

Geekbrains

**Разработка веб-приложения для автоматизации оценочного
процесса в экзаменационных комиссиях с использованием
клиент-серверной архитектуры и применением технологий
Vue.js, Java Spring и MySQL**

Программа: Разработчик

Специализация: Веб-разработка на Java

Абубакиров Рустем Анварович

Сургут

2024 г.

Содержание

1.	Введение	3
1.1.	Описание проекта	3
1.2.	Цель и задачи	4
2.	Исследование методологий разработки и выбор архитектуры	5
2.1.	Обзор методологий разработки (Agile, Waterfall и др.)	5
2.2.	Анализ требований и функциональности приложения	7
2.3.	Обоснование выбора клиент-серверной архитектуры	9
2.4.	Обоснование выбора методологии	12
3.	Выбор технологий и инструментов разработки	15
3.1.	Обзор технологий Vue.js, Java Spring и MySQL	15
3.2.	Сравнительный анализ с альтернативными технологиями	16
3.3.	Обоснование выбора технологий и инструментов	17
4.	Проектирование архитектуры приложения	19
4.1.	Подготовка данных для UML-диаграммы	19
4.2.	Разработка Use-Case диаграммы	21
4.3.	Разработка ERD	24
4.4.	Планирование Customer Journey Map (CJM)	25
4.5.	Выбор паттерна проектирования	26
4.6.	Схема клиент-серверной архитектуры	27
5.	Разработка веб-приложения	29
5.1.	Разработка клиентской части приложения на Vue.js	29
5.1.1.	Создание интерфейса для ввода данных перед экзаменом	30
5.1.2.	Реализация возможности экзаменатора оценивать выполнение экзаменуемыми этапов и формирование протокола экзамена	35
5.2.	Разработка базы данных MySQL для хранения информации об экзаменуемых, экзаменаторах, специальностях, этапах и баллах	37

5.3.	Создание серверной части с помощью Java Spring	45
5.3.1.	Создание RESTful API для взаимодействия с клиентской частью	47
5.3.2.	Реализация логики подсчета баллов и формирования протокола экзамена	58
5.4.	Интеграция клиентской и серверной частей приложения	61
6.	Тестирование и отладка	63
6.1.	Проведение модульных и интеграционных тестов	63
6.2.	Отладка ошибок и исправление дефектов	65
7.	Разработка документации	66
7.1.	Описание функциональности приложения	66
7.2.	Инструкция по установке и запуску приложения	67
7.3.	Техническая документация	68
7.4.	Форматирование и оформление документации	69
8.	Заключение	70
8.1.	Основные результаты проекта	70
8.2.	Выводы и рекомендации по дальнейшему развитию приложения	71
9.	Список используемой литературы	72
10	Приложения	

1. Введение

1.1. Описание проекта

В современном мире автоматизация играет ключевую роль в повышении эффективности и улучшении качества различных процессов. В рамках медицинской сферы одним из важных аспектов является оценка практических навыков медицинских специалистов, проводимая экзаменационными комиссиями. Однако проведение таких экзаменов вручную требует значительных временных и человеческих ресурсов, а также подвержено риску ошибок.

В данном дипломном проекте представляется разработка веб-приложения для автоматизации оценочного процесса в экзаменационных комиссиях Сургутского симуляционно-тренингового центра, обеспечивающего проведение аккредитации медицинских специалистов на базе медицинского института Сургутского государственного университета.

Целью проекта является создание инструмента, который позволит медицинским экспертам эффективно проводить оценку практических навыков экзаменуемых, минимизируя время, затраченное на подсчет баллов, и исключая возможность ошибок при составлении протокола проведенного экзамена.

Основными функциональными требованиями к разрабатываемому приложению являются возможность ввода отметок экзаменатора в режиме реального времени, автоматический подсчет баллов в соответствии с заданными правилами, формирование протокола проведенного экзамена и сохранение всех данных в базе данных для последующего анализа.

Для реализации проекта выбрана клиент-серверная архитектура с использованием технологий Vue.js для разработки клиентской части, Java Spring для серверной части и MySQL для хранения данных. Этот выбор обусловлен необходимостью обеспечения высокой производительности, масштабируемости и надежности приложения.

В дальнейшем в данной работе будут подробно рассмотрены методологии разработки, выбранные архитектурные решения, а также процесс разработки и тестирования приложения. Ожидается, что результаты данного проекта помогут существенно улучшить процесс оценки практических навыков медицинских специалистов и повысить качество обучения в медицинских учебных заведениях.

1.2. Цель и задачи

Тема проекта: Разработка веб-приложения для автоматизации оценочного процесса в экзаменационных комиссиях.

Цель: Создание инструмента, позволяющего медицинским экспертам проводить оценку практических навыков экзаменуемых с минимальными временными затратами и исключением возможных ошибок.

Проблема, которую решает проект: Ручной подсчет баллов и составление протокола экзамена занимает много времени и может привести к неточным результатам оценки.

Задачи проекта:

1. Разработать веб-приложение с удобным интерфейсом для ввода отметок экзаменатора.
2. Реализовать автоматический подсчет баллов в соответствии с заданными правилами.
3. Создать механизм формирования протокола проведенного экзамена.
4. Обеспечить сохранение всех данных в базе данных для последующего анализа.

Инструменты:

Для разработки приложения будут использованы современные инструменты и технологии программирования, такие как Vue.js для клиентской части, Java Spring для серверной части и MySQL для хранения данных.

Для тестирования приложения будут использоваться современные инструменты автоматизированного тестирования, такие как Selenium для функционального тестирования пользовательского интерфейса, JUnit для модульного тестирования серверной части и Postman для тестирования API.

Кроме того, для ручного тестирования и обеспечения качества продукта будет использоваться ручное тестирование.

Выполняемые роли при реализации данного проекта:

руководитель проекта, разработчик frontend (Vue.js), разработчик backend (Java Spring), специалист по базам данных (MySQL), QA-инженер, тестировщик.

2. Исследование методологий разработки и выбор архитектуры

2.1. Обзор методологий разработки (Agile, Waterfall и др.)

