

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет «Медиакоммуникации и мультимедийные технологии»

Кафедра «Медиатехнологии»

|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
| И. о. зав. кафедрой | | | «МТ» |
| \_\_\_\_\_\_\_\_\_\_\_\_ | | О. В. Витченко | |
| подпись | |  | |
| «\_\_\_\_\_» | \_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г. | | |

**ВЫПУСКНАЯ** **КВАЛИФИКАЦИОННАЯ РАБОТА**

**(бакалаврская работа)**

Тема: «РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ ПОДДЕРЖКИ ПРОЦЕССА ОБУЧЕНИЯ НА ЦИФРОВОЙ КАФЕДРЕ ДГТУ»

Направление подготовки 09.03.03 Прикладная информатика

Профиль Разработка игр и прикладных программ

Обозначение ВКР 09.03.03.530000.000 Группа МКИС42

Обучающийся М. М. Гапонов

подпись, дата

Руководитель ВКР доцент, И.С. Трубчик

подпись, дата

Консультанты по разделам:

Безопасность и экологичность

проекта доцент, Черемисин А.И.

подпись, дата

Нормоконтроль ст. преп., О.А. Попова

подпись, дата

Ростов-на-Дону

2024



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
 «ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет «Медиакоммуникации и мультимедийные технологии»

Кафедра «Медиатехнологии»

|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
| И. о. зав. кафедрой | | | «МТ» |
| \_\_\_\_\_\_\_\_\_\_\_\_ | | О. В. Витченко | |
| подпись | |  | |
| «\_\_\_» | \_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г. | | |

**ЗАДАНИЕ**

на выполнение выпускной квалификационной работы

Тема: «РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ ПОДДЕРЖКИ ПРОЦЕССА ОБУЧЕНИЯ НА ЦИФРОВОЙ КАФЕДРЕ ДГТУ»

Обучающийся Гапонов Михаил Максимович

Обозначение ВКР 09.03.03.530000.000 Группа МКИС42

Тема утверждена приказом по ДГТУ от «23» января 2024 г. №266-ЛС-О

Срок представления ВКР к защите «21» июня 2024 г.

Исходные данные для выполнения выпускной квалификационной работы:

Персональный компьютер на операционной системе Windows. Программы для разработки – интегрированная среда разработки IntelliJ IDEA, VPS, Postman. Профильная и специализированная литература. Интернет-ресурсы. Материалы, полученные на преддипломной практике.

Содержание выпускной квалификационной работы

Введение:

В 21 веке студенты имеют доступ к множеству образовательных платформ, но управление информацией остается вызовом. Расписание занятий, материалы курсов и оценки разбросаны по разным системам, что усложняет их организацию и отслеживание академического прогресса. Цель этого исследования – создать единую платформу, где будет упорядочена вся необходимая информация, объединяя расписание, оценки, материалы и коммуникацию с преподавателями.

Наименование и краткое содержание разделов:

1 Постановка задач и исследование предметной области.

2 Обоснование выбора инструментов и технологий для реализации поставленной задачи

В этом разделе проведено исследование предметной области, рассмотрены ключевые понятия о web-приложениях для Android, также описание технического описания от Цифровой кафедры.

3 Разработка backend приложения.

Раздел содержать описание разработки REST архитектуры приложение, базы данных, а также основных модулей приложений.

4 Безопасность и экологичность работы.

В этой главе поднимаются экологические аспекты утилизации микропроцессорной техники, проводятся расчеты искусственного освещения помещения.

Заключение:

В результаты работы проведен анализ конкурентов на рынке, технологий необходимых для разработки backend web-приложение на платформе Android. Была разработана архитектура приложения и реализована практическая часть.

Перечень графического и иллюстративного материалов:

1. Рисунки – 5.

2. Презентация Power Point – 10 слайдов.

Руководитель ВКР \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ доцент, И.С. Трубчик

подпись, дата

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М. М. Гапонов

подпись, дата

**Аннотация**

Предмет исследования – разработка backend приложения на Android для поддержания учебного процесса студентов для Цифровой кафедры во фреймворке Spring.

Объект исследования – различные платформы разработанные для организации обучения.

Целью является разработка backend приложения на Android для поддержания учебного процесса студентов для Цифровой кафедры. Также было необходимо исследовать предметную область, поставить задачи разработки. Выбрать необходимые средства и инструменты для реализации проекта. Разработать backend web-приложения на Android, на архитектуре REST.

Работа предназначена помочь работникам и преподавателям Цифровой кафедры с взаимодействием с студентами и абитуриентами данного подразделения ДГТУ.

Выпускная квалификационная работа состоит из 71 листов, содержит 5 изображений, 3 таблицы, 23 листинга, 11 использованных источников.

**Annotation**

The subject of the research is the development of a server application on Android to support the educational process of students for the Digital Department in the Spring platform.

The object of the research – various platforms developed for the organization of training.

The goal is to develop a server application for Android to support the educational process of students for the Digital Department. It was also necessary to become familiar with the subject area and set development tasks. Select the necessary tools and tools to implement the project. Develop backend web applications on Android using REST architecture.

Their work helps teachers of digital departments with interaction with students and applicants of the Digital Department. The final qualifying work consists of 71 pages, contains 5 images, 3 tables, 23 code listings and 13 references.

**Содержание**

[Введение 6](#_Toc168685784)

[1 Постановка задач и исследование предметной области 8](#_Toc168685785)

[1.1 Исследование предметной области 8](#_Toc168685786)

[1.2 Постановка задачи 9](#_Toc168685787)

[2 Обоснование выбора инструментов и технологий для реализации поставленной задачи 12](#_Toc168685788)

[2.1 Java Spring 12](#_Toc168685789)

[2.2 Intellij IDEA 13](#_Toc168685790)

[2.3 Postman 14](#_Toc168685791)

[2.4 MySql и Hibernate 14](#_Toc168685792)

[2.5 VPS сервер 17](#_Toc168685793)

[3 Разработка backend приложения 17](#_Toc168685794)

[3.1 Разработка архитектуры приложения 17](#_Toc168685795)

[3.2 Структура базы данных 20](#_Toc168685796)

[3.3 Создание и настройка проекта, подключение базы данных 23](#_Toc168685797)

[3.4 Регистрация и авторизация 27](#_Toc168685798)

[3.5 Разработка курсов и лекций 31](#_Toc168685799)

[3.6 Разработка чат бота 34](#_Toc168685800)

[3.7 Разработка технической поддержки 35](#_Toc168685801)

[4 Безопасность и экологичность работы 35](#_Toc168685802)

[4.1 Утилизация микропроцессоров 36](#_Toc168685803)

[4.2 Расчет системы искусственного освещения 39](#_Toc168685804)

[4.3 Выводы по главе 44](#_Toc168685805)

[Заключение 45](#_Toc168685806)

[Перечень использованных информационных ресурсов 48](#_Toc168685807)

[Приложение А 49](#_Toc168685808)

# Введение

В 21-м веке у студентов появилось огромное количество вариантов обучения, воплощенных во множестве образовательных платформ. В ходе своей учебы в вузе студенты часто сталкиваются с вызовом управления информацией. Расписание, материалы курсов, ведомости и средства связи с преподавателями разбросаны по разным платформам и ресурсам. Эта фрагментация информации может стать причиной существенных трудностей для студентов.

1. Расписание и материалы: информация о расписании занятий, материалах курсов и домашних заданиях может быть разделена между несколькими онлайн-платформами, электронными ресурсами, мессенджерами. Это создает сложности в организации, и студенты могут упустить важную информацию.
2. Ведомости и оценки: получение информации о своих успехах также требует перехода между различными платформами. Ведомости, оценки и обратная связь могут храниться в разных системах, усложняя для студентов отслеживание своего академического прогресса.
3. Средства коммуникации: для связи с преподавателями студентам приходится использовать электронную почту, различные мессенджеры, платформы для обсуждения материалов и другие инструменты. Это может привести к путанице в общении.

Чтобы реализовать решение всех этих задач в одном приложении была поставлена задача разработать единую платформу для студентов Цифровой кафедры (ЦК) ДГТУ, на которой вся информация: расписание, ведомости, материалы к практическим занятиям и лекциям, информация о деятельности ВУЗа, важные объявления – будет структурирована и упорядочена. Другими словами, разрабатываемый проект должен представлять собой одновременно соцсеть, базу данных для курсов, а также форум для помощи студентам ДГТУ в освоении курсов ЦК ДГТУ.

Проект «Цифровая кафедра» реализуется в ДГТУ, который является участником программы «Приоритет-2030» в рамках федерального проекта «Развитие кадрового потенциала ИТ-отрасли» национальной программы «Цифровая экономика Российской Федерации». В соответствии с данной программой, обучение новым цифровым компетенциям проводится параллельно с основным образованием в университете. В ходе обучения, помимо теоретических основ, студенты проходят практику на предприятиях приоритетных отраслей экономики Донского региона. По итогам обучения и презентации цифровых решений (проектов) студенты получают Диплом о профессиональной переподготовке (256 часов).

На сегодняшний день для решения таких задач в ДГТУ используется ресурсы, такие как donstu.ru и ДО.СКИФ.

Мой ДГТУ (https://my.e.donstu.ru/ ) – это основной портал для студентов и преподавателей ДГТУ. На этом сайте можно найти расписание занятий, учебные материалы, а также информацию о курсах и программах. Портал предоставляет доступ к электронным библиотекам, методическим материалам и различным административным услугам, таким как подача заявлений или получение справок. Сайт также позволяет следить за новостями университета, мероприятиями и различными академическими событиями.

ДО.СКИФ (https://do.skif.donstu.ru/) – это специализированная платформа для тестирования и оценки знаний студентов. На этом сайте преподаватели могут создавать и проводить тесты, экзамены и контрольные работы в онлайн-режиме. Студенты, в свою очередь, имеют возможность проходить тестирования, получать оценки и анализировать свои результаты.

Однако, у ЦК ДГТУ есть большой объем специальных задач, обусловленных правилами национальной программы. Таким образом, создание единой платформы для студентов, на которой вся информация будет структурирована и упорядочена, является актуальной задачей.

## Постановка задачи и исследование предметной области

### Исследование предметной области

Для поддержания процесса обучения студентов используется множество различных приложений, сервисов и сайтов. Они различны по своему функционалу, удобству интерфейса и доступности. Рассмотрим самые популярные платформы на данный момент.

Moodle – это один из самых популярных систем управления обучением. Ключевое ее преимущество заключается в том, что система может быть настроена для каждого вуза по-разному. Платформу можно использовать, и изменять под свои нужны. Благодаря этому, можно настраивать свои курсы, задания, прикреплять текстовые, видео и аудио материалы, презентации и интерактивные уроки. Преподаватели могут отслеживать прогрессию студентов и ставить им оценки. Система работает, как web-сайт, десктопное приложение и мобильное приложение. Слабая сторона Moodle – неудобный интерфейс.

1С:Образование – это программное обеспеченье для организации и поддержки учебного процесса среди муниципальных образовательных организаций. Система позволяет эффективно управлять всеми аспектами деятельности учебного образовательного процесса. Благодаря нее преподаватели могут вести учет учебных данных, контактных информаций об учащихся, а также контролировать выполнение заданий. В 1С можно создавать курсы, занятия и расписания, назначать преподавателей и аудитории для академических групп. Из минусов можно выделить очень дорогие тарифы для учреждений, необходимость наличия специалистов для поддержания системы, плохой интерфейс.

Google Класс – сервис от компании Google, который помогает преподавателям распределять знания и эффективно взаимодействовать с учащимися. Сервис доступен как web-сайт и приложение на телефон. Преподаватель может создавать курсы, прикреплять к ним учебные текстовые, видео и аудио материалы. Так же сервис имеет функцию создания ленты новостей для учащихся. Огромным плюсом является, что Google класс полностью бесплатен, и не нуждается в дополнительной поддержке и сложной настройке. Из этого вытекают его минусы, такие как скудность функционала и невозможность тонкой настройки под различные нужды.,

Анализируя, сильные и слабые стороны ключевых конкурентов, можно выделить преимущества, предварительно согласованные с Цифровой кафедрой, которые будут реализованы в проекте:

* приложение должно быть для мобильного устройства, чтобы обеспечить постоянный доступ к информации;
* преподаватели и администраторы, должны иметь возможность создавать, изменять и удалять курсы и материалы к курсам;
* у преподавателей и администраторов должен быть доступ к базе данных студентов;
* должен удобный адаптивный интерфейс;
* возможность связи студентов и преподавателей;
* отправка push уведомлений для информирования студентов.

#### Постановка задачи

Основная задача выпускной квалификационной работы – создание backend web-приложения на платформе Android для поддержания учебного процесса студентов, а также для упрощения взаимодействия между преподавателями и студентами. Backend должен быть построен на архитектуре Rest.

От Цифровой Кафедры ДГТУ было предложено следующее техническое задание.

#### Регистрация пользователя

Пользователь должен вводить свои данные, такие как:

* электронная почта (обязательное);
* пароль (обязательное, минимум восемь символов);
* фамилия (обязательное);
* имя(обязательное);
* отчество(обязательное);
* группа(обязательное);
* дата рождения (обязательное);
* номер студенческого билета(обязательное).

