется модуль *passport*, а также *passport-local-mongoose*. Подключение плагина *passport-local-mongoose* представлено в листинге 3.10.

**Листинг 3.10 – Подключение плагина для аутентификации в модели Admin**

*const* passportLocalMongoose = require('passport-local-mongoose');

adminSchema.plugin(passportLocalMongoose);

В листинге 3.11 показано подключение модуля *passport* в главном файле приложения *server.js*.

**Листинг 3.11 – Подключение веб-сокетов**

*const* passport = require('passport');

*const* cookieParser = require('cookie-parser');

*const* expressSession = require('express-session');

app.use(cookieParser());

app.use(expressSession({

    secret: 'secret',

    saveUninitialized: false,

    resave: false

}));

app.use(passport.initialize());

app.use(passport.session());

**Продолжение листинга 3.11**

passport.use(Admin.createStrategy());

passport.serializeUser(Admin.serializeUser());

passport.deserializeUser(Admin.deserializeUser());

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

    res.render('login/index');

});

app.post('/login', (*req, res, next*) => {

    passport.authenticate('local',

        (*err, user, info*) => {

            if (err) {

                return next(err);

            }

            if (!user) {

                return res.redirect('/login?info=' + info);

            }

            req.logIn(user, function(*err*) {

                if (err) {

                    return next(err);

                }

                return res.redirect('/');

            });

        })(*req, res, next*);

});

В листинге 3.12 показано подключение middleware *connect-ensure-login* для обеспечения доступа к ресурсу только аутентифицированным пользователям.

**Листинг 3.12 – Подключение промежуточного слоя connect-ensure-login**

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

    try {

        res.render('courses/admin');

    } catch {

        res.redirect('/');

    }

});

## **3.3 Проектирование структуры клиента**

На стороне клиента используются представления, которые динамически конфигурируются с помощью движка EJS, который также является шаблонизатором. Он был подключён с помощью модуля express-ejs-layouts.

**Листинг 3.13 – Подключение шаблонизатора**

*const* expressLayouts = require('express-ejs-layouts');

app.set('view engine', 'ejs');

app.set('views', \_\_dirname + '/views');

В приложении есть отдельные директории с представлениями для каждого роутера. Также представлены отдельно частичные представления, которые вставляются в другие и представление с заголовком общее для всех страниц.

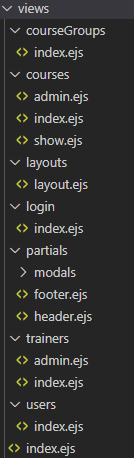


Рисунок 3.13 – Содержимое директория views

Также есть директорий public, который хранит в себе дополнительный функционал для клиентской части. В нём расположены клиентские стили и стили bootstrap, файлы клиентской части кода на JavaScript, в которых представлен код AJAX запросов на получение данных из базы данных.

Описанный директорий показан на рисунке 3.14.

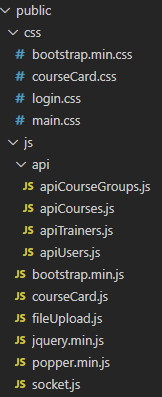


Рисунок 3.14 – Содержимое директория public

В листинге 3.15 представлен пример динамического формирования содержимого страницы. Контроллер передаёт в представление объект courses, а представление с помощью цикла forEach вставляет данные для заполнения карт-курсов.

**Листинг 3.15 – Пример динамического формирования содержимого страницы**

<% courses.forEach(*course* => { %>

    <div class="card">

   <div class="card-img" style="background-image: url('<%= course.group. coverImagePath %>');">

            <div class="overlay">

                <div class="overlay-content">

                    <a href="/courses/<%= course.id %>">View Course</a>

                </div>

            </div>

        </div>

        <div class="card-content">

            <a href="/courses/<%= course.id %>">

                <h2>

                    <%= course.title %>

                </h2>

                <p>

                    <%= course.group.title %>,

                    <%= course.type %>

                </p>

            </a>

        </div>

    </div>

<% }) %>

AJAX – технология обращения к серверу без перезагрузки страницы. За счёт этого уменьшается время отклика веб-приложения.

Листинг 3.16 показывает, как обрабатывается POST запрос ‘/courseGroups’ на стороне сервера на создание новой группы курсов.

**Листинг 3.16 – Обработка ajax-запроса на стороне сервера**

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

    if (!req.body) res.status(400);

*const* courseGroup = new CourseGroup({

        title: req.body.title

    });

    saveCover(courseGroup, req.body.cover);

    try {

        await courseGroup.save();

        res.send(courseGroup);

    } catch {

        res.status(400);

    }

});

Такой запрос можно отправить с помощью AJAX (листинг 3.17).