Современная среда разработки программного обеспечения предлагает различные методологии и фреймворки, которые учитывают гибкость и требования проекта. Среди них особенно выделяются Agile и Waterfall.

Agile

Методология Agile не только предлагает гибкий подход к разработке, но и акцентирует внимание на взаимодействии и сотрудничестве между членами команды. Agile включает в себя несколько подходов, таких как Scrum, Kanban, Extreme Programming (XP) и др. Каждый из этих подходов предлагает свои методы и принципы для эффективного управления проектом.

Данный подход актуален, когда на начальном этапе разработки отсутствует полное и точное видение конечного продукта. Продукт может дорабатываться в ходе разработки, однако подход ориентирован на квалифицированную команду, замена членов команды в ходе реализации проекта может вызывать значительные издержки.

Waterfall

Методология Waterfall в отличие от методологии Agile, представляет собой последовательный и линейный процесс разработки. Этот подход состоит из ряда фаз, включая сбор требований, проектирование, разработку, тестирование и внедрение. Фазы выполняются последовательно, и каждая завершается, прежде чем начнется следующая.

Данный подход актуален, когда в начале проекта имеется полное и точное представление о конечном продукте. Изменения в ходе реализации подхода обходятся большими издержками. В то же время замена специалистов в ходе проекта не критична.

Другие методологии и модели разработки:

V-образная модель:

Модель V-образной разработки представляет собой последовательную модель, которая связывает каждый этап разработки с соответствующим этапом тестирования. По мере продвижения вниз по левой стороне "V" выполняются этапы анализа, проектирования и разработки, а по мере движения вверх по правой стороне "V" выполняются тестирование, верификация и валидация.

Спиральная модель:

Спиральная модель разработки объединяет элементы последовательного и итеративного разработки, позволяя на каждом этапе разработки возвращаться к предыдущим этапам для уточнения и обновления требований и дизайна. Основным преимуществом спиральной модели является возможность управления рисками и изменениями в процессе разработки.

Инкрементная модель:

Инкрементная модель разработки включает в себя последовательное создание итераций или инкрементов продукта. Каждая итерация добавляет новые функции или улучшения к предыдущей версии продукта. Этот подход позволяет быстрее предоставить базовый функционал и затем постепенно расширять его.

Итеративная модель:

Итеративная модель разработки включает в себя повторяющиеся циклы разработки, где каждый цикл представляет собой полный процесс разработки, включая анализ, проектирование, реализацию и тестирование. Каждая итерация обычно добавляет новые функции или улучшения к предыдущей версии продукта. Каждая методология имеет свои сильные и слабые стороны, и выбор конкретной зависит от требований и особенностей проекта. Критически важно выбрать подход, который наилучшим образом соответствует потребностям команды и целям проекта.

2.2. Анализ требований и функциональности приложения

Основные требования:

Управление экзаменами:

Система должна обеспечивать возможность создания, редактирования и удаления экзаменов, включая информацию о дате, времени, месте проведения, специальности экзаменуемого и номере этапа.

Управление экзаменуемыми:

Приложение должно позволять добавлять, редактировать и удалять данные о экзаменуемых, включая их фамилию, данные о специальности и другую релевантную информацию.

Управление экзаменаторами:

Необходимо обеспечить возможность управления списком экзаменаторов, включая добавление, редактирование и удаление данных о них.

Формирование протоколов экзаменов:

Система должна автоматически формировать протоколы проведенных экзаменов на основе данных, введенных экзаменаторами, включая информацию о фамилии экзаменуемого, экзаменаторе, специальности, дате, времени, номере этапа, варианте и итоговом балле.

Автоматизированный подсчет баллов:

Приложение должно автоматически подсчитывать баллы на основе отметок, выставленных экзаменаторами, и формировать итоговый результат экзамена.

Интерфейс пользователя:

Интерфейс приложения должен быть интуитивно понятным и удобным в использовании для экзаменаторов. Он должен предоставлять удобные инструменты для ввода данных и просмотра результатов.

Безопасность:

Приложение должно обеспечивать безопасность данных, включая защиту от несанкционированного доступа и возможность резервного копирования.

Функциональность приложения:

Ввод данных перед экзаменом:

Пользовательский интерфейс предоставляет форму для ввода данных перед экзаменом.

Используются текстовые поля для ввода фамилии экзаменуемого и выбора специальности.

Для выбора номера этапа и варианта предусмотрены выпадающие списки.

Оценивание выполнения заданий:

Экзаменаторы имеют доступ к интерфейсу, где они могут проставлять отметки о выполнении или невыполнении каждого этапа задания.

Для каждого этапа предусмотрен чек-бокс для отметки выполнения.

Автоматический подсчет баллов:

Приложение автоматически подсчитывает баллы на основе отметок, выставленных экзаменаторами.

Используется скрипт для обработки данных и расчета итогового балла, который отображается в соответствующем поле интерфейса.

Формирование протоколов экзаменов:

После завершения экзамена система автоматически формирует протоколы проведенных экзаменов.

Используется шаблон протокола, в который подставляются данные об экзаменуемом, экзаменаторе, специальности, дате, времени, номере этапа, варианте и итоговом балле.

Управление данными:

Для управления данными о студентах, экзаменаторах и других сущностях используются соответствующие формы и списки.

Используются кнопки для добавления, редактирования и удаления записей.

Для обеспечения безопасности данных предусмотрены соответствующие механизмы аутентификации и авторизации.

2.3. Обоснование выбора клиент-серверной архитектуры

Преимущества клиент-серверной архитектуры:

В сравнении с desktop-приложениями клиент-серверная архитектура предлагает следующие преимущества:

Универсальный доступ в локальной сети:

Клиент-серверное приложение может быть доступно для всех пользователей в локальной сети, обеспечивая удобство использования без необходимости установки на каждом компьютере.

Централизованное управление данными:

Данные хранятся и обрабатываются на сервере, что обеспечивает их централизованное управление и защиту от потери.

Удобство обновлений:

Обновления приложения и данных могут быть легко внедрены централизованно на сервере, что упрощает поддержку и обновление всей системы.