Затем должна происходить валидация данных:

* проверка уникальности электронной почты;
* проверка соответствия пароля и подтверждения пароля.
* валидация формата электронной почты и номера телефона.
* сохранение данных пользователя в базе данных.

#### Аутентификация

Пользователь должен ввести свои данные:

* электронная почта (обязательное);
* пароль (обязательное);

Затем должна происходить валидация данных:

* проверка наличия пользователя с указанной электронной почтой;
* проверка соответствия введенного пароля хешу пароля в базе данных.

#### Чат-бот

Общие требования:

* чат-бот должен быть доступен на всех страницах приложения;
* интеграция с базой данных для получения ответов на часто задаваемые вопросы;
* возможность расширения базы знаний чат-бота администратором.

Интерфейс чат-бота для пользователя:

* окно чата с возможностью отправки сообщений;
* история чата с предыдущими вопросами и ответами.

Административный интерфейс:

* форма для добавления новых вопросов и ответов в базу знаний;
* просмотр статистики использования чат-бота (часто задаваемые вопросы, количество обращений и т.д.).

Обработка запросов:

* чат-бот анализирует введенный вопрос;
* поиск ответа в базе знаний;
* возвращение ответа пользователю;
* если вопрос не найден в базе знаний, чат-бот предлагает обратиться в поддержку.

#### Курсы и лекции

Управление курсами (только для администратора и преподавателей):

* создание курсов;
* удаление курсов;
* изменение информации курсов;
* добавление лекций к курсам;
* сохранение курса в базе данных;

Управление лекциями (только для администратора и преподавателей):

* добавление новых лекции;
* изменение информации лекции;
* добавление материалов к лекциям;
* привязка лекций к курсам;
* сохранение лекции в базе данных.

#### Техническая поддержка

Пользователь может написать сообщение в тех. поддержку, принцип отправки обращения:

* пользователь вводит электронную почту, тему обращение и текст обращения;
* проверка обязательных полей;
* обращение сохраняется в базе данных;
* отправка уведомления администратору о новом обращении;

Администратор может:

* просматривать все обращения с возможностью фильтрации по статусу (новое, в обработке, закрытое);
* отвечать на обращение;
* закрывать обращение.

#### Управление пользователями (только администратор) и возможности пользователей

Администратору доступен список всех пользователей с возможностью фильтрации по группе.

Администратор может:

* удалять пользователя;
* банить пользователя;
* изменять данные пользователя;
* изменять роль пользователя;
* присваивать курс пользователю;

Возможности пользователя:

* смотреть и изменять информацию о себе;
* просматривать свой курс и лекции;
* скачивать файлы, прикрепленные к лекциям преподавателем;
* обращаться в тех. поддержку;
* пользоваться чат ботом для получения ответов на вопросы.

## Обоснование выбора инструментов и технологий для реализации поставленной задачи

Для создания Web-приложение для мобильных устройств необходимо выбрать инструменты и технологии, такие как язык программирования, фреймворк, среду разработки, хостинг сервера и средства для тестирования. В качестве языка и фреймоворка используются: Java, Spring Framework, среда разработки IntelliJ IDEA. В качестве СУБД используется MySQL, для отслеживания POST и GET запросов будем использовать POSTMAN. Используется VPS с установленным дистрибутивом операционной системы Linux Ubuntu 22.04 LTS.

### Java Spring

Spring Framework — универсальный [фреймворк](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) [с открытым исходным кодом](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) для [Java](https://ru.wikipedia.org/wiki/Java)-платформы. Но Spring — это скорее не один фреймворк, а общее названия для целого ряда небольших фреймворков, каждый из которых выполняет какую-то свою работу. Например, Spring Boot, Spring Security, Spring Data.

Преимущества Spring Framework:

* поддержка разных архитектурных стилей, таких как MVC, RESTful;
* Spring обладает высокой производительностью, благодаря эффективному распределению ресурсов;
* Spring Security обеспечивает мощную и настраиваемую защиту нашего приложения;
* возможность создать как небольшой, так и масштабируемое приложение для продакшена.

Основные компоненты Spring, которые внедрены в наш проект:

* Spring Boot – компонент фреймворка, предназначенный для упрощения создания приложения, он автоматически упаковывает необходимые зависимости, а также настраивает конфигурацию приложения на основе зависимостей. Еще один плюс, что Spring Boot имеет встроенный web-сервер;
* Spring Security – компонент фреймворка, предоставляющий механизмы для аутентификации и авторизации, а также другие возможности обеспечения безопасности для приложений. При помощи него можно разрешать и запрещать доступ к тем или иным модулям приложения;
* Spring Web – фреймворк, который предоставляет основу для создания веб-сайтов или RESTful сервисов, он поддерживает контроллеры и REST контроллеры. Благодаря Spring Web, приложение может обрабатывать GET и POST запросы;
* Spring Data Jpa — фреймворк, который осуществляет взаимодействие приложения с сущностями базы данных, организации их в репозитории, извлечение, сохранения или изменения данных.

### IntelliJ IDEA

IntelliJ IDEA – интегрированная среда разработки программного обеспечения для многих языков программирования, в частности Java, JavaScript, Python, разработанная компанией [JetBrains](https://ru.wikipedia.org/wiki/JetBrains).

Основные преимущества IntelliJ IDEA:

* поддержка языков, таких как Java, Kotlin, Groovy, а также языков разметки и шаблонов HTML, CSS, JavaScript, FreeMarker;
* безопасный рефакторинг кода;
* умное автодополнение кода;
* поддержка фреймворков Spring и Hibernate;
* встроенная система контроля версий GIT;
* интеграция сборщика Gradle.

### Postman

Postman — это HTTP-клиент для тестирования API. HTTP-клиенты тестируют отправку запросов с клиента на сервер и получение ответа от сервера.

С помощью Postman тестировщик может:

* составлять и отправлять HTTP-запросы к API;
* создавать коллекции (набор последовательных запросов) и папки запросов для сокращения времени тестирования;
* менять параметры запросов (например, ключи авторизации и URL);
* менять окружения для запросов (например, на тестовом стенде, локально или на сервере);
* добавлять при вызове API контрольные точки (фиксацию момента передачи данных);
* проводить автоматизированное тестирование API по коллекции запросов с помощью Collection Runner.

### MySql и Hibernate

Web-приложение будет работать с большим количеством данных. Требуется где-то хранить данные, изменять, удалять. Для с этими задачами реляционная база данных, чтобы ей управлять понадобится система управления базой данных или СУБД. Существует много различных СУБД, например: MySQL, PostgresSQL, MondoDB, Microsoft SQL SERVER. Для нашего приложения подойдет MySQL, она имеет клиент-серверную архитектуру. Это означает, что база данных хранится в одном источнике – на сервере. А связываться с ней могут клиенты – сторонние устройства. Клиенты отправляют запросы к БД, а потом получают от сервера информацию.

Основные преимущества MySQL:

* MySQL работает на Windows, Linux FreeBSD, это нам позволить развернуть СУБД на сервере;
* она полностью бесплатно и с открытым исходным кодом;
* встроенный сервер, на котором запускается СУБД;
* MySQL предлагает пользователям высокий уровень безопасности. В MySQL есть встроенные инструменты безопасности, которые поддерживают управление пользователями и их привилегиями;
* MySQL поддерживает различные типы данных, такие как текстовые, числовые, временные и пространственные;
* MySQL позволяет настраивать разные уровни доступа к данным;
* MySQL обеспечивает высокий уровень безопасности при хранении и передаче данных. У системы продвинутая система контроля прав для разных пользователей, есть шифрование и верификация.

У данной СУБД, также есть и минусы. Сниженная производительность в больших проектах, и непредсказуемое поведение при большом количестве запросов к базе, и недостаток функций по сравнению с другими СУБД, но для наших целей она подходит.

При упоминании баз данных и систем управления базами данных, не могу не обратить внимание на Hibernate. Это фреймворк для языка Java, предназначенный для работы с базами данных. Он реализует объектно-реляционную модель – технологию, которая связывает программные сущности и соответствующие записи в базе.

Объектно-реляционная модель, или ORM, позволяет создать программную «виртуальную» базу данных из объектов. Объекты описываются на языках программирования с применением принципов ООП. Java Hibernate – популярное воплощение этой модели.

Hibernate построен на спецификации JPA 2.1 – наборе правил, который описывает взаимодействие программных объектов с записями в базах данных. JPA поясняет, как управлять сохранением данных из кода на Java в базу. Но сама по себе спецификация – только теоретические правила, а в «чистой» Java ее реализации нет. Hibernate – одна из самых популярных реализаций JPA на рынке.

На практике это выглядит так, прописываются аннотации @Entity, чтобы указать Hibernate, что класс является сущностью, которая будет отображена в базу данных. Аннотация @Table нужна для установки имени таблицы. Далее прописываются аннотации @Column для полей класса, чтобы фреймворк создал необходимые столбцы в базе данных. Аннотация @Id указывает на то, что поле является первичным ключом. После запуска программы в базе данных создается таблица. После чего мы можем взаимодействовать с данным из таблицы, через объекты класса. Принцип создание таблицы при помощи Hibernate показан в листинге 1.

Листинг 1 –Код работы Hibernate

@Entity

@Table(name=«course»)

@AllArgsConstructor

@NoArgsConstructor

@Data

public class Course {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

@Column(name=«id»)

private Long id;

@Column(name=«title»)

private String title;

@Column(name=«description», columnDefinition = «text»)

private String description;

@JsonManagedReference

@OneToMany(cascade = CascadeType.REFRESH, fetch = FetchType.LAZY,

mappedBy = «course»)

private List<User> users = new ArrayList<>();

@JsonManagedReference

@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,

mappedBy = «course»)

private List<Lecture> lectures = n

### VPS сервер

VPS – это виртуальный выделенный сервер, который хостинг провайдер предоставляет в аренду пользователям. Грубо говоря у провайдера, есть комнаты с большим количеством мощных компьютеров – серверов, при аренде пользователям выделяют небольшую часть, с которыми они могут совершать различные манипуляции. На VPS, арендованном для проекта, установлен Ubuntu 22.04 и Nginx.

Nginx – это программное обеспечение с открытым исходным кодом для создания легкого и мощного веб-сервера. Также его используют в качестве почтового или обратного прокси-сервера. То есть Nginx перенаправляет клиентские запросы к бекэнд-серверу проекта, что позволяет распределить нагрузку, кэшировать ответы.

## Разработка backend приложения

### Разработка архитектуры приложения

Для реализации web-приложения на платформе Android, подходит архитектура RESTful, так как сервер не может возвращать представления клиенту, вместо этого он будет общаться с клиентом при помощи передачи данных в формате Json.

REST (REpresentational State Transfer — «передача репрезентативного состояния» или «передача самоописываемого состояния») — архитектурный стиль взаимодействия компонентов распределённого приложения в сети. Термин REST был введен в 2000 году Роем Филдингом, одним из авторов HTTP-протокола. Системы, поддерживающие REST, называются RESTful-системами.

Основные ограничения REST:

* модель клиент-сервер. Первое ограничение заключается в разделении клиента и сервера. Сервер реализует логику хранения данных, сложные вычисления и безнес-логику, а клиент реализует взаимодействие с сервером. Благодаря такому разделению мы можем масштабировать приложение – разделять большую нагрузку между несколькими серверами и подключать большое количество клиентов;
* отсутствие состояния. Во время запросов клиента информация о состоянии клиента на сервере не хранится (Stateless protocol). Запросы от клиента должны быть сформированы так, чтобы сервер получил всю необходимую информацию о запросе, состояние сессии сохраняется только на клиенте.
* кэширование. Веб-службы RESTful поддерживают кэширование, то есть процесс сохранения некоторых ответов на клиенте или на посреднике для сокращения времени ответа сервера;
* единобразие интерфейса. Hypermedia as the Engine of Application State (HATEOAS) — одно из ограничений REST, согласно которому сервер возвращает не только ресурс, но и его связи с другими ресурсами и действия, которые можно с ним совершить;
* слоистая архитектура. Концепция слоистой архитектуры заключается в том, что ни клиент, ни север не должны знать, как происходят вызовы дальше своих крайних соседей. Для этого используются «прослойки» – прокси серверы.

Протокол HTTP. Когда в браузере вводится какой-либо URL-адрес, например [www.telegram.ru](http://www.telegram.ru), на сервер отправляется запрос, идентифицированный URL-адресом. Формат этих запросов определяется протоколом HTTP – Hyper Text Transfer Protocol.

HTTP определяет следующую структуру запроса:

1. строка запроса (request line) — определяет тип сообщения;
2. заголовки запроса (header fields) — характеризуют тело сообщения, параметры передачи и прочие сведения;
3. тело сообщения (body) — необязательное;
4. метод, используемый в HTTP-запросе, указывает, какое действие вы хотите выполнить с этим запросом, например:
   1. GET: получить подробную информацию о ресурсе;
   2. POST: создать новый ресурс;
   3. PUT: обновить существующий ресурс;
   4. DELETE: удалить ресурс.