**Листинг 3.17 – Обработка ajax-запроса на стороне клиента**

function addCourseGroup(*event*) {

    event.preventDefault();

    $.ajax({

        type: 'POST',

        url: $(this).attr('action'),

        contentType: "application/json; charset=utf-8",

        data: JSON.stringify({

            title: $('#addTitle').val(),

            cover: $('.filepond--data input').val()

        }),

        success: function(*data*) {

            $('#addModal').modal('hide');

            getCourseGroups();

            $('#addCourseGroup')[0].reset();

        }

    });

};

# **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.

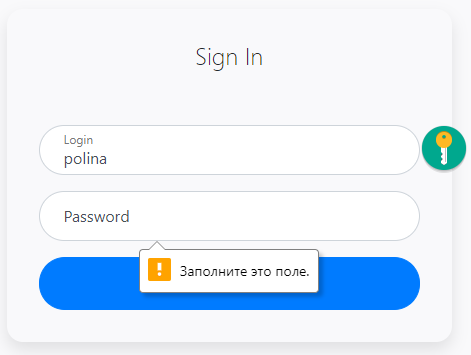


Рисунок 4.1 – Обработка ввода пустых данных

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.

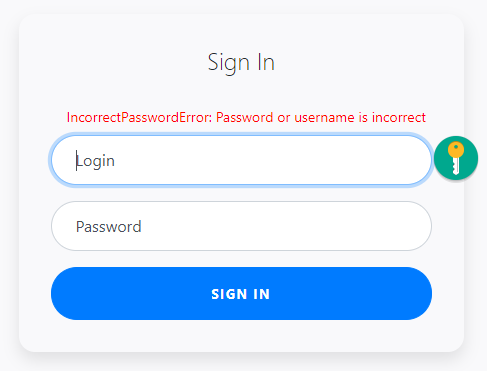


Рисунок 4.2 – Обработка некорректного логина или пароля

## **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.

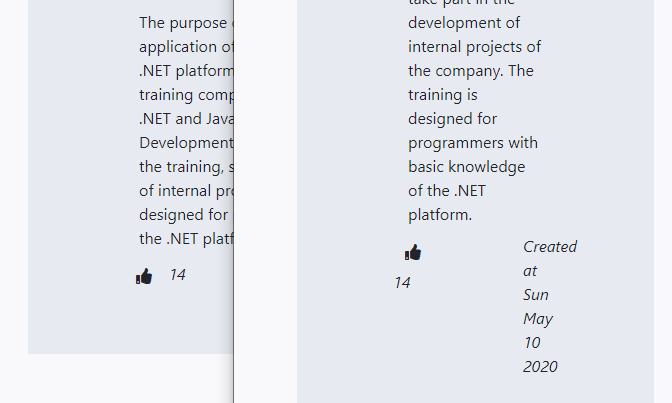


Рисунок 4.3 – Тестирование WebSockets

## **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.

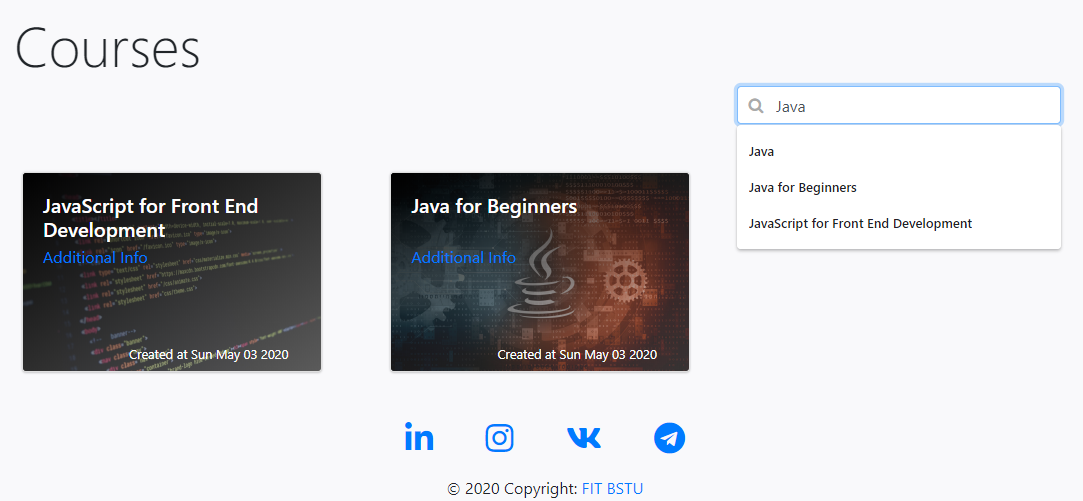


Рисунок 4.4 – Тестироние поиска по названиям курсов

# **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.

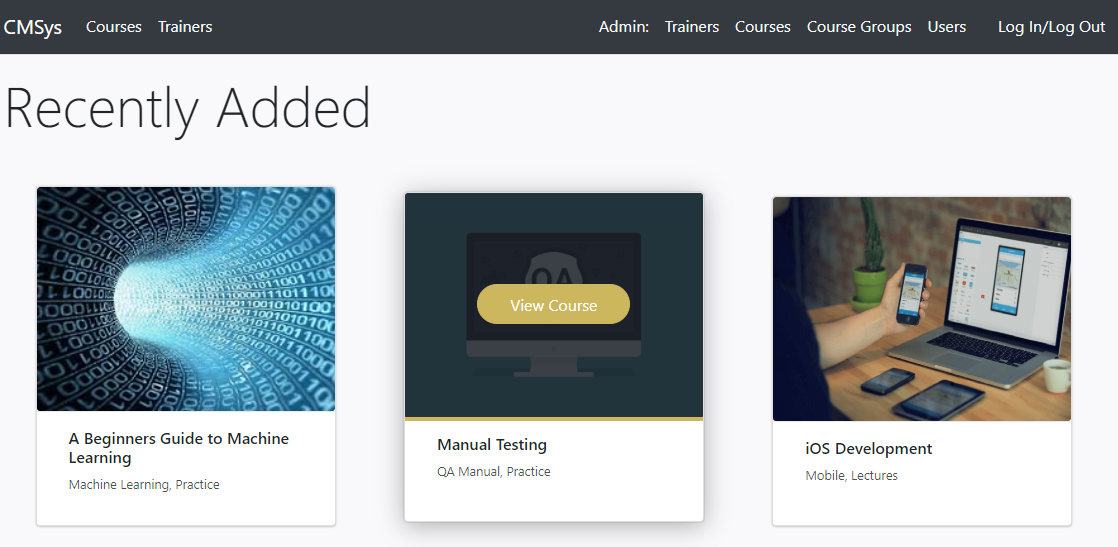


Рисунок 5.1 – Начальная страница web-приложения

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.

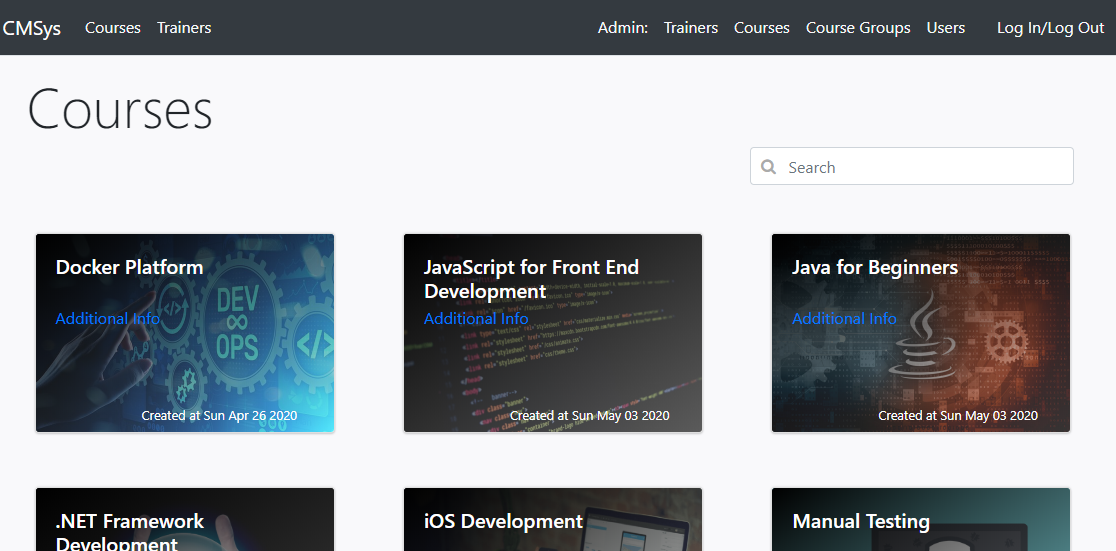


Рисунок 5.2 – Страница со курсами

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).