Безопасность данных:

Централизованное хранение данных на сервере обеспечивает более высокий уровень безопасности, так как доступ к ним может быть контролируемым и ограниченным.

Трудности, при использовании Desktop-приложения:

Desktop-приложения представляют собой программные приложения, устанавливаемые и запускаемые на компьютере пользователя, в то время как веб-приложения работают через веб-браузер и доступны через локальную сеть или Интернет. Рассмотрим основные различия и недостатки desktop-приложений по сравнению с веб-приложениями:

Установка и обновление:

Desktop-приложения требуют установки на компьютер пользователя, что может потребовать дополнительных усилий со стороны пользователя и администратора системы. В отличие от этого, веб-приложения доступны через браузер и не требуют установки на устройство пользователя.

Платформозависимость:

Desktop-приложения, как правило, разрабатываются под определенную операционную систему (например, Windows, macOS, Linux), что делает их платформозависимыми. Веб-приложения, напротив, доступны из любого браузера на различных операционных системах, обеспечивая платформонезависимость.

Обновления и поддержка:

Desktop-приложения требуют регулярных обновлений и поддержки со стороны разработчиков, так как они установлены на компьютере пользователя. Веб-приложения, с другой стороны, могут обновляться централизованно на сервере, что обеспечивает легкость и быстроту обновлений для всех пользователей.

Доступность и распространение:

Веб-приложения доступны любому пользователю в локальной сети либо через Интернет и могут быть легко распространены и использованы пользователем с любого устройства в локальной сети, либо с доступом в Интернет. Desktop-приложения требуют установки на каждом устройстве, что может быть неудобно и требовать дополнительных действий со стороны пользователя.

Обновления данных:

В веб-приложениях обновления данных и функциональности могут быть прозрачно внедрены без необходимости обновления самого приложения на компьютере пользователя. В desktop-приложениях необходимо выпускать новые версии и обновления для внесения изменений.

Безопасность:

Desktop-приложения могут быть более уязвимы для вирусов и вредоносного программного обеспечения, поскольку они установлены непосредственно на устройстве пользователя. Веб-приложения защищены на уровне сервера и могут быть обновлены централизованно для устранения уязвимостей.

В целом, desktop-приложения имеют ряд недостатков по сравнению с веб-приложениями, таких как необходимость установки, платформозависимость, сложности обновления и распространения, а также потенциальные проблемы с безопасностью. Веб-приложения, с другой стороны, обеспечивают более простую установку и обновление, платформонезависимость, легкость распространения и более высокий уровень безопасности.

Вывод:

Клиент-серверная архитектура, таким образом, является оптимальным выбором для разработки приложения, предназначенного для автоматизации оценочного процесса в экзаменационных комиссиях, учитывая особенности использования такого приложения.

2.4. Обоснование выбора методологии

Выбор методологии разработки является ключевым аспектом успешной реализации проекта. Для данного проекта выбирается каскадная модель разработки с учетом особенностей задачи и доступных ресурсов.

Основные причины выбора каскадной модели:

Предсказуемость процесса:

Каскадная модель предполагает последовательное выполнение этапов разработки, что позволяет предсказать сроки и затраты на каждый этап проекта. Это особенно важно при работе с жесткими сроками, как в данном случае.

Простота планирования и управления:

Каскадная модель обеспечивает четкое разделение работы на этапы, что упрощает планирование и управление процессом разработки. Каждый этап имеет определенные цели и результаты, что облегчает оценку прогресса и контроль качества.

Минимизация рисков:

Каскадная модель позволяет выявить и решить проблемы на ранних этапах разработки, благодаря четкому планированию и последовательному выполнению этапов. Это помогает минимизировать риски возникновения проблем в дальнейшем и обеспечивает более высокое качество конечного продукта.

Оптимизация ресурсов:

Каскадная модель позволяет оптимизировать использование ресурсов, так как каждый этап разработки выполняется после завершения предыдущего. Это позволяет сосредоточить усилия на конкретных задачах и эффективно использовать доступные ресурсы.

Соответствие требованиям проекта:

Учитывая заданные сроки и критерии успешного завершения проекта, каскадная модель разработки представляется наиболее подходящей методологией для обеспечения достижения поставленных целей в рамках имеющихся ограничений.

План выполнения проекта по каскадной модели:

1. Подготовительный этап (1 день) – 08.04
 - 1.1 Описание проекта
 - 1.2 Цель и задачи
2. Исследование методологий разработки и выбор архитектуры (1 день) – 09.04
 - 2.1 Обзор методологий разработки (Agile, Waterfall и др.)
 - 2.2 Анализ требований и функциональности приложения
 - 2.3 Обоснование выбора клиент-серверной архитектуры
 - 2.4 Обоснование выбора методологии
3. Выбор технологий и инструментов разработки (1 день) – 10.04
 - 3.1 Обзор технологий Vue.js, Java Spring и MySQL
 - 3.2 Сравнительный анализ с альтернативными технологиями
 - 3.3 Обоснование выбора технологий и инструментов
4. Проектирование архитектуры приложения (2 дня) – 11.04-12.04
 - 4.1 Подготовка данных для UML-диаграммы
 - 4.2 Разработка Use-Case диаграммы
 - 4.3 Разработка ERD
 - 4.4 Планирование Customer Journey Map (CJM)
 - 4.5 Выбор паттерна проектирования
 - 4.6 Схема клиент-серверной архитектуры
5. Разработка веб-приложения (2 недели) – 12.04-26.04
 - 5.1 Проектирование пользовательского интерфейса с использованием Vue.js
 - 5.2 Создание серверной части с помощью Java Spring
 - 5.2.1 Создание RESTful API для взаимодействия с клиентской частью
 - 5.2.2 Реализация логики подсчета баллов и формирования протокола экзамена

5.3 Разработка клиентской части приложения на Vue.js

5.3.1 Создание интерфейса для ввода данных перед экзаменом

5.3.2 Реализация возможности экзаменатора оценивать выполнение экзаменуемым этапов и формирование протокола экзамена

5.4 Разработка базы данных MySQL для хранения информации о экзаменуемых, экзаменаторах, специальностях, этапах и баллах

5.5 Интеграция клиентской и серверной частей приложения

6. Тестирование и отладка (3 дня) – 27.04-29.04

6.1 Проведение модульных и интеграционных тестов

6.2 Отладка ошибок и исправление дефектов

7. Разработка документации (4 дня) – 30.04-03.05

7.1 Описание функциональности приложения

7.2 Инструкция по установке и запуску приложения

7.3 Техническая документация

7.4 Форматирование и оформление документации

8. Заключение (1 день) – 04.05

8.1 Основные результаты проекта

8.2 Выводы и рекомендации по дальнейшему развитию приложения

9. Список используемой литературы (1 день) – 05.05

10. Приложения (1 день) – 06.05

10.1 Код приложения

10.2 Дополнительные материалы

Общее время выполнения проекта: 5 недель.