Простыми словами клиент запрашивает информацию у сервера в формате JSON, а сервер в свою очередь возвращает представление JSON.

Архитектура приложения будет делится на три уровня Controller, Service и Repository:

* контроллеры – это классы предназначены для обработки запросов, приходящих на сервер, запуска методов сервиса и ответа в нужном в формате;
* сервисы – это классы, которые отвечают за бизнес-логику приложение;
* репозитории – это классы, отвечающие за сохранение и извлечение некоторого данных из СУБД. В них нет сложной логики, они всего лишь формируют и выполняют запросы к базе данных.

Рассмотрим, как это работает на примере передачи запроса с данными от клиента к серверу. Клиент посылает POST запрос: Email, пароль, ФИО, номер телефона и так далее – на URL сервера /registration, его обрабатывает Controller, пример функции указан на листинге 2.

Листинг 2 – Функция для обработки POST запроса

@PostMapping(«/registration»)  
public ResponseEntity<?> createUser(@RequestBody User user) {  
 if (!userService.createUser(user)) {  
 return ResponseEntity.*badRequest*().body(«Пользователь с почтой: « + user.getEmail() + « уже существует»);  
 }  
 String credentials = user.getEmail() + «:» + user.getPassword();  
 String encodedCredentials = Base64.*getEncoder*().encodeToString(credentials.getBytes(StandardCharsets.*UTF\_8*));  
  
 return ResponseEntity.*ok*(«Регистрация прошла успешно»);  
}

Он в свою очередь вызывает функцию userService – CreateUser, в которые передаются данные из запроса. В userService, происходят необходимые модификации и проверки данных (проверка уникальности email или валидация пароля), если они требуются. И затем вызывается UserRepository, который в свою очередь наследуется от JpaRepository – это интерфейс, предоставляемый Spring Data Jpa, дающий методы для выполнения CRUD операций (Create, Read, Update, Delete). При вызове функции save, Spring Data JPA использует внутреннюю реализацию – EntityManager, которая уже непосредственно сохраняет данные в базе данных.

Структура приложения показана на рисунке 1.

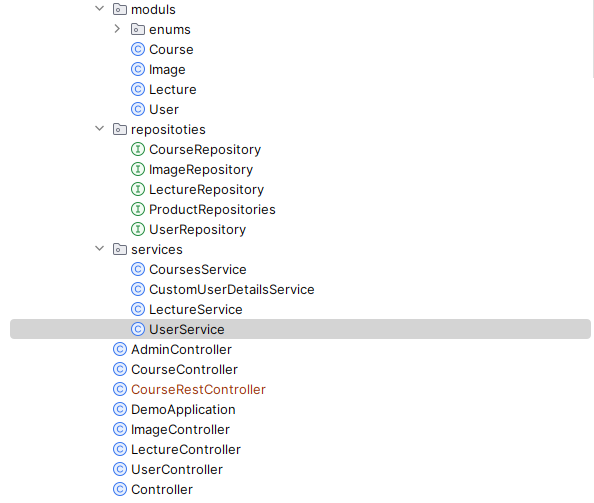


Рисунок 1 – Структура проекта

### Структура базы данных

База данных состоит из 7 объектов-таблиц: users, course, user-role, images, lectures, faq, support\_ticket (даталогическое описание БД)

#### Таблицы users

users – таблица, в которой хранятся данные о пользователе. Поля таблицы:

* id: BIGINT первичный ключ, уникальный для каждого пользователя;
* active: BIT переменная, которая указывает активен ли пользователь;
* date\_of\_created: DATETIME дата создания аккаунта;
* email: VARCHAR(255) почта пользователя, по которой происходит регистрация;
* father\_name: VARCHAR(255) отчество пользователя;
* first\_name: VARCHAR(255) имя пользователя;
* last\_name: VARCHAR(255) фамилия пользователя;
* father\_name: VARCHAR(255) отчество пользователя;
* group: VARCHAR(255) фамилия пользователя;
* birthday: DATA дата рождения пользователя;
* password: VARCHAR(1000) пароль пользователя;
* phone\_number: VARCHAR(255) номер пользователя;
* image\_id: BIGINT (Внешний ключ, ссылается на id в таблице images) фотография профиля пользователя;
* course\_id: BIGINT (Внешний ключ, ссылается на id в таблице course) поле, которая указывает на курс, в котором зарегистрирован пользователь.

#### Таблица user-role

user\_role – таблица, которая хранит роли пользователей:

* user\_id: BIGINT внешний ключ, ссылается на id в таблице users;
* roles: ENUM поле, содержащее роли пользователей.

#### Таблица image

image – таблица, в которой хранятся данные о фотографии. Поля таблицы:

* id: BIGINT (Первичный ключ) уникальный идентификатор для каждого изображения;
* bytes: LONGBLOB байтовое представление изображения;
* content-type: VARCHAR(255) — MIME тип контента изображения (например, image/jpeg);
* is\_preview\_image: BIT — переменная, указывающая, является ли изображение превью;
* name: VARCHAR(255) — название изображения;
* original\_file\_name: VARCHAR(255) — исходное название файла изображения;
* size: BIGINT — размер изображения в байтах.

#### Таблица lectures

lectures – таблица, в которой хранятся данные о лекциях. Поля таблицы:

* id: BIGINT (Первичный ключ) — уникальный идентификатор для каждой лекции;
* comment: VARCHAR(255) — комментарий к лекции;
* date\_of\_create: DATETIME — дата создания лекции;
* name: VARCHAR(255) — название лекции;
* number\_of\_lecture: INT — количество лекций;
* course\_id: BIGINT — внешний ключ, ссылающийся на id в таблице course, указывающий, к какому курсу относится лекция.

#### Таблица course

course – таблица, в которой хранятся данные о курсах. Поля таблицы:

* id: BIGINT (Первичный ключ) — уникальный идентификатор для каждого курса;
* date\_of\_created: DATETIME — дата создания курса;
* descrition: TEXT — описание курса;
* preview\_image\_id: BIGINT — внешний ключ, ссылающийся на id в таблице images, указывающий на превью изображение курса;
* title: VARCHAR(255) — название курса.

#### Таблица faq

faq – таблица, в которой хранятся данные о вопросах и ответах в чат боте.

* id: BIGINT (Первичный ключ) — уникальный идентификатор для каждого вопроса. Поля таблицы:
* QuestionNumber: INT — номер вопроса;
* question: VARCHAR(255) — текст вопроса;
* answer: VARCHAR(255) — текст ответа на вопрос;
* user: User — внешний ключ, ссылающийся на пользователя, задавшего вопрос;
* date\_of\_create: DATETIME — дата создания записи.

#### Таблица support\_ticket

* support\_ticket – таблица, в которой хранятся данные об обращения в службу поддержки. Поля таблицы:
* id: BIGINT (Первичный ключ) — уникальный идентификатор для каждой заявки;
* subject: VARCHAR(255) — тема обращения;
* message: TEXT — текст сообщения обращения
* adminResponce: TEXT — текст ответа администратора;
* closed: BOOLEAN — статус заявки (закрыта/открыта);
* user: User — внешний ключ, ссылающийся на пользователя, отправившего заявку;
* date\_of\_create: DATETIME — дата создания записи;  
  Структура базы данных показана на рисунке 2.

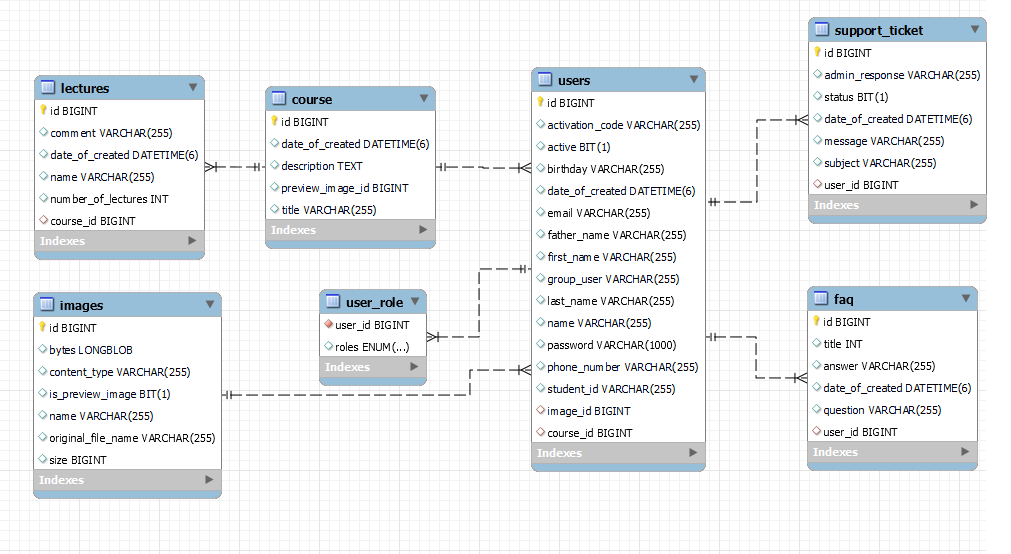


Рисунок 2 – Структура базы данных

#### Отношения между таблицами

Таблица course связана с users One-to-Many, то есть у одного курса может быть множество зарегистрированных пользователей. В таблице users есть внешний ключ course\_id, который указывает на id курса. Так же course связана с lectures, так же связью One-to-Many, соответственно один курс – много лекций, внешний ключ – course\_id. Таблица users связан One-to-Many с user\_role, поле user\_id в таблицу user\_role указывает на id в таблице users. Так же users связана с images One-to-One, то есть у одного пользователь может быть только одна фотография. Таблица faq Many-to-One связана с таблицей user, то есть у каждого пользователя может быть много обращений в поддержку, внешний ключ user\_id. Таблица support\_ticket связана One-to-One с таблицей users. У одного пользователя может быть лишь один support\_ticket.

#### Создание и настройка проекта, подключение базы данных

Для создания проекта необходимо перейти на сайт https://start.spring.io/ и сгенерировать проект, указав название, описание, сборщик проекта, необходимые компоненты фреймворка Spring, а также язык программирования, и его версию. Сайт https://start.spring.io/ показан на рисунке 3.

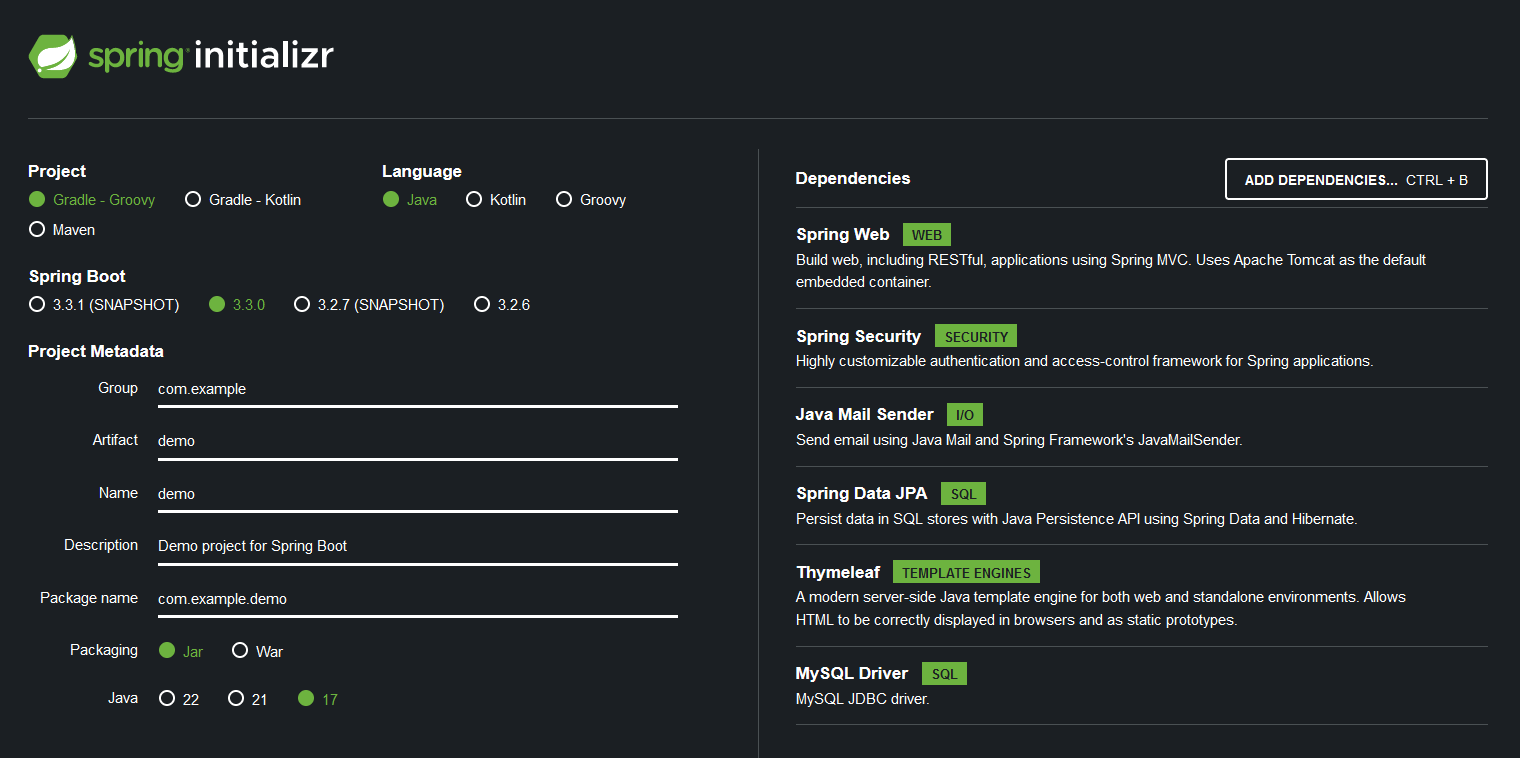


Рисунок 3 – Страница сайта Spring Initializr

Далее нам надо настроить файл application.properities. Он является конфигурационным файлом для Spring Boot приложения. Файл показан в листинге 2.