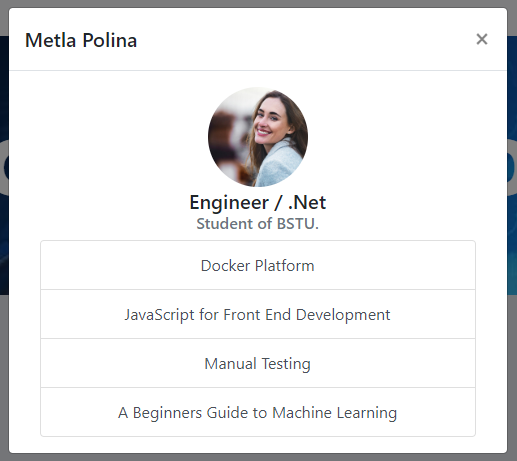


Рисунок 5.4 – Модальное окно с информацией о тренере

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.

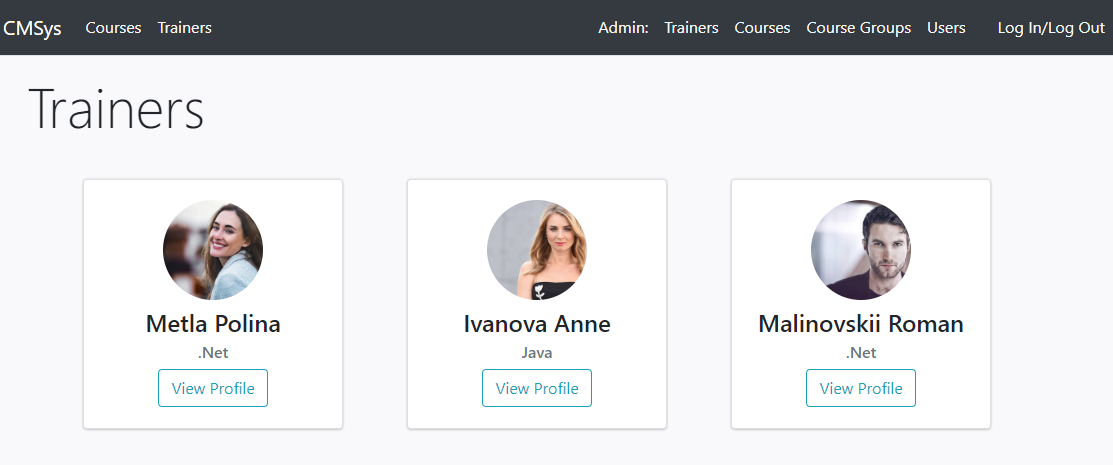


Рисунок 5.5 – Страница тренеров

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.

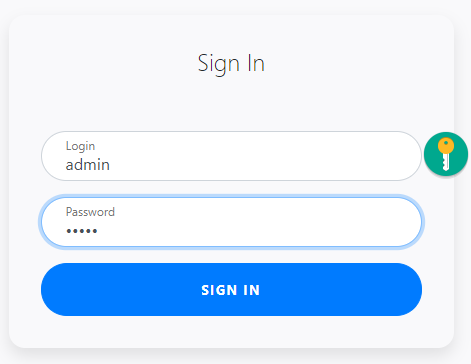


Рисунок 5.6 – Фрагмент экрана с формой для авторизации админа

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.