3. Выбор технологий и инструментов разработки

3.1. Обзор технологий Vue.js, Java Spring и MySQL

Фреймворк Vue.js: Vue.js - это прогрессивный JavaScript-фреймворк, который отлично подходит для создания интерактивных пользовательских интерфейсов. Он предоставляет удобные инструменты для создания одностраничных приложений (SPA) и основан на компонентном подходе к разработке. Vue.js позволяет создавать и многократно использовать компоненты, которые легко интегрируются в приложение. Он также обладает простым синтаксисом и небольшим размером, что делает его отличным выбором для создания клиентской части веб-приложения.

Фреймворк Java Spring: Java Spring - это один из самых популярных фреймворков для разработки веб-приложений на языке Java. Он предоставляет широкий спектр инструментов и библиотек для создания масштабируемых и надежных приложений. Spring включает в себя модули для управления транзакциями, безопасностью, взаимодействия с базами данных и многое другое. Одним из наиболее распространенных модулей Spring является Spring Boot, который обеспечивает простоту и скорость развертывания приложений.

СУБД MySQL: MySQL - это одна из самых популярных систем управления реляционными базами данных (СУБД). Она обладает высокой производительностью, надежностью и масштабируемостью, что делает ее привлекательным выбором для хранения данных в веб-приложениях. MySQL поддерживает SQL-запросы, индексы, транзакции и множество других функций, необходимых для эффективной работы с данными.

Таким образом, использование Фреймворка Vue.js для клиентской части приложения позволит создать динамичный и отзывчивый пользовательский интерфейс. Java Spring будет обеспечивать мощную и надежную серверную часть приложения, а MySQL - эффективное хранилище данных.

Данные технологии в совокупности позволяют обеспечить разработку высококачественного и производительного веб-приложения, соответствующего предъявленным требованиям.

3.2. Сравнительный анализ с альтернативными технологиями

Проведем сравнительный анализ выбранных технологий с альтернативными. Ниже приведены результаты анализа основных альтернативных технологий:

Фронтенд-фреймворк:

Vue.js: Высокая популярность, простота в изучении и использовании, богатая функциональность, хорошая документация и активное сообщество разработчиков.

React: Широко используется в индустрии, эффективно работает с большими проектами, однако требует больше времени на изучение и имеет более строгий подход к архитектуре приложения.

Angular: Мощный и обширный фреймворк, предоставляющий полный стек решений для разработки веб-приложений, но может быть избыточен для данного проекта и требует больше времени на изучение.

Бэкенд-фреймворк:

Java Spring: Широко используется в индустрии, обладает большим сообществом разработчиков и хорошей документацией, предоставляет множество готовых решений для различных задач веб-разработки.

Node.js (Express.js): Популярен в разработке веб-приложений, особенно в сфере разработки API, легко масштабируется и поддерживается множеством библиотек, но может быть менее привычным для разработчиков, не имеющих опыта работы с JavaScript.

Python (Django, Flask): Прост в изучении, имеет много готовых решений и библиотек, но может быть менее эффективным для высоконагруженных приложений и менее подходит для проектов, требующих жесткой типизации данных.

Система управления базами данных:

MySQL: Широко используется в веб-разработке, стабильный, масштабируемый и хорошо документированный, подходит для множества приложений различной сложности.

PostgreSQL: Предоставляет более расширенные возможности по сравнению с MySQL, такие как поддержка JSON, полнотекстовый поиск и географические запросы, но может быть избыточным для данного проекта и требует больше ресурсов для поддержки.

3.3. Обоснование выбора технологий и инструментов

При выборе технологий и инструментов для разработки веб-приложения для автоматизации оценочного процесса в экзаменационных комиссиях были учтены следующие факторы и обстоятельства:

Знакомство с технологиями:

Команда разработчиков имеет опыт работы с технологиями Vue.js, Java Spring и MySQL. Это позволит ускорить процесс разработки, так как команда уже знакома с основами данных технологий.

Гибкость и производительность:

Vue.js был выбран для разработки клиентской части приложения благодаря его гибкости, простоте использования и возможности создания высокопроизводительных SPA. Java Spring обеспечит мощную и гибкую серверную часть приложения, обеспечивая высокую производительность и масштабируемость. MySQL выбрана в качестве базы данных из-за ее простоты в использовании, распространенности и надежности.

Интеграция и совместимость:

Vue.js и Java Spring хорошо интегрируются друг с другом, обеспечивая гладкую работу всего стека технологий. Это обеспечивает удобство в разработке, отладке и поддержке приложения.

Безопасность:

Java Spring обладает встроенными механизмами безопасности, такими как Spring Security, что делает его привлекательным выбором для разработки безопасных веб-приложений. Это особенно важно для приложения, обрабатывающего конфиденциальные данные, такие как результаты экзаменов.

Поддержка сообщества:

Vue.js, Java Spring и MySQL имеют активные сообщества разработчиков и обширные ресурсы, такие как документация, форумы поддержки и т. д. Это обеспечивает доступ к знаниям и помощи при возникновении проблем или вопросов в процессе разработки приложения.

Широкий выбор инструментов и библиотек:

Vue.js и Java Spring имеют обширные экосистемы инструментов и библиотек, которые могут ускорить разработку и обеспечить высокое качество кода. Существует множество плагинов и расширений для Vue.js, а Java Spring предлагает богатый выбор инструментов для разработки RESTful API, обработки данных и многое другое.