Листинг 2 – Код файл application.properties

spring.application.name=demo

spring.datasource.url=jdbc:mysql://127.0.0.1:3306/digital\_department

spring.datasource.username=root

spring.datasource.password=1234

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.freemarker.expose-request-attributes=true

spring.jpa.properties.hibernate.jdbc.lob.non\_contextual\_creation=true

spring.mail.host=smtp.yandex.ru

spring.mail.username=gapmeeee@yandex.ru

spring.mail.password=gxmwreaiswdkbqfx

spring.mail.port=465

spring.mail.protocol=smtps

mail.debug=true

spring.mail.properties.mail.smtp.starttls.enable=true

spring.mail.properties.mail.smtp.ssl.trust=smtp.yandex.com

Он содержит настройки для различных компонентов и сервисов, которые используются в приложении:

* spring.application.name=digital\_department. Задает имя приложения. В данном случае, имя приложения – digital\_department;
* spring.datasource.url. URL для подключения к базе данных MySQL. Приложение подключается к базе данных digital\_department, находящейся на локальном хосте (127.0.0.1) и порту 3306;
* spring.datasource.username и spring.datasource.password. Учетные данные для подключения к базе данных (пользователь root и пароль 1234);
* spring.jpa.hibernate.ddl-auto. Определяет стратегию для инициализации базы данных. Значение update означает, что Hibernate будет автоматически обновлять схему базы данных на основе моделей сущностей;
* spring.jpa.show-sql. Если установлено в true, Hibernate будет выводить SQL-запросы в консоль для удобства отладки;
* spring.freemarker.expose-request-attributes=true. Позволяет FreeMarker получить доступ к атрибутам запроса в шаблонах;
* spring.jpa.properties.hibernate.jdbc.lob.non\_contextual\_creation=true. Оптимизирует работу с большими объектами в Hibernate, отключая контекстное создание LOB;

Эти настройки конфигурируют Spring Boot для отправки электронной почты через SMTP сервер Yandex:

* spring.mail.host. SMTP сервер для отправки писем (smtp.yandex.ru);
* spring.mail.username и spring.mail.password. Учетные данные для доступа к почтовому ящику (gapmeeee@yandex.ru и пароль gxmwreaiswdkbqfx);
* spring.mail.port. Порт SMTP сервера (465);
* spring.mail.protocol. Протокол для отправки почты (smtps);
* mail.debug. Если установлено в true, включает режим отладки для почтовой системы;
* spring.mail.properties.mail.smtp.starttls.enable. Включает поддержку STARTTLS;

Настройка VPS сервера.

Подключение к серверу по протоколу SSH через программу Putty:  
 ssh root@your\_server\_ip.

Необходимы строки для настройки операционной системы Ubuntu:

1. обновление списка пакетов и установка обновления:
   * sudo apt update;
   * sudo apt upgrade -y.
2. установка Java – sudo apt install openjdk-17-jdk -y;
3. установка MySQL –sudo apt install mysql-server -y;
4. настройка MySQL – sudo mysql\_secure\_installation.

Создание базы данных и пользователя указано в листинге 3.

Листинг 3 – Создание базы данных

sudo mysql -u root -p

CREATE DATABASE digital\_department;

CREATE USER 'root'@'%' IDENTIFIED BY '1234';

GRANT ALL PRIVILEGES ON digital\_department.\* TO 'root'@'%';

FLUSH PRIVILEGES;

EXIT;

Сервер Nginx устанавливаем командой sudo apt install nginx -y;

Настройка Nginx для проксирования запросов к Spring Boot приложению. Конфигурационный файл создается командой   
sudo nano /etc/nginx/sites-available/your\_app.

Конфигурация Nginx показана в листинге 5.

Листинг 4 – Конфигурация Nginx

server {

listen 80;

server\_name your\_server\_ip;

location / {

proxy\_pass http://127.0.0.1:8080;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Proto $scheme;

}

}

Проверка конфигурации Nginx на наличие ошибок и его перезапуск:

* sudo nginx -t;
* sudo systemctl restart nginx.

### Регистрация и авторизация

#### Регистрация

Код классов UserRestController, userRepository, UserService, User указаны листингах А.1– А.4 соответственно.

Пользователь заполняет форму регистрации на клиенте, вводит свои данные:

* ФИО;
* никнейм;
* почта;
* номер телефона;
* группа;
* пароль;
* студенческий билет;
* дата рождения.

Затем отправляет запрос на сервер с помощью HTTP POST запроса. При отправке запроса на сервер данные пользователя передаются в контроллер UserRestController в метод createUser. В методе createUser происходит проверка наличия пользователя с таким же email в базе данных с помощью вызова метода userRepository.findByEmail(email).

Если пользователь с таким email уже существует, контроллер возвращает сообщение об ошибке, и пользователь остается на странице регистрации. Если не существует, создается новый объект User, ему присваиваются введенные данные, а пароль надежно шифруется алгоритмом BCrypt, и методом userRepository.save(user) сохраняет пользователя в базу данных. В базу данных пользователь сохраняется с ролью студента.

После сохранения пользователя в базе данных, генерируется уникальный код активации с помощью UUID.randomUUID().toString(), который присваивается пользователю и также сохраняется в базе данных. Далее формируется письмо с активационным кодом и отправляется на указанный email пользователя с помощью вызова метода mailSender.send() из класса MailService. Пользователю отображается страница с сообщением о необходимости активации аккаунта, и он может перейти по ссылке из письма для активации. При переходе по ссылке на активацию, пользователь отправляет GET запрос на сервер с уникальным кодом активации в качестве части URL. Письмо с кодом активации показано на рисунке 4.

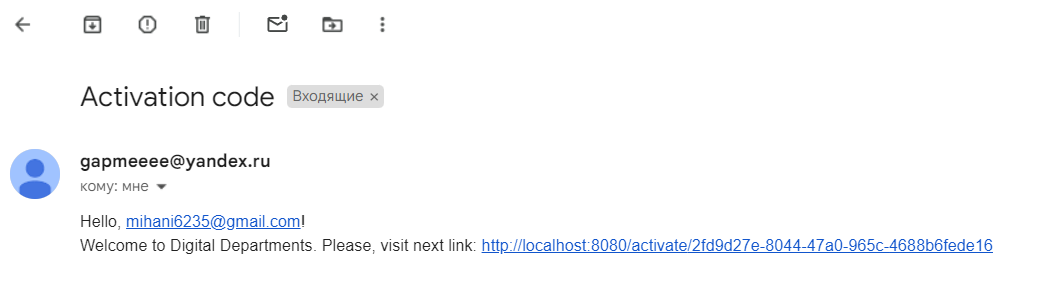
**

Рисунок 4 – Письмо с кодом активации

Сервер обрабатывает запрос, вызывает метод activate в классе UserRestController, которому передаются в качестве параметра код активации из URL, и с помощью функции activateUser из класса UserService вызывается процесс активации пользователя, в котором по коду активации находится соответствующий пользователь в базе данных. Если пользователь найден, его activationCode удаляется из базы данных, что означает успешную активацию, и данные пользователя обновляются в базе данных с помощью метода userRepository.save(user). После успешной регистрации пользователя приложение возвращает ответ с кодом 200.

#### Аутентификация

Аутентификация осуществляется, при каждом запросе к защищенному ресурсу, так как при REST архитектуре сервер не хранит информацию о сессии пользователя. Учетные данные представлены в виде строки «Имя\_пользователя:Пароль» закодированные алгоритмом Base64.

Процесс аутентификации осуществляется:

* отправка HTTP запроса клиента на защищенный ресурс сервера.
* отправка требований аутентификации на «клиент». Сервер, обнаружив, что запрос не содержит аутентификационных данных, отправляет обратно ответ с кодом состояния 401 (Unauthorized) и заголовком WWW-Authenticate: Basic realm=«realm». Заголовок realm определяет область аутентификации, которая обычно представляет собой идентификатор защищенной области или приложения.
* отправка учетных данных с клиента на сервер. Клиент повторно отправляет запрос с заголовком Authorization, содержащим строку «Basic», за которой следует закодированная в Base64 строка «Имя\_пользователя:Пароль».
* проверка учетных данных, полученных с клиента на сервере. Сервер извлекает учетные данные из заголовка Authorization, декодирует их из Base64 и затем проверяет их на соответствие сохраненным данным пользователя.
* выполнение аутентификации. Если учетные данные верны, сервер выполняет аутентификацию пользователя. Затем пользователь получает доступ к запрашиваемому ресурсу.

Таки образом, нет отдельной функции, обрабатывающей аутентификацию пользователя, все запросы к запрещенным ресурсом, будут аутентифицироваться в системе.

#### Безопасность

Код класса SecurityConfig показан в листинге А.5. Для настройки безопасности проекта нужно создать класс SecurityConfig. Класс SecurityConfig указан в листинге. В нем определяются правила доступа к различным URL-адресам, настраивается способ аутентификации пользователей и устанавливается метод шифрования паролей. Этот класс предоставляет централизованный способ определения параметров безопасности в приложении, обеспечивая контроль доступа к защищенным ресурсам и защиту пользовательских данных.

Этот класс помечен аннотациями @Configuration и @EnableWebSecurity, указывающими на то, что он является конфигурационным классом Spring и включает поддержку безопасности веб-приложений с использованием Spring Security.

Также присутствует аннотация @EnableGlobalMethodSecurity (prePostEnabled = true), активирующая поддержку аннотаций безопасности, таких как @PreAuthorize и @PostAuthorize.

Далее в этом классе определен конструктор, в который передается экземпляр пользовательского сервиса (CustomUserDetailsService), предоставляющего данные о пользователях для аутентификации и авторизации.

Метод securityFilterChain создает цепочку фильтров безопасности для обработки HTTP-запросов. Внутри метода настраиваются правила доступа к различным URL-адресам: некоторые URL разрешены для всех (permitAll), в то время как для остальных требуется аутентификация (authenticated).

Метод userDetailsService создает и возвращает экземпляр пользовательского сервиса (CustomUserDetailsService), который будет использоваться для загрузки пользовательских данных во время аутентификации.

Метод passwordEncoder создает и возвращает экземпляр кодировщика паролей (PasswordEncoder). В данном случае используется алгоритм BCrypt.

И наконец, метод authenticationManager создает и возвращает AuthenticationManager, который является основным интерфейсом для аутентификации пользователей. Он принимает учетные данные пользователя, проверяет их подлинность и возвращает информацию о пользователе и его правах доступа. Для настройки authenticationManager в Spring Security используется AuthenticationManagerBuilder, который позволяет определить пользовательский сервис для загрузки информации о пользователях и кодировщик паролей для проверки паролей пользователей. Класс SecurityConfig в Spring Security обычно содержит настройки authenticationManager, определяющие правила доступа, пользовательский сервис и кодировщик паролей для вашего приложения.

Таким образом, данный класс SecurityConfig определяет конфигурацию безопасности вашего приложения, включая правила доступа, пользовательский сервис, кодировщик паролей и менеджер аутентификации.

#### Разработка курсов и лекций

Код классов CourseRestController, CourseService, CourseRepository, LectureService, LectureRepository, Course указан листинге А.6–А.11 соответственно.

Когда пользователь успешно зарегистрирован и активировал свою учетную запись, он может взаимодействовать с курсами. Пользователь отправляет Get запрос и сервер возвращает ответ в виде JSON файла с курсом. Если пользователь зарегистрирован на курсе, то CourseRestController вызывает метод coursesService.getCourseById(id), где id – это параметр из URL запроса «/courses/{id}», после чего делается запрос в базу данных, в котором ищется и возвращается курс по id. Далее данные из базы данных возвращаются на «клиент». Если пользователь не зарегистрирован ни одном из курсов, возвращается предупреждение о том, что он не зарегистрирован ни на одном из курсов, и просьба обратится к администратору или преподавателю.

Администраторы имеют доступ к дополнительным функциям, которые помечены аннотацией @PreAuthorize(«hasRole('ROLE\_ADMIN')»), таким как создание и удаление курсов.