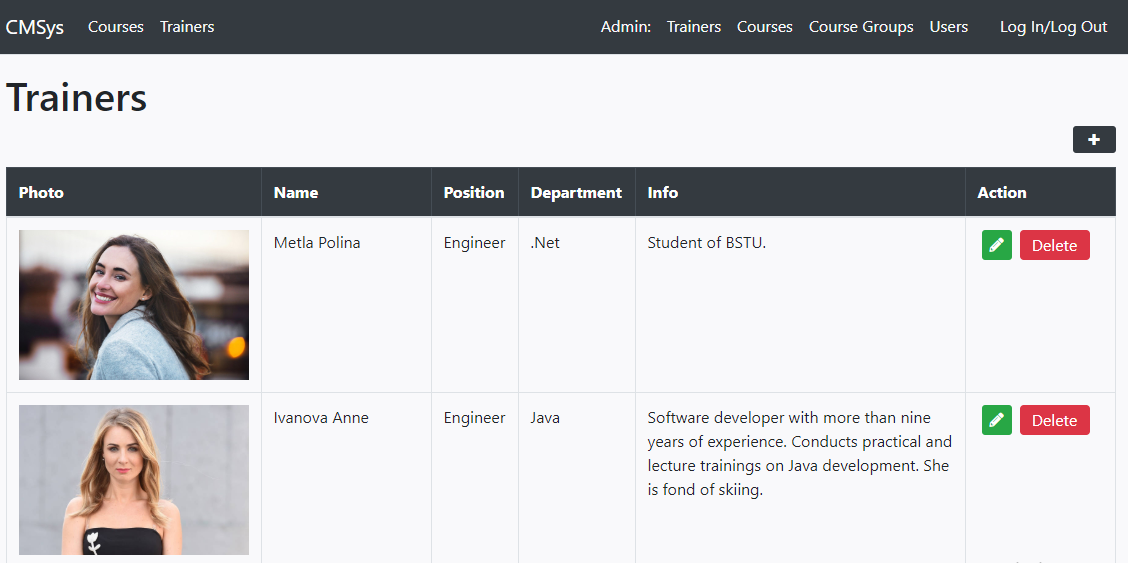


Рисунок 5.7 – Страница тренеров для администратора

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

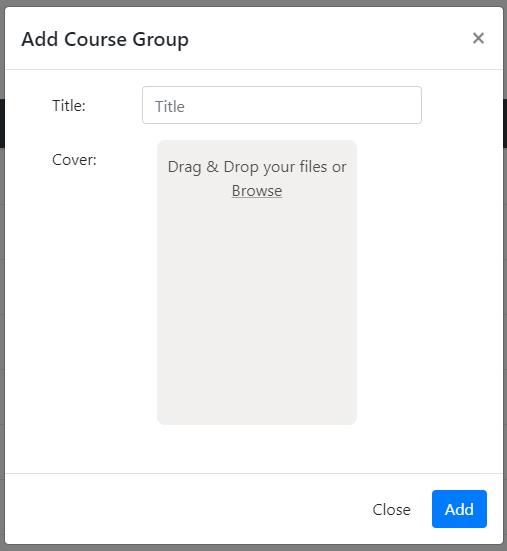


Рисунок 5.8 – Модальное окно добавления группы курсов

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

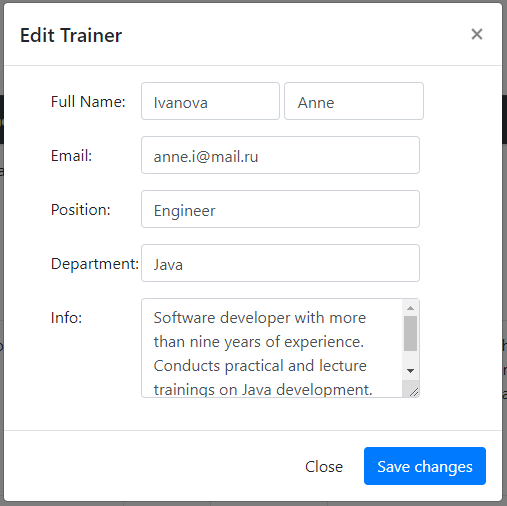


Рисунок 5.9 – Модальное окно для редактирования информации о тренере

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.

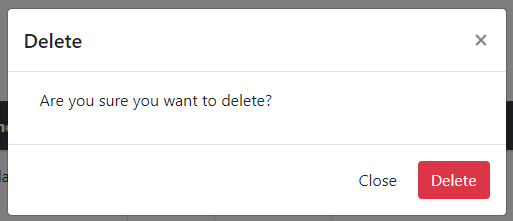


Рисунок 5.10 – Модальное окно предупреждения об удалении объекта

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:

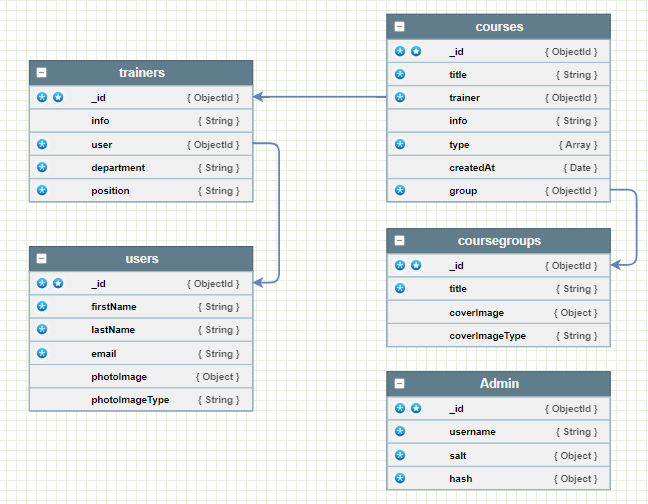
* просмотр информации о курсах и тренерах для все пользователей;
* просмотр информации о курсах, тренерах, группах курсов, а также сотрудниках компании для администратора;
* добавление курсов, тренеров и групп курсов;
* изменение курсов, тренеров и групп курсов;
* удаление курсов, тренеров и групп курсов;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