Принятый для реализации проекта стек технологий:

Исходя из проведенного сравнительного анализа технологий и инструментов, а также учитывая текущий опыт команды разработки, для реализации проекта был принят следующий стек технологий:

- Клиентская часть приложения (фронтенд) будет разработана с использованием фреймворка Vue.js, который обеспечивает гибкость, производительность и простоту в создании интерактивного пользовательского интерфейса.
- Серверная часть приложения (бэкенд) будет основана на фреймворке Java Spring, который обеспечивает высокую производительность, гибкость и безопасность. Java Spring также обладает широким набором инструментов для разработки RESTful API, обработки данных и управления бизнес-логикой приложения.
- В качестве системы управления базами данных будет использоваться MySQL, так как она обеспечивает надежное хранение и управление данными, а также широко распространена и поддерживается в индустрии.

Данный стек выбран из-за его соответствия требованиям проекта, гибкости, производительности, безопасности и удобства в разработке. Кроме того, учитывая опыт команды разработки с этими технологиями, их выбор обеспечивает эффективное выполнение поставленных задач в рамках ограниченных сроков.

4. Проектирование архитектуры приложения

4.1. Подготовка данных для UML-диаграммы

Язык UML является важным инструментом для определения и описания взаимодействия пользователей с разрабатываемой системой, их целей и функциональных возможностей системы.

Для того, чтобы разработать UML-диаграмму использования приложения, требуется понимание сценариев взаимодействия пользователей с разрабатываемой системой.

Опишем процесс проведения экзамена.

Основные элементы процесса:

Экзаменатор - Пользователь, ответственный за оценку экзаменуемых и взаимодействие с интерфейсом для ввода результатов оценки, а также для получения отображения заданий и результатов оценки, последующим сохранением результата экзамена, распечаткой протокола экзамена.

Экзаменуемый: Пользователь, выполняющий экзаменационный сценарий и не взаимодействующий с интерфейсом приложения.

Станция – Помещение с обстановкой, имитирующей обучающую ситуацию (кейс), возможную в медицинской практике, в соответствии с заданием, в которой экзаменуемый отрабатывает экзаменационный сценарий.

Паспорт станции – в контексте разработки приложения под паспортом станции понимается базовый сценарий, который должен быть отыгран на станции экзаменуемым.

Ситуация - вариант возможной ситуации в рамках базового сценария станции. Как правило, базовый сценарий станции подразумевает несколько возможных вариантов ситуации.

Основные действия:

- Экзаменатор входит в систему, используя свои учетные данные;
- Экзаменатор вводит ФИО экзаменуемого;
- Экзаменатор выбирает из выпадающего списка специальностей, специальность экзаменуемого;
- Экзаменатор выбирает вариант задания (после выбора варианта, список экзаменационных вопросов подгружается из базы данных и отображается в рабочем поле интерфейса. Напротив каждого вопроса находятся по 2 чек-бокса, означающие: первый – да(выполнил этап), второй – нет(не выполнил этап). Одновременно нельзя отметить оба чек-бокса. Только один любой чек-бокс может быть отмечен или оба чек-бокса могут быть не отмечены);
- Экзаменатор выбирает паспорт (номер) станции;
- Экзаменатор следит за исполнением заданий(этапов), которые должен выполнить экзаменуемый, в соответствии со сценарием;
- При выполнении задания(этапа) сценария, экзаменатор в интерфейсе приложения отмечает чек-бокс "да", при невыполнении или неправильном выполнении - чек-бокс "нет";
- После завершения экзаменуемым сценария, экзаменатор нажимает в приложении кнопку "Результат", приложение суммирует баллы этапов с нажатыми чек-боксами "да" и отнимает баллы этапов с нажатыми чек-боксами "нет";
- Если экзаменатор пропустил этап и не отметил ни один чек-бокс, выводится сообщение о том, что не все этапы проверены, не отмеченные этапы подсвечиваются;
- После получения и отображения результирующего балла Экзаменатор сохраняет результаты оценки экзамена в базе данных, путем нажатия на кнопку "Сохранить" - выводится окно, требующее подтвердить сохранение, после подтверждения результаты, включая отметки о выполнении этапов, сохраняются в базе данных. Если нет, то идет возврат к предыдущему состоянию;
- Экзаменатор генерирует протокол экзамена с результатами оценки, который можно отправить на печать или скачать в формате pdf;

4.2. Разработка Use-Case диаграммы

Основные сценарии использования (UseCases):

Экзаменатор:

Вход в систему:

Экзаменатор входит в систему, используя свои учетные данные.

Ввод информации об экзаменуемом:

- Экзаменатор вводит ФИО экзаменуемого.
- Выбирает специальность экзаменуемого.

Выбор варианта задания:

- Выбирает паспорт (номер) станции.
- Экзаменатор выбирает ситуацию (вариант) задания из списка.

Оценка выполнения заданий:

- Экзаменатор следит за исполнением экзаменуемым заданий (этапов) и отмечает выполненные этапы.

Подсчет и сохранение результатов:

- После завершения экзаменуемым сценария, экзаменатор подсчитывает баллы и сохраняет результаты оценки экзамена в базе данных.

Генерация протокола экзамена:

- Экзаменатор генерирует протокол экзамена с результатами оценки для дальнейшего использования.

Экзаменуемый:

Выполнение экзаменационного сценария:

Экзаменуемый выполняет предложенный сценарий.

Эти Use-Cases описывают основные действия, которые выполняют пользователи в системе. Таким образом, на диаграмме должны быть следующие элементы и связи:

Акторы:

Экзаменатор

Экзаменуемый

Use-Cases:

Вход в систему

Ввод информации об экзаменуемом

Выбор варианта задания

Оценка выполнения заданий

Подсчет и сохранение результатов

Генерация протокола экзамена

Выполнение экзаменационного сценария

Отношения:

Связь между акторами и Use-Cases должна быть представлена стрелками, указывающими на Use-Cases, которые актор может выполнять.

Расположение:

Акторы обычно располагаются слева, а Use-Cases - справа. Стрелки указывают направление действия от акторов к Use-Cases.