Чтобы создать курс класс CourseController обрабатывает Post запросы администраторов по URL «/courses/create», в которых передаются название курсов и описание к ним. CourseRestController передает эти данные CourseServices при помощи метода saveCourse. После чего уже Service сохраняет их в базу данных, добавляя время создания и id курса.

Для удаления курса, администратор отправляет Post запрос по URL «/courses/delete/{id}». После чего вызывается функция deleteCourses(id) (id это переменная из URL ссылки, которая означает id нужного курса), в которой вызывается метод класса coursesService – deleteCourse() с параметром id. В Service делается запрос к базе данных на удаление курса по id.

Еще одна из функций администраторов и преподавателей – это добавление и удаление лекций к курсам. Принцип создания лекций точно такой же, как и курсов. При создании новой лекции администраторы или преподаватели отправляют заполняют данные в форме о лекции: название, описание и файлы к лекции, затем отправляют POST запрос по URL «/courses/{id}/lecture/create». После чего вызывается функция класса CourseRestController – createLecture (@PathVariable Long id, Lecture lecture), где id – это параметр из URL ссылки, означающий id курса, а lecture – это лекция, переданная из формы. Далее для переменной lecture вызывается метод SetCourse(), в качестве параметра передается лекция, найденная по id. Затем при помощи функции saveLecture() класса LectureService новая лекция сохраняется в базу данных.

#### Роли пользователей

Код перечисления Role показано в листинге А.12. В коде Role представлено определение перечисления (enum) Role, которое реализует интерфейс GrantedAuthority. Этот интерфейс используется в Spring Security для представления прав доступа пользователя. Метод getAuthority() возвращает имя роли в виде строки. Роли, определенные в перечислении (ROLE\_USER, ROLE\_ADMIN, ROLE\_TEACH), могут быть присвоены пользователям в приложении. Например, при аутентификации пользователей в вашем приложении через Spring Security.

В контроллерах или методах сервиса вы можете аннотировать доступ к методам или участкам кода с помощью аннотации @PreAuthorize или других аннотаций Spring Security, указав роли, которые имеют доступ к этому методу или ресурсу.

#### Возможности администратора

Код классов AdminRestController, CourseRestController показаны в листингах А.13, А.6 соответственно.

Администратор имеет доступ к управлению пользователями и курсами. Класс AdminRestController, объявленный с аннотацией @PreAuthorize(«hasRole('ROLE\_ADMIN')»), обеспечивает доступ только для пользователей с ролью администратора.

Метод admin обрабатывает GET-запросы по пути «/admin». При вызове этого метода сервер возвращает список пользователей и их курсов, полученный из сервисов UserService и CoursesService. Эти данные затем отображаются на странице администратора.

Метод userBan обрабатывает POST-запросы по пути «/admin/user/ban/{id}». При получении запроса метод вызывает сервис UserService чтобы изменить статус активности пользователя, заблокировав или разблокировав его.

Метод userEdit обрабатывает POST-запросы по пути «/admin/user/edit/{user}». Он получает данные измененного пользователя из тела запроса, а затем добавляет новую информацию о пользователе, в базу данных.

Если все прошло удачно возвращает код 200 и сообщение, о том, что изменения пользователя прошли успешно на «клиент».

Метод addCourse обрабатывает POST-запросы по пути «/admin/user/edit/courses/{user}». Он получает id курса из тела запроса и id пользователя из ссылки. А затем назначает курс пользователю, вызывая методы setCourse и saveUser из UserService, а также addUserInCourse из CoursesService. После чего возвращает клиенту ответ с кодом 200 и сообщение, о том, что все выполнилось успешно.

Технически взаимодействие с базой данных осуществляется через сервисы UserService и CoursesService, которые используют репозитории для выполнения CRUD-операций (создание, чтение, обновление, удаление) над сущностями User и Course. Например, метод list в UserService и CoursesService возвращает все записи пользователей и курсов соответственно, а методы banUser, changeUserRoles и addUserInCourse обновляют соответствующие записи в базе данных.

#### Разработка чат бота

Код классов FAQController, FAQService, FAQRepository и FAQ показаны в листингах А.14 – А.17 соответственно.

На рисунке 5 изображена UML-диаграмма последовательностей, передающая суть взаимодействия данных объектов.

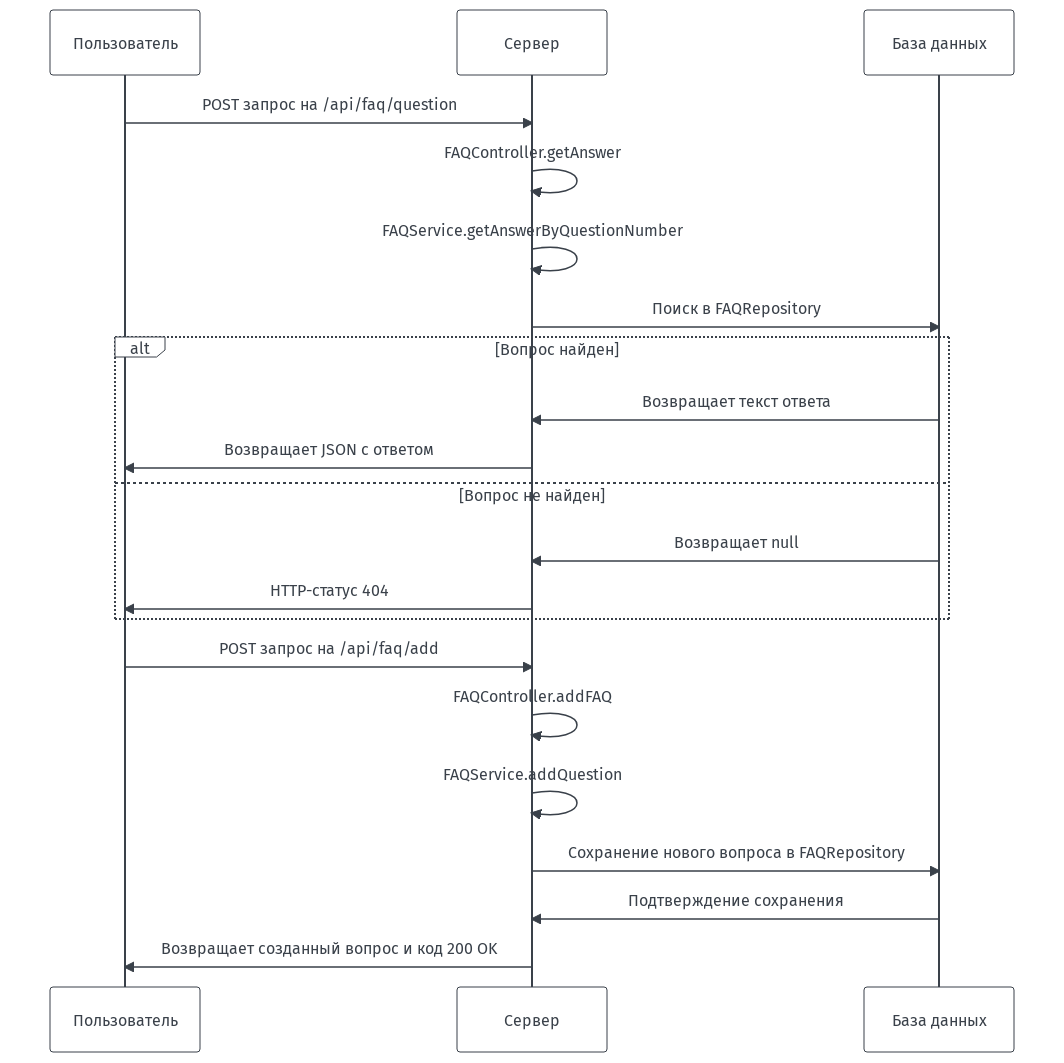


Рисунок 5. – UML-диаграмма последовательностей чат-бота

Контроллер FAQController отвечает за обработку HTTP-запросов, связанных с вопросами и ответами. Он обрабатывает запросы и возвращает ответы в формате JSON. Метод getAnswer обрабатывает POST-запросы по пути /api/faq/question. Он принимает один параметр – номер вопроса в теле запроса. Этот параметр передается с помощью аннотации @RequestParam.

Затем вызывается метод getAnswerByQuestionNumbe из FAQService для обработки данных. Этот метод принимает номер вопроса и при помощи репозитория FAQRepository ищет данные в базе. Если вопрос найден, метод возвращает текст ответа. Если вопрос не найден, метод возвращает null. После он возвращается клиенту в формате JSON, если нет, возвращается   
HTTP-статус 404.

Метод addFAQ обрабатывает POST-запросы по пути /api/faq/add. Он принимает параметры в теле запроса: questionNumber, question, и answer, которые представляют собой номер, текст вопроса и текст ответа соответственно для нового вопроса. Эти параметры также передаются с помощью аннотации @RequestParam. После получения данных о новом вопросе, контроллер вызывает метод addQuestion сервиса FAQService, передавая ему текст вопроса и ответа. Сервис обрабатывает запрос, создает новый объект типа FAQ с полученными данными и сохраняет его в базе данных, используя репозиторий FAQRepository. Затем контроллер возвращает клиенту ответ, содержащий созданный вопрос в теле ответа, и код 200 OK.

#### Разработка технической поддержки

Код классов SupportTicketController, SupportTicketRepository, SupportTicketService показаны в листингах А.18, А19, А.20 соответственно.

Контроллер SupportTicketController отвечает за обработку HTTP-запросов, связанных с заявками в поддержку.

Метод createSupportTicket обрабатывает POST-запросы по пути /api/support-tickets. Он принимает один параметр – объект SupportTicket в теле запроса. Затем вызывается метод createSupportTicket из SupportTicketService для обработки полученных данных с клиента. Этот метод принимает объект SupportTicket и при помощи репозитория SupportTicketRepository сохраняет его в базе данных. Если заявка создана успешно, метод возвращает созданную заявку с HTTP-статусом 201 (Created).

Метод getAllSupportTickets обрабатывает GET-запросы по пути /api/support-tickets. Он не принимает параметров. Метод вызывает getAllSupportTickets из SupportTicketService, который возвращает список всех заявок в базе данных. Затем контроллер возвращает этот список клиенту в формате JSON с HTTP-статусом 200 (OK).

Метод respondToSupportTicket обрабатывает POST-запросы по пути /api/support-tickets/{ticketId}/respond. Он принимает два параметра: ID заявки в URL пути и ответ администратора в виде строки, переданной через параметр запроса. Затем вызывается метод respondToSupportTicket из SupportTicketService, который обновляет заявку, добавляя ответ администратора. Если заявка найдена и обновлена успешно, метод возвращает обновленную заявку с HTTP-статусом 200 (OK). Если заявка не найдена, возвращается HTTP-статус 404 (Not Found).

Метод closeSupportTicket обрабатывает POST-запросы по пути /api/support-tickets/{ticketId}/close. Он принимает один параметр - ID заявки в URL пути. Затем вызывается метод closeSupportTicket из SupportTicketService, который закрывает заявку, изменяя её статус на «закрыта». Если заявка найдена и обновлена успешно, метод возвращает закрытую заявку с HTTP-статусом 200 (OK). Если заявка не найдена, возвращается HTTP-статус 404 (Not Found).

Контроллер также инъектирует сервис SupportTicketService с помощью аннотации @Autowired, что позволяет использовать методы сервиса для выполнения логики бизнес-процессов. Аннотация @RestController означает, что все методы контроллера возвращают данные в формате JSON. Аннотация @RequestMapping(«/api/support-tickets») задает базовый URL для всех методов контроллера.

В данной главе были разработаны архитектура приложения, и структура базы данных. Произведены основные настройки конфигурации Spring Boot проекта и VPS сервера. Настроена безопасность приложения при помощи Spring Security. А также разработаны основные модули, такие как регистрация, аутентификация, курсы и лекции, роли пользователей возможности администраторов и пользователей, техническая поддержка и чат-бот.

## Безопасность и экологичность работы

### Экологические аспекты утилизации микропроцессорной техники

Для начала важно понимать, что такое микропроцессорная техника. Микропроцессорная система представляет собой комплектное изделие, состоящее из одного или нескольких устройств, главным из которых является микропроцессор или микроконтроллер. Микропроцессор выполняет роль центрального блока, управляющего всеми компонентами устройства и осуществляющего арифметические и логические операции с информацией.

Когда компьютерное оборудование ломается или устаревает, его нельзя просто выбросить. Необходима специальная утилизация, которая предотвратит вред окружающей среде и избавит от штрафов (согласно статье 8.2 Административного правового кодекса). Эта практика постепенно внедряется и в России.

Правильная утилизация важна из-за содержания в оргтехнике драгоценных металлов и опасных химических веществ, таких как ртуть, мышьяк, свинец и кадмий, а также небольшого количества серебра, золота и платины. Переработка таких устройств сложна, так как различные компоненты требуют разных методов утилизации. Неправильная переработка может привести к попаданию токсинов в организм человека через дыхательные пути и кожу, что вызывает хронические заболевания.