Следующим шагом будет создание Use-Case-диаграммы для наглядного представления взаимодействия между акторами и функциональностью системы.

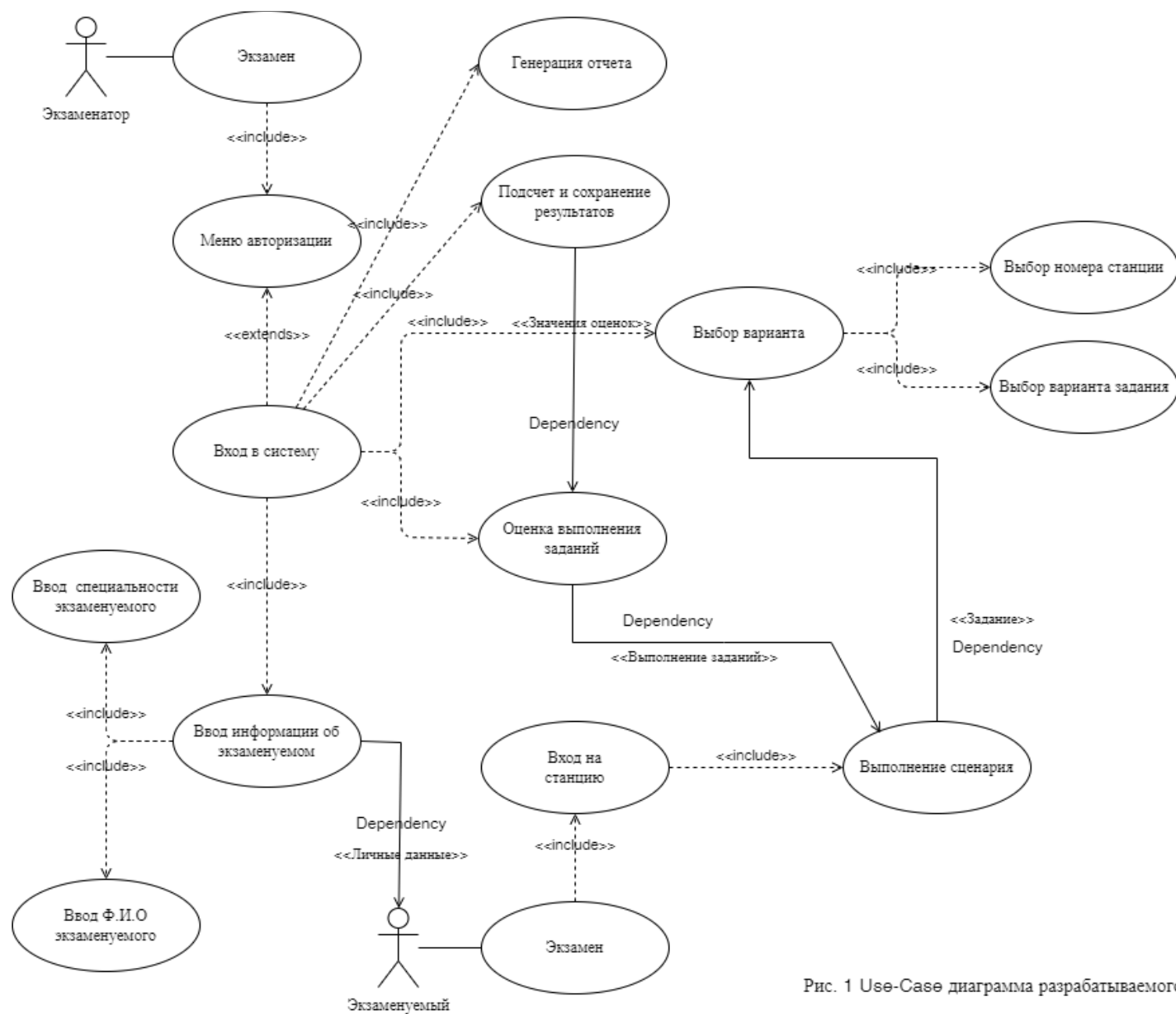


Рис. 1 Use-Case диаграмма разрабатываемого приложения

4.3. Разработка ERD

ERD (Entity-RelationshipDiagram) - это тип диаграммы в области баз данных, который используется для визуального представления структуры и взаимосвязей между сущностями (entities) в базе данных.

ERD помогает разработчикам и аналитикам понять структуру базы данных, ее основные компоненты и взаимосвязи между ними. Это полезный инструмент при проектировании, моделировании и анализе баз данных.

ERD состоит из сущностей (таблиц), атрибутов (столбцов) и связей между этими сущностями.

Главные компоненты ERD включают:

Сущности (Entities): Представляют основные объекты или понятия, для которых хранится информация в базе данных. Каждая сущность обычно представляется как прямоугольник с названием сущности внутри.

Атрибуты (Attributes): Показывают характеристики сущностей, то есть информацию, которая хранится о каждой сущности. Атрибуты могут отображаться как овалы, связанные с соответствующей сущностью. Мы будем использовать другое отображение атрибутов, атрибуты будем отображать также в прямоугольнике, расположенном под отображением сущности.

Отношения (Relationships): Описывают связи между различными сущностями. Они могут быть однонаправленными или двунаправленными, а также могут иметь разные типы, такие как один к одному, один ко многим, многие к одному и многие ко многим.

Разработку ERD, для целей нашего приложения, начнем с определения необходимого набора сущностей.

Далее, определим набор атрибутов для каждой сущности, после чего определим отношения между сущностями.

При этом, отношения между сущностями должны быть определены исходя из тех задач, которые будет выполнять приложение.

Соответственно, также необходимо определиться со списком задач, которые потребуются решать приложению.

4.4. Планирование Customer Journey Map (CJM):

На этапе проектирования приложения планирование Customer Journey Map (CJM) ограничено отсутствием конкретного опыта использования приложения. Поэтому необходимо создать предположительный план пользовательского пути, основываясь на выполнимых действиях и ожиданиях пользователей.

Предполагаемый план CJM для нашего приложения включает следующие этапы:

Регистрация и вход в систему:

Пользователь регистрирует свой аккаунт или входит в систему, используя свои учетные данные.