Электронные отходы должны проходить двухэтапную утилизацию. На первом этапе происходит разборка, утилизация и переплавка деталей, при этом выделяются вредные вещества, такие как бромированные или хлорированные фенилы и эфиры. Неполное сгорание топлива может привести к выбросу опасных ароматических углеводородов, которые вызывают заболевания дыхательных путей и сердечно-сосудистой системы.

Второй этап переработки делится на несколько категорий в зависимости от содержания и стоимости извлекаемых элементов:

* Элементы с низким содержанием драгоценных металлов, такие как платы телевизоров, блоки питания, компьютеры и бытовая электроника;
* Радиоэлементы с более высоким содержанием ценных металлов;
* Элементы с высоким содержанием золота, платины и других редкоземельных металлов, такие как интегральные схемы и оптоэлектроника.

Для каждой категории используется свой метод переработки. Существуют физические и химические методы:

Физические методы включают:

1. механическую переработку, при которой происходит сжатие массы и выделение тепла. Температура свыше 250°C разрушает химические связи и выделяет вредные фенолы;
2. воздушную сепарацию, при которой частицы разделяются по плотности и массе;
3. электростатическое разделение, где под действием электрического поля металлические частицы отделяются от пластмассовых;
4. магнитную сепарацию, при которой частицы разделяются по магнитным свойствам.

Химические методы включают:

1. пиролиз, при котором нагреваемая масса постепенно отделяет вещества с разной температурой плавления;
2. гидрометаллургический метод, при котором отходы обрабатываются кислотами и щелочами с последующим электролизом для выделения металлов;
3. биометаллургическая сепарация, при котором используются микроорганизмы для выделения драгоценных металлов из медной руды;
4. газификация, при котором отходы нагреваются до 1600°C в системе печей с подачей кислорода.

За последние десятилетия страны стали намного внимательнее относиться к проблеме утилизации электронных отходов. Сейчас существуют законы и системы поощрения за утилизацию опасных отходов. Например, в США компания Dell предоставляет гранты на создания пунктов приема техники, а компания Sony создала проект по переработке своей продукции.

В России проблема утилизации микропроцессорной техники также остается актуальной, хотя законы, регулирующие утилизацию, начали формироваться сравнительно недавно, в данный момент они активно развиваются. Также появилось большое количество небольших компаний, имеющих соответствующую сертификацию, которые за небольшую плату утилизируют отходы.

### Расчет системы искусственного освещения

В современном мире человек получает очень много зрительной информации. На протяжении всего дня человек пользуется смартфонами, компьютерами, телевизорами, книгами, из-за чего глаза очень сильно напрягаются и переутомляются, что в дальнейшем может привести к заболеваниям. Чтобы хоть как-то уберечь глаза необходимо хорошо осветить рабочее место.

Недостаток света может привести к ухудшению зрения, головным болям и снижению общей продуктивности. Кроме того, хорошее освещение способствует улучшению концентрации и повышению работоспособности. Естественный свет является лучшим вариантом для освещения рабочего пространства, но при его недостатке следует использовать искусственное освещение, близкое по характеристикам к естественному.

Все необходимые и актуальные схемы освещения представлены  
в СНиП 23-05-95.

Для расчета системы равномерного освещения горизонтальной рабочей поверхности используют метод коэффициента использования светового потокам.

Произведем расчет искусственного освещения, выполняемого светильниками ЛСПО1 с двумя люминесцентными лампами, длинной l = 1534 мм.

Исходные данные для расчетов показаны на таблице 2.

Таблица 2 – Исходные данные для расчетов

|  |  |
| --- | --- |
| Параметры для расчета | Значения |
| 1 | 2 |
| А, м | 12 |
| В, м | 6 |
| h, м | 4 |
| 1 | 2 |
| Pп | 0,8 |
| Pс | 30 |
| Eн, лк | 300 |

Определение высоты подвеса светильника над рабочей поверхностью по формуле 1:

, (1)

где hP = 0,8м – высота рабочей поверхности;

hC = 0,25 (h – hp) – расстояние светового центра светильника от потолка.

Таким образом HП = 2,4.

Определение индекса помещения по формуле 2:

, (2)

где А и В соответственно длина и ширина помещения.

Таким образом i равна 1,67.

Выбор коэффициента запаса К и коэффициент использования светового потока ƞ по табличным значениям:

К = 1,5;

Ƞ = 46.

Определение расстояния между рядами светильников и от стен до ближайшего ряда. Оптимальное расстояние L между рядами светильников находится из соотношения 3:

, (3)

где λ – коэффициент оптимального расстояния между геометрическими центрами светильников по светотехническим требованиям.

Таким образом λ равна 2,64.

Определение общего количества светильников N при условии равномерного освещения осуществляется по формулам 4, 5, 6:

; (4)

; (5)

; (6)

Определение светового потока одной лампы по формуле 7:

, (7)

где – расчетный световой поток, лк.

Таким образом ФЛ.РАСЧ равно 2421.

Определение по таблице 3 данные подходящего типа лампы и ее мощности:

Таблица 3 – Параметры люминесцентных ламп общего назначения   
(ГОСТ 6825-91)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Мощность W, Вт | Сила тока I, А | Напряжение U, В | Световой поток Ф, лм | | | | |
| Среднее значение (после 100 ч горения) для ламп цветности | | | | |
| ЛБ | ЛТБ | ЛХБ | ЛД | ЛДЦ |
| 30 | 0,35 | 104±0,4 | 2180  (–140) | 2020  (–100) | 1940  (–100) | 1800  (–180) | 1500  (–80) |
| 40 | 0,43 | 103±0,3 | 3200  (–160) | 3100  (–155) | 3000  (–150) | 2500  (–125) | 2200  (–110) |
| 65 | 0,67 | 110±0,0 | 4800  (–240) | 4850  (–340) | 4400  (–220) | 4000  (–200) | 3150  (–160) |
| 80 | 0,87 | 102±10,2 | 5400  (–270) | 5200  (–250) | 5040  (–240) | 4300  (–215) | 3800  (–190) |

Исходя из расчетов необходимой в нашем случае лампой является лампа ЛД40 со следующими характеристиками:

Ф1ТАБЛ = 2500;

W = 40Вт;

I = 0,43А;

V = 103 ± 0,3B.

Световой поток выбранной лампы по формуле 8 равен:

, (8)

где – расчетный световой поток, лк.

– световой поток, определенный нами по табличным данным.

Таким образом, лампа подобрана верно.

Определение соответствия проектируемой системы освещения требованиям СанПиН 2.2.4.3359-16, выполняется проверочным расчетом искусственного освещения. Освещение удовлетворяет требованиям при условии ЕФ ≥ ЕН. Проверочный расчет фактической освещенности по формуле 9:

(9)

Таким образом ЕФ равна 309,76.

Определение потребляемой мощности осветительной установки по   
формуле 10:

(10)

где p – мощность лампы, Вт;

N – число светильников;

n – число ламп в светильниках;

kp – коэффициент, учитывающий потери пускорегулирующей аппаратуры.

Таким образом P равна1600.

Проведя расчёты, можно уверенно сделать вывод, так, как Eф > Ен, то расчет соответствует проектируемой системе искусственного освещения, и требованиям СанПиН 2.2.4.3359-16.

### Выводы по главе

В этой главе были изложены аспекты утилизации микропроцессорной техники, которые необходимы для поддержания экологии нашей планеты. Также был проведен системы искусственного освещения. Такие условия должны обеспечивать комфортную работу для сотрудников, студентов и школьников. Достаточное освещение позволит сохранить хорошую работоспособность в течение всего рабочего дня, повысить, как в количественном, так и в качественном отношениях производительность труда.

# Заключение

В первой главе выпускной квалификационной работы был произведен анализ предметной области, в котором было выявлено, что на данный момент требуется разработка web-приложения для поддержания процесса обучения студентов и облегчения взаимодействия студентов и преподавателей Цифровой кафедры. Был проведен анализ главных конкурирующих образовательных сервисов. А также сформированы технические требования для написания web-приложения для Android, нужные Цифровой кафедре.

Во второй главе был проведен анализ инструментов и технологий, необходимых для разработки web-приложения для мобильных устройств на платформе Android. Рассмотрены основные компоненты, такие как Java и Spring Framework, которые обеспечивают гибкость, производительность и безопасность приложения. Среда разработки IntelliJ IDEA выбрана за её мощные инструменты для рефакторинга кода, автодополнения и интеграции с фреймворками Spring и Hibernate. Для тестирования API используется Postman, позволяющий эффективно управлять и тестировать HTTP-запросы. В качестве системы управления базой данных выбрана MySQL, обеспечивающая высокий уровень безопасности и поддержку различных типов данных. Hibernate фреймворк используется для упрощения работы с базой данных посредством объектно-реляционного отображения (ORM). Для хостинга и развертывания приложения выбран VPS сервер с операционной системой Linux Ubuntu 22.04 LTS и сервером Nginx, что обеспечивает надежность, масштабируемость и гибкость в управлении серверными ресурсами. Эти технологии и инструменты были тщательно проанализированы для обеспечения надежности, производительности и удобства разработки, что делает их идеальными для реализации поставленных задач проекта.

В данной главе описаны основные этапы разработки backend-приложения для платформы Android. Рассмотрена архитектура приложения на основе RESTful подхода. Изложены принципы работы архитектуры, включающей уровни Controller, Service и Repository.

Структура базы данных проанализирована и описана с указанием таблиц и полей, необходимых для хранения информации о пользователях, изображениях, лекциях и курсах.

Подробно изложен процесс создания и настройки проекта с использованием Spring Boot.

Описаны шаги по настройке VPS сервера.

Также был описан основной функционал backend приложения:

* В разделе «Регистрация и авторизация» описаны шаги процесса регистрации пользователя с подробным описанием каждого этапа, включая проверку наличия пользователя в базе данных, шифрование пароля, отправку письма с активационным кодом и обработку запроса на активацию. Также представлены основные принципы аутентификации пользователей при помощи Spring Security.
* В разделе «Разработка курсов и лекций» было рассмотрено создание, удаление и изменение курсов и лекций администратором, а также взаимодействие пользователя с ними.
* В разделе «Возможности администратора» были рассмотрены возможности администратора, такие как удаление пользователей, блокировка, личных данных пользователя.
* В разделе «Разработка чат-бота» был рассмотрен функционал чат-бота.
* В разделе «Техническая поддержка» была показана реализация тех. поддержки для пользователя и реализация ответов администраторов на вопросы.

В главе «4 Безопасность и экологичность работы» мы рассмотрели почему очень важно правильно утилизировать микропроцессоры. В микропроцессорной технике содержится большое количество опасных химических веществ и драгоценных металлов, поэтому ее переработка требует специализированных методов, чтобы избежать загрязнения окружающей среды и здоровья людей.

Также была рассчитана система искусственного освещения, который играет важную роль в обеспечении комфортной и безопасной рабочей среды. Правильное освещение не только снижает нагрузку на глаза и предотвращает утомление, но также способствует повышению продуктивности и общему благополучию работников. Расчет проведен с учетом всех необходимых параметров и стандартов по требования СанПин.

# Перечень использованных информационных ресурсов

1. Документация Spring. – Текст : электронный // spring.io : [Сайт]. – URL: https://docs.spring.io/spring-boot/documentation.html (дата обращения: 10.04.2024).
2. Документация Java. – Текст : электронный // oracle.com : [Сайт]. – URL: https://docs.oracle.com/en/java/ (дата обращения: 10.04.2024).
3. Менеджер Docker. – Текст : электронный // docker.com : [Сайт]. – URL: https://docs.docker.com/manuals/ (дата обращения: 10.04.2024).
4. Уоллс Крейг. Spring в действии / Крейг Уоллс. – Москва :   
   ДМК Пресс, – 2022. – 544с.
5. Документация Postman. – Текст : электронный //  postman.com :   
   [Сайт]. – URL: https://learning.postman.com/docs/introduction/overview/ (дата обращения: 13.04.2024).
6. Документация MySQL. – Текст : электронный // dev.mysql.com [Сайт]. – URL: https://dev.mysql.com/doc/ (дата обращения: 11.04.2024).
7. Документация Nginx.  – Текст : электронный // ndinx.org [Сайт]. – URL: https://nginx.org/en/docs/ (дата обращения: 16.04.2024).
8. Securing a Web Application в Spring. – Текст : электронный // spring.io [Сайт]. – URL: https://spring.io/guides/gs/securing-web (дата обращения: 14.04.2024).
9. Accessing Data with JPA. – Текст : электронный // spring.io [Сайт]. – URL: https://spring.io/guides/gs/accessing-data-jpa (дата обращения: 14.04.2024).
10. Building REST services with Spring. – Текст: электронный // spring.io [Сайт]. – URL: https://spring.io/guides/tutorials/rest (дата обращения: 7.04.2024).
11. Отправка email в Spring. – Текст : электронный // spring.io [Сайт]. – URL: https://docs.spring.io/spring-boot/docs/2.0.x/reference/html/boot-features-email.html (дата обращения: 8.04.2024).

# Приложение А

Листинг А.1 – код контроллера UserRestController (для Java 17)

package com.example.demo.restApiControllers;  
  
import com.example.demo.moduls.User;  
import com.example.demo.services.UserService;  
import lombok.RequiredArgsConstructor;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.nio.charset.StandardCharsets;  
import java.security.Principal;  
import java.util.Base64;  
  
@RestController  
@RequiredArgsConstructor  
@RequestMapping(«/api»)  
public class UserRestController {  
 private final UserService userService;  
  
 @PostMapping(«/registration»)  
 public ResponseEntity<?> createUser(@RequestBody User user) {  
 if (!userService.createUser(user)) {  
 return ResponseEntity.*badRequest*().body(«Пользователь с почтой: « + user.getEmail() + « уже существует»);  
 }  
 String credentials = user.getEmail() + «:» + user.getPassword();  
 String encodedCredentials = Base64.*getEncoder*().encodeToString(credentials.getBytes(StandardCharsets.*UTF\_8*));  
  
 return ResponseEntity.*ok*(«Регистрация прошла успешно»);  
 }  
  
 @GetMapping(«/user/{user}»)  
 public ResponseEntity<?> userInfo(@PathVariable(«user») User user, Principal principal) {  
 if (userService.getUserByPrincipal(principal).getEmail().equals(user.getEmail())) {  
 return ResponseEntity.*ok*(user);  
 }  
 return ResponseEntity.*status*(403).body(«Доступ запрещен»);  
 }  
  
 @GetMapping(«/login»)  
 public ResponseEntity<?> login(Principal principal) {  
 if(principal==null){  
 return ResponseEntity.*status*(401).body(«Пользователь не найден»);  
 }  
 User user = userService.getUserByPrincipal(principal);  
 return ResponseEntity.*ok*(user);  
 }  
}

Листинг А.2 – код сервиса UserService (для Java 17)

package com.example.demo.services;  
  
  
import com.example.demo.moduls.User;  
import com.example.demo.moduls.enums.Role;  
import com.example.demo.repositoties.UserRepository;  
import lombok.RequiredArgsConstructor;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.stereotype.Service;  
  
import java.security.Principal;  
import java.util.\*;  
import java.util.stream.Collectors;  
  
@Service  
@Slf4j  
@RequiredArgsConstructor  
public class UserService {  
 private final UserRepository userRepository;  
 private final PasswordEncoder passwordEncoder;  
 private final MailService mailSender;  
 public boolean createUser(User user){  
 String email = user.getEmail();  
 if (userRepository.findByEmail(email)!= null) return false;  
 user.setActive(true);  
 user.setPassword(passwordEncoder.encode(user.getPassword()));  
 user.getRoles().add(Role.*ROLE\_ADMIN*);  
 user.setActivationCode(UUID.*randomUUID*().toString());  
 *log*.info(«Saving new User with Email: {}», email);  
 userRepository.save(user);  
 String message = String.*format*(  
 «Hello, %s! \n» +  
 «Welcome to Digital Departments. Please, visit next link: http://localhost:8080/activate/%s»,  
 user.getUsername(),  
 user.getActivationCode()  
 );  
 mailSender.send(user.getEmail(), «Activation code», message);  
 return true;  
 }  
 public List<User> list(){  
 return userRepository.findAll();  
 }  
  
 public void banUser(Long id) {  
 User user = userRepository.findById(id).orElse(null);  
 if (user!=null){  
 if(user.isActive()){  
 user.setActive(false);  
 }  
 else {  
 user.setActive(true);  
 }  
 *log*.info(«Ban user with id = {}; email: {}», user.getId(), user.getEmail());  
 }  
 userRepository.save(user);  
  
 }  
  
 public void changeUserRoles(User user, String role) {  
 Set<String> roles = Arrays.*stream*(Role.*values*())  
 .map(Role::name)  
 .collect(Collectors.*toSet*());  
 System.*out*.println(«Attempting to change role to: « + role);  
 user.getRoles().clear();  
 if (roles.contains(role)) {  
 user.getRoles().clear(); *// Очищаем текущие роли пользователя* user.getRoles().add(Role.*valueOf*(role)); *// Добавляем новую роль* System.*out*.println(«Role changed to: « + role);  
 }  
 userRepository.save(user);  
 }  
  
 public void saveUser(User user){  
 userRepository.save(user);  
 }  
  
 public User getUserByPrincipal(Principal principal) {  
 if (principal==null){  
 return new User();  
 }  
 return userRepository.findByEmail(principal.getName());  
 }  
  
 public boolean activateUser(String code) {  
 User user = userRepository.findByActivationCode(code);  
 if(user==null){  
 return false;  
 }  
 user.setActivationCode(null);  
 userRepository.save(user);  
 return true;  
 }  
  
 public User getUserById(Long userId) {  
 return userRepository.findById(userId).orElse(null);  
 }  
}

Листинг А.3 – код репозитория UserRepository (для Java 17)

package com.example.demo.repositoties;  
import com.example.demo.moduls.User;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface UserRepository extends JpaRepository<User, Long> {  
 User findByEmail(String email);  
  
 User findByActivationCode(String code);  
}

Листинг А.4 – класс User (для Java 17)

package com.example.demo.moduls;  
  
import com.example.demo.moduls.enums.Role;  
import com.fasterxml.jackson.annotation.JsonBackReference;  
import com.fasterxml.jackson.annotation.JsonIgnore;  
import com.fasterxml.jackson.annotation.JsonManagedReference;  
import jakarta.persistence.\*;  
import lombok.Data;  
import org.springframework.security.core.GrantedAuthority;  
import org.springframework.security.core.userdetails.UserDetails;  
  
import java.time.LocalDateTime;  
import java.util.\*;  
  
@Entity  
@Table(name = «users»)  
@Data  
public class User implements UserDetails {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 @Column(name = «id»)  
 private Long id;  
 @Column(name = «email», unique = true)  
 private String email;  
 @Column(name = «phone\_number»)  
 private String phoneNumber;  
 @Column(name = «name»)  
 private String name;  
 @Column(name=«first\_name»)  
 private String first\_name;  
 @Column(name=«last\_name»)  
 private String last\_name;  
 @Column(name = «father\_name»)  
 private String father\_name;  
 @Column(name = «groupUser»)  
 private String group;  
 @Column(name = «birthday»)  
 private String birthday;  
 @Column(name = «studentId»)  
 private String studentId;  
 @Column(name = «active»)  
 private boolean active;  
  
 private String activationCode;  
  
 @OneToOne(cascade = CascadeType.*ALL*, fetch = FetchType.*EAGER*)  
 @JoinColumn(name = «image\_id»)  
 private Image avatar;  
 @Column(name = «password», length = 1000)  
 private String password;  
  
 @ElementCollection(targetClass = Role.class, fetch = FetchType.*EAGER*)  
 @CollectionTable(name = «user\_role»,  
 joinColumns = @JoinColumn(name=«user\_id»))  
 @Enumerated(EnumType.*STRING*)  
 private Set<Role> roles = new HashSet<>();  
  
 private LocalDateTime dateOfCreated;  
 @JsonBackReference  
 @ManyToOne(cascade = CascadeType.*ALL*, fetch = FetchType.*LAZY*)  
 @JoinColumn  
 private Course course;  
  
 @JsonManagedReference  
 @OneToMany(cascade = CascadeType.*ALL*, fetch = FetchType.*LAZY*,  
 mappedBy = «user»)  
 private List<FAQ> FAQs = new ArrayList<>();  
  
 @JsonManagedReference  
 @OneToMany(cascade = CascadeType.*ALL*, fetch = FetchType.*LAZY*,  
 mappedBy = «user»)  
 private List<SupportTicket> SupportTicket = new ArrayList<>();  
 @PrePersist  
 private void init(){  
 dateOfCreated = LocalDateTime.*now*();  
 }  
  
 *//Security* public boolean isAdmin(){return roles.contains(Role.*ROLE\_ADMIN*);}  
 public boolean isUser(){return roles.contains(Role.*ROLE\_USER*);}  
  
  
 @Override  
 public Collection<? extends GrantedAuthority> getAuthorities() {  
 return roles;  
 }  
  
 @Override  
 public String getUsername() {  
 return email;  
 }  
  
 @Override  
 public boolean isAccountNonExpired() {  
 return true;  
 }  
  
 @Override  
 public boolean isAccountNonLocked() {  
 return true;  
 }  
  
 @Override  
 public boolean isCredentialsNonExpired() {  
 return true;  
 }  
  
 @Override  
 public boolean isEnabled() {  
 return active;  
 }  
}

Листинг А.5 – код класса для безопасности SecurityConfig (для Java 17)

package com.example.demo.configuration;  
  
import com.example.demo.services.CustomUserDetailsService;  
import lombok.RequiredArgsConstructor;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.security.authentication.AuthenticationManager;  
import org.springframework.security.authentication

.AuthenticationProvider;  
import org.springframework.security

.authentication.dao.DaoAuthenticationProvider;  
import org.springframework.security.config.annotation

.authentication.builders.AuthenticationManagerBuilder;  
import org.springframework.security.config.annotation

.method.configuration.EnableGlobalMethodSecurity;  
import org.springframework.security.config

.annotation.web.builders.HttpSecurity;  
import org.springframework.security.config

.annotation.web.configuration.EnableWebSecurity;  
import org.springframework.security

.core.userdetails.UserDetailsService;  
import org.springframework.security.crypto.bcrypt

.BCryptPasswordEncoder;  
import org.springframework.security.crypto

.password.PasswordEncoder;  
import org.springframework.security.web

.SecurityFilterChain;  
import org.springframework.security.web.util

.matcher.AntPathRequestMatcher;  
  
import static org.springframework.security.config.Customizer.*withDefaults*;  
  
@Configuration  
@RequiredArgsConstructor  
@EnableGlobalMethodSecurity(prePostEnabled = true)  
  
public class SecurityConfig {  
  
 private final CustomUserDetailsService userDetailService;  
  
 @Bean  
 public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
 http  
 .csrf(csrf -> csrf.disable()) *// Устаревший метод заменен на новый* .authorizeHttpRequests(authorizeRequests ->  
 authorizeRequests  
 .requestMatchers(«/», «/registration», «/activate/\*», «/api/login»,»/api/registration»)  
 .permitAll()  
 .requestMatchers(«/api/admin/test»).hasAuthority(«ROLE\_ADMIN»)  
 .anyRequest().authenticated()  
 )  
 .httpBasic(*withDefaults*()) *// Используем HTTP Basic Authentication* .logout(logout -> logout.permitAll());  
  
 return http.build();  
 }  
  
 @Bean  
 public UserDetailsService userDetailsService() {  
 return userDetailService;  
 }  
  
 @Bean  
 public PasswordEncoder passwordEncoder() {  
 return new BCryptPasswordEncoder();  
 }  
  
 @Bean  
 public AuthenticationManager authenticationManager(HttpSecurity http) throws Exception {  
 AuthenticationManagerBuilder authenticationManagerBuilder =  
 http.getSharedObject(AuthenticationManagerBuilder.class);  
 authenticationManagerBuilder  
 .userDetailsService(userDetailService)  
 .passwordEncoder(passwordEncoder());  
 return authenticationManagerBuilder.build();  
 }  
}

Листинг А.6 – код контроллера CourseRestController (для Java 17)

package com.example.demo.restApiControllers;  
  
import com.example.demo.moduls.Course;  
import com.example.demo.moduls.Lecture;  
import com.example.demo.services.CoursesService;  
import com.example.demo.services.LectureService;  
import com.example.demo.services.UserService;  
import lombok.RequiredArgsConstructor;  
import org.springframework.http.ResponseEntity;  
import org.springframework.security.access.prepost.PreAuthorize;  
import org.springframework.web.bind.annotation.\*;  
  
import java.io.IOException;  
import java.security.Principal;  
import java.util.List;  
  
@RestController  
  
@RequestMapping(«/api/courses»)  
@RequiredArgsConstructor  
public class CourseRestController {  
 private final CoursesService coursesService;  
 private final UserService userService;  
 private final LectureService lectureService;  
 @PreAuthorize(«hasRole('ROLE\_ADMIN')»)  
 @GetMapping  
 public ResponseEntity<List<Course>> getCourses(@RequestParam(name = «title», required = false) String title) {  
 List<Course> courses = coursesService.listCourses(title);  
 return ResponseEntity.*ok*(courses);  
 }  
  
 @GetMapping(«/{id}»)  
 public ResponseEntity<Course> getCourseInfo(@PathVariable Long id, Principal principal) {  
 Course course = userService.getUserByPrincipal(principal).getCourse();  
 if (course == null) {  
 return ResponseEntity.*notFound*().build();  
 }  
 return ResponseEntity.*ok*(course);  
 }  
 @PreAuthorize(«hasRole('ROLE\_ADMIN')»)  
 @PostMapping(«/create/test»)  
 public ResponseEntity<?> test(@RequestBody Course course) throws IOException {  
 coursesService.saveCourse(course);  
 return ResponseEntity.*ok*(course);  
 }  
  