Выбор экзамена:

После входа в систему пользователь выбирает экзамен, который планирует оценить.

Просмотр экзаменационных вопросов:

Пользователь просматривает список экзаменационных вопросов, связанных с выбранным экзаменом.

Оценка ответов:

Пользователь оценивает ответы на экзаменационные вопросы, отмечая выполненные этапы и принимая решение о присвоении баллов.

Сохранение результатов:

После завершения оценки пользователь сохраняет результаты в системе.

Генерация протокола экзамена:

Пользователь генерирует протокол экзамена с результатами оценки для последующего использования или печати.

Хотя опыт использования конкретного приложения является ценным источником информации для создания CJM, на данном этапе мы оперируем предположениями и ожиданиями пользователей на основе выполнимых действий.

Предполагаемый план CJM поможет нам сформировать представление о том, как пользователи будут взаимодействовать с нашим приложением и какие этапы следует учитывать при его разработке.

4.5. Выбор паттерна проектирования

При выборе паттерна проектирования для разработки веб-приложения для автоматизации оценочного процесса в экзаменационных комиссиях был проведен анализ основных факторов, определяющих требования к архитектуре приложения.

Предпосылки к выбору паттерна проектирования:

Размер и сложность проекта:

Ожидается небольшой объем пользовательского трафика, не более 12 пользователей одновременно.

Планируется включение трех основных функциональных элементов: интерфейс экзаменатора, база данных и бэкенд.

Уровень опыта команды разработчиков:

Отсутствует опыт работы с различными паттернами проектирования.

Требования к интерфейсу:

Требуется простой интерфейс без сложных пользовательских элементов управления.

Планируется использование Single Page Application (SPA).

Требования к тестированию и поддержке:

Требуется легкое тестирование логики приложения и его компонентов.

Причины выбора паттерна проектирования:

Исходя из проведенного анализа, был выбран паттерн проектирования MVC (Model-View-Controller) в качестве наиболее подходящего для данного проекта по следующим причинам:

Простота и понятность: MVC предоставляет простую и понятную структуру, которая легко воспринимается командой разработчиков без опыта работы с паттернами проектирования.

Разделение логики приложения: MVC позволяет разделить логику приложения на три основных компонента: модель, представление и контроллер, что упрощает поддержку и дальнейшее развитие приложения.

Совместимость с SPA: Планируемое использование Single Page Application сочетается с архитектурой MVC, что обеспечивает гибкость и эффективность в разработке интерфейса приложения.

Легкость тестирования: MVC позволяет легко тестировать логику приложения и его компоненты, что соответствует требованиям к тестированию.

Альтернативы:

В качестве альтернативных паттернов проектирования рассматривались MVP (Model-View-Presenter) и MVVM (Model-View-ViewModel). Однако, учитывая отсутствие опыта работы с паттернами проектирования у команды разработчиков и требования к интерфейсу, MVC был выбран как наиболее подходящий вариант.

Раздел 4.6: Схема клиент-серверной архитектуры

Разрабатываемое приложение будет использовать клиент-серверную архитектуру для обеспечения эффективного взаимодействия между клиентской и серверной частями.

Клиентская часть:

Пользовательский интерфейс (UI): Взаимодействие с пользователем будет осуществляться через интерфейс, предоставляемый клиентской частью приложения. Это может быть веб-интерфейс, мобильное приложение или десктопное приложение.

Обработка пользовательских действий: Клиентская часть будет обрабатывать действия пользователя, такие как клики, нажатия клавиш, заполнение форм и отправка запросов на сервер.

Отображение данных: Полученные данные от сервера будут отображаться пользователю в удобном для восприятия виде.

Серверная часть:

Бизнес-логика: Здесь будет находиться вся бизнес-логика приложения, отвечающая за обработку запросов, взаимодействие с базой данных, выполнение вычислений и принятие решений.

Управление данными: Серверная часть будет отвечать за управление данными, включая сохранение, обновление, извлечение и удаление информации из базы данных.

Безопасность: Серверная часть будет обеспечивать безопасность приложения, включая аутентификацию и авторизацию пользователей, защиту от атак и утечек данных.

API (Application Programming Interface): Взаимодействие между клиентской и серверной частями будет происходить через API, который будет определять доступные конечные точки и формат запросов и ответов.

Взаимодействие между клиентом и сервером:

HTTP протокол: В качестве основного протокола передачи данных между клиентом и сервером будет использоваться HTTP, который обеспечивает удобство и широкую поддержку.

RESTful API: Для реализации API будет использоваться принцип RESTful, который предоставляет гибкую и масштабируемую архитектуру взаимодействия между клиентом и сервером.

JSON (JavaScript Object Notation): Формат обмена данными между клиентом и сервером будет основан на JSON, который является легковесным и удобным для чтения и записи как людьми, так и компьютерами.

Развертывание и масштабирование:

Облачные сервисы: Для развертывания приложения можно использовать облачные сервисы, такие как Amazon Web Services (AWS), Microsoft Azure или Google Cloud Platform, которые предоставляют гибкость и масштабируемость.

Контейнеризация: Использование контейнеризации с помощью технологий, таких как Docker, позволит легко управлять и масштабировать приложение.

Мониторинг и логирование: Важными аспектами при развертывании и масштабировании будут мониторинг производительности приложения и регистрация логов для отслеживания проблем и оптимизации работы.

Это общая схема клиент-серверной архитектуры, которая будет использоваться при разработке нашего приложения. Конкретные технологии и реализация могут меняться в зависимости от требований и целей проекта.

5. Разработка веб-приложения

5.1. Разработка клиентской части приложения на Vue.js

При разработке пользовательского интерфейса (UI) для веб-приложения применим фреймворк Vue.js. Vue.js обеспечит простоту в использовании, модульность и гибкость, что позволяет создать динамический и отзывчивый интерфейс приложению.

Компонентный подход

Применим компонентный подход к разработке пользовательского интерфейса, разбивая его на небольшие и переиспользуемые компоненты. Это позволит создать чистый и модульный код, а также упростит тестирование и поддержку приложения.