 @PreAuthorize(«hasRole('ROLE\_ADMIN')»)  
 @PostMapping(«/create»)  
 public ResponseEntity<?> createCourse(@RequestBody Course course) throws IOException {  
 coursesService.saveCourse(course);  
 return ResponseEntity.*ok*(course);  
 }  
  
 @PreAuthorize(«hasRole('ROLE\_ADMIN')»)  
 @DeleteMapping(«/delete/{id}»)  
 public ResponseEntity<Void> deleteCourse(@PathVariable Long id) {  
 coursesService.deleteCourse(id);  
 return ResponseEntity.*noContent*().build();  
 }  
  
 @PreAuthorize(«hasRole('ROLE\_ADMIN')»)  
 @PostMapping(«/{id}/lecture/create»)  
 public ResponseEntity<Lecture> createLecture(@PathVariable Long id, @RequestBody Lecture lecture) {  
 Course course = coursesService.getCourseById(id);  
 if (course == null) {  
 return ResponseEntity.*notFound*().build();  
 }  
 lecture.setCourse(course);  
 lectureService.saveLecture(lecture);  
 return ResponseEntity.*ok*(lecture);  
 }  
  
 @PreAuthorize(«hasRole('ROLE\_ADMIN')»)  
 @DeleteMapping(«/{courseId}/lecture/delete/{lectureId}»)  
 public ResponseEntity<Void> deleteLecture(@PathVariable Long lectureId) {  
 lectureService.deleteLectures(lectureId);  
 return ResponseEntity.*noContent*().build();  
 }  
}

Листинг А.7 – код сервиса CourseService (для Java 17)

package com.example.demo.services;  
import com.example.demo.moduls.Course;  
import com.example.demo.moduls.User;  
import com.example.demo.moduls.Lecture;  
import com.example.demo.repositoties.CourseRepository;  
import com.example.demo.repositoties.UserRepository;  
import lombok.RequiredArgsConstructor;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.stereotype.Service;  
import java.io.IOException;  
import java.util.List;  
  
@Service  
@Slf4j  
@RequiredArgsConstructor  
public class CoursesService {  
 private final UserRepository userRepository;  
 private final CourseRepository courseRepository;  
  
 public List<Course> listCourses(String title){  
 if(title!=null) return courseRepository.findByTitle(title);  
 return courseRepository.findAll();  
 }  
  
 public List<Course> list(){  
 return courseRepository.findAll();  
 }  
 public void saveCourse(Course course) throws IOException{  
 Course CourseFromBd = courseRepository.save(course);  
  
 }  
 public void deleteCourse(long id){  
 courseRepository.deleteById(id);  
 }  
 public Course getCourseById(Long id) {  
 return courseRepository.findById(id).orElse(null);  
 }  
  
 public void addUserInCourse(User user, Course course){  
 course.addUsersInList(user);  
 courseRepository.save(course);  
 }  
  
 public void addLectureInCourse(Lecture lecture, Course course){  
 course.addLecturesInList(lecture);  
 courseRepository.save(course);  
 }  
}

Листинг А.8 – код репозитория CourseRepository (для Java 17)

package com.example.demo.repositoties;  
import com.example.demo.moduls.Course;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
import java.util.List;  
  
public interface CourseRepository extends JpaRepository<Course, Long>{  
 List<Course> findByTitle(String title);  
  
}

Листинг А.9 – код сервиса LectureService (для Java 17)

import com.example.demo.moduls.Course;  
import com.example.demo.moduls.Lecture;  
import com.example.demo.repositoties.CourseRepository;  
import com.example.demo.repositoties.LectureRepository;  
import lombok.RequiredArgsConstructor;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.stereotype.Service;  
  
import java.io.IOException;  
import java.util.List;  
  
@Service  
@Slf4j  
@RequiredArgsConstructor  
public class LectureService {  
 private final CourseRepository courseRepository;  
 private final LectureRepository lectureRepository;  
  
 public List<Lecture> listLectures(){ return lectureRepository.findAll();}  
  
 public void saveLecture(Lecture lecture){  
 lectureRepository.save(lecture);  
 }  
 public void deleteLectures(long id){lectureRepository.deleteById(id);}  
  
 public Lecture getLectureById(Long id){return lectureRepository.findById(id).orElse(null);}  
}

Листинг А.10 – код репозитория LectureRepository (для Java 17)

package com.example.demo.repositoties;  
  
import com.example.demo.moduls.Lecture;  
import org.springframework.data.jpa.repository.JpaRepository;  
public interface LectureRepository extends JpaRepository<Lecture, Long> {  
 Lecture findByNumberOfLectures(int numberOfLecture);  
}

Листинг А.11 – код класса Course (для Java 17)

package com.example.demo.moduls;  
import com.fasterxml.jackson.annotation.JsonIgnore;  
import com.fasterxml.jackson.annotation.JsonManagedReference;  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import java.time.LocalDateTime;  
import java.util.ArrayList;  
import java.util.List;  
  
  
@Entity  
@Table(name=«course»)  
@AllArgsConstructor  
@NoArgsConstructor  
@Data  
  
public class Course {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 @Column(name=«id»)  
 private Long id;  
 @Column(name=«title»)  
 private String title;  
 @Column(name=«description», columnDefinition = «text»)  
 private String description;  
 @JsonManagedReference  
 @OneToMany(cascade = CascadeType.*REFRESH*, fetch = FetchType.*LAZY*,  
 mappedBy = «course»)  
 private List<User> users = new ArrayList<>();  
 @JsonManagedReference  
 @OneToMany(cascade = CascadeType.*ALL*, fetch = FetchType.*LAZY*,  
 mappedBy = «course»)  
 private List<Lecture> lectures = new ArrayList<>();  
  
 private Long previewImageId;  
 private LocalDateTime dateOfCreated;  
  
 public void addUsersInList(User user){  
 users.add(user);  
 }  
 public void addLecturesInList(Lecture lecture){  
 lectures.add(lecture);  
 }  
  
 @PrePersist  
 private void init(){  
 dateOfCreated = LocalDateTime.*now*();  
 }  
}

Листинг А.12 – код перечисление Role (для Java 17)

package com.example.demo.moduls.enums;  
  
import org.springframework.security.core.GrantedAuthority;  
  
public enum Role implements GrantedAuthority {  
 *ROLE\_USER*, *ROLE\_ADMIN*,*ROLE\_TEACH*;  
  
 @Override  
 public String getAuthority() {  
 return name();  
 }  
}

Листинг А.13 – код контроллера AdminRestController (для Java 17)

package com.example.demo.restApiControllers;  
  
import com.example.demo.moduls.Course;  
import com.example.demo.moduls.User;  
import com.example.demo.moduls.enums.Role;  
import com.example.demo.services.CoursesService;  
import com.example.demo.services.UserService;  
import lombok.RequiredArgsConstructor;  
import org.springframework.http.ResponseEntity;  
import org.springframework.security.access.prepost.PreAuthorize;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
import java.util.Map;  
@RestController  
@PreAuthorize(«hasRole('ROLE\_ADMIN')»)  
@RequestMapping(«/api/admin»)  
@RequiredArgsConstructor  
  
public class AdminRestController {  
 private final UserService userService;  
 private final CoursesService coursesService;  
  
 @GetMapping  
 public ResponseEntity<Map<String, List<?>>> admin() {  
 return ResponseEntity.*ok*(Map.*of*(  
 «users», userService.list(),  
 «courses», coursesService.list()  
 ));  
 }  
  
 @PostMapping(«/user/ban/{id}»)  
 public ResponseEntity<Void> userBan(@PathVariable(«id») Long id) {  
 userService.banUser(id);  
 return ResponseEntity.*noContent*().build();  
 }  
  
 @PostMapping(«/user/edit/{userId}»)  
 public ResponseEntity<?> userEdit(@PathVariable(«userId») Long userId, @RequestBody User updatedUser) {  
 User user = userService.getUserById(userId);  
 user = updatedUser;  
 userService.saveUser(user);  
 return ResponseEntity.*status*(200).body(«Изменения пользователя прошли успешно»);  
 }  
  
  
 @PostMapping(«/user/edit/courses/{userId}»)  
 public ResponseEntity<?> addCourse(@PathVariable(«userId») Long userId, @RequestParam(«courseId») Long courseId) {  
 User user = userService.getUserById(userId);  
 Course course = coursesService.getCourseById(courseId);  
 if (user == null || course == null) {  
 return ResponseEntity.*notFound*().build();  
 }  
 user.setCourse(course);  
 userService.saveUser(user);  
 coursesService.addUserInCourse(user, course);  
 return ResponseEntity.*status*(200).body(«Курс успешно добавлен»);  
 }  
 @GetMapping(«/test»)  
 public String printMessage() {  
 String message = «Hello, this is a REST endpoint!»;  
 System.*out*.println(message);  
 return message;  
 }  
}

Листинг А.14 – код контроллера FAQController (для Java 17)

package com.example.demo.restApiControllers;

import com.example.demo.moduls.FAQ;

import com.example.demo.services.FAQService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

@RestController

@RequestMapping(«/api/faq»)

public class FAQController {

private final FAQService faqService;

@Autowired

public FAQController(FAQService faqService) {

this.faqService = faqService;

}

@PostMapping(«/answer»)

public ResponseEntity<String> getAnswer(@RequestParam int questionNumber) {

return faqService.getAnswerByQuestionNumber(questionNumber)

.map(faq -> ResponseEntity.ok(faq.getAnswer()))

.orElseGet(() -> ResponseEntity.notFound().build());

}

@PostMapping(«/add»)

public ResponseEntity<FAQ> addQuestion(@RequestBody FAQRequest request) {

FAQ newQuestion = faqService.addQuestion(request.getQuestion(), request.getAnswer());

return ResponseEntity.ok(newQuestion);

}

}

Листинг А.15 – код сервиса FAQService (для Java 17)

package com.example.demo.services;

import com.example.demo.moduls.FAQ;

import com.example.demo.repositories.FAQRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.Optional;

@Service

public class FAQService {

private final FAQRepository faqRepository;

@Autowired

public FAQService(FAQRepository faqRepository) {

this.faqRepository = faqRepository;

}

public Optional<FAQ> getAnswerByQuestionNumber(int questionNumber) {

return faqRepository.findByQuestionNumber(questionNumber);

}

}

Листинг А.16 – код репозитория FAQRepository (для Java 17)

package com.example.demo.repositories;

import com.example.demo.moduls.FAQ;

import org.springframework.data.jpa.repository.JpaRepository;

import java.util.Optional;

public interface FAQRepository extends JpaRepository<FAQ, Long> {

Optional<FAQ> findByQuestionNumber(int questionNumber);

Листинг А.17 – код класса FAQ (для Java 17)

package com.example.demo.moduls;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

@Entity

@Table(name = «faq»)

@Data

@NoArgsConstructor

@AllArgsConstructor

public class FAQ {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long id;

private int questionNumber;

private String question;

private String answer;

}

Листинг А.18 – код контроллера SupportTicketController (для Java 17)

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping(«/api/support-tickets»)

public class SupportTicketController {

private final SupportTicketService ticketService;

@Autowired

public SupportTicketController(SupportTicketService ticketService) {

this.ticketService = ticketService;

}

@PostMapping

public ResponseEntity<SupportTicket> createSupportTicket(@RequestBody SupportTicket ticket) {

SupportTicket createdTicket = ticketService.createSupportTicket(ticket);

return new ResponseEntity<>(createdTicket, HttpStatus.CREATED);

}

@GetMapping

public ResponseEntity<List<SupportTicket>> getAllSupportTickets() {

List<SupportTicket> tickets = ticketService.getAllSupportTickets();

return new ResponseEntity<>(tickets, HttpStatus.OK);

}

@PostMapping(«/{ticketId}/respond»)

public ResponseEntity<SupportTicket> respondToSupportTicket(

@PathVariable Long ticketId,

@RequestParam String adminResponse

) {

SupportTicket ticket = ticketService.respondToSupportTicket(ticketId, adminResponse);

return new ResponseEntity<>(ticket, HttpStatus.OK);

}

@PostMapping(«/{ticketId}/close»)

public ResponseEntity<SupportTicket> closeSupportTicket(@PathVariable Long ticketId) {

SupportTicket ticket = ticketService.closeSupportTicket(ticketId);

return new ResponseEntity<>(ticket, HttpStatus.OK);

}

}

Листинг А.19 – код репозитория SupportTicketRepository (для Java 17)  
import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import com.example.demo.moduls.SupportTicket;

@Repository

public interface SupportTicketRepository extends JpaRepository<SupportTicket, Long> {

}

Листинг А.20 – код сервиса SupportTicketService (для Java 17)

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class SupportTicketService {

private final SupportTicketRepository ticketRepository;

@Autowired

public SupportTicketService(SupportTicketRepository ticketRepository) {

this.ticketRepository = ticketRepository;

}

public SupportTicket createSupportTicket(SupportTicket ticket) {

ticket.setCreationDateTime(LocalDateTime.now());

return ticketRepository.save(ticket);

}

public List<SupportTicket> getAllSupportTickets() {

return ticketRepository.findAll();

}}