Маршрутизация

Для управления навигацией в приложении будем использовать маршрутизацию Vue Router. Vue Router позволяет определять различные маршруты и связывать их с соответствующими компонентами, обеспечивая плавное переключение между страницами без перезагрузки браузера.

Обработка событий и взаимодействие с данными

Vue.js предоставляет удобные средства для обработки событий пользовательского взаимодействия и взаимодействия с данными. Будем использовать директивы Vue для связывания данных с элементами интерфейса и реагирования на действия пользователя, такие как клики, ввод текста и другие события.

Адаптивный дизайн

При проектировании пользовательского интерфейса не требуется создавать адаптивный дизайн, так как приложение будет использоваться только на десктопных устройствах.

5.1.1 Создание интерфейса для ввода данных перед экзаменом

В веб-приложении планируется реализовать следующие функции и элементы пользовательского интерфейса:

Страница авторизации:

На странице авторизации пользователю предоставляется форма для входа в приложение.

Форма содержит следующие поля:

Форма входа с полями для ввода логина и пароля.

Кнопка "Войти", позволяющая пользователю войти в систему.

Ссылка "Забыли пароль?", для восстановления доступа к аккаунту.

Примерный вид данной страницы приведен на Рис.2

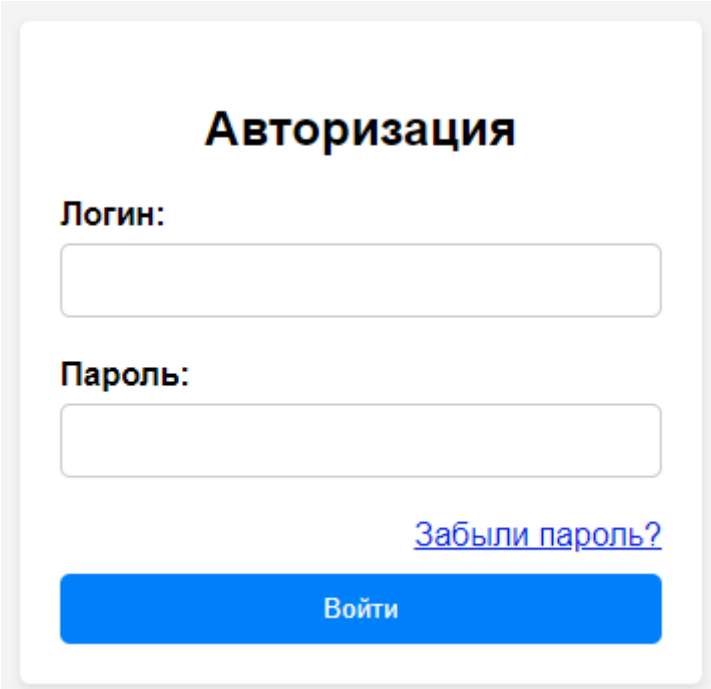
The image shows a web form for authorization. At the top, the title "Авторизация" is centered in a bold, black font. Below the title, there are two input fields. The first field is preceded by the label "Логин:" and the second by "Пароль:". Both labels are in a bold, black font. The input fields are simple white rectangles with thin gray borders. Below the password field, there is a blue hyperlink that reads "Забыли пароль?". At the bottom of the form, there is a prominent blue button with the white text "Войти". The entire form is enclosed in a light gray border.

Рис. 2 Примерный вид формы авторизации

Страница регистрации.

На странице регистрации пользователю предоставляется форма для создания нового аккаунта. Форма содержит следующие поля:

Логин: Поле для ввода уникального логина пользователя.

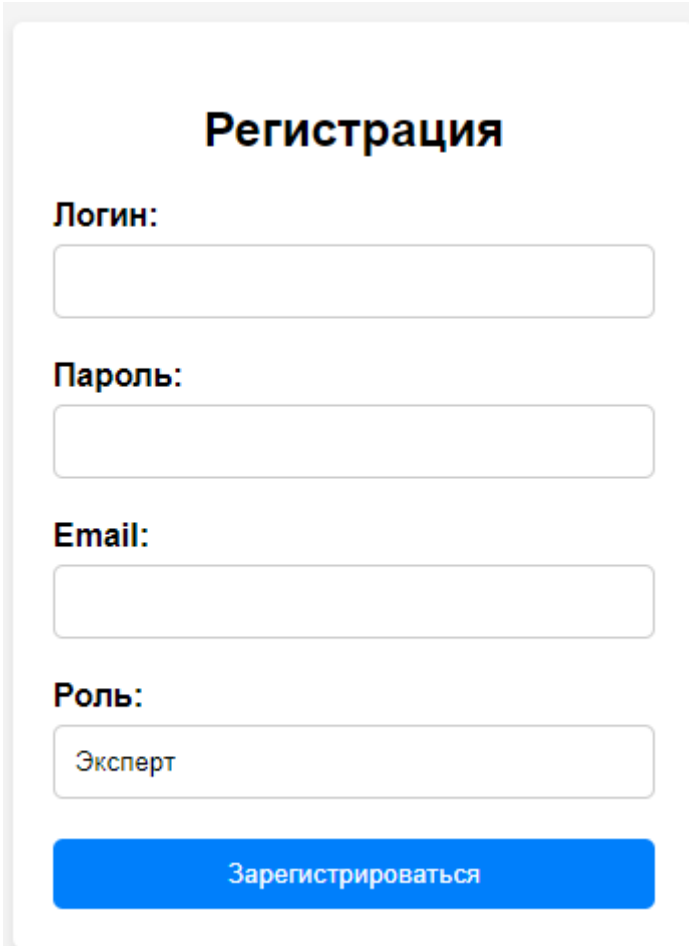
Пароль: Поле для ввода пароля. Текст вводится скрыто.

Email: Поле для ввода адреса электронной почты.

Роль: Выпадающий список для выбора роли пользователя. Доступны две опции: "Пользователь" и "Администратор".

После заполнения всех полей пользователь может нажать кнопку "Зарегистрироваться", чтобы создать новый аккаунт. При успешной регистрации пользователь будет перенаправлен на другую страницу, предоставляющую доступ к приложению.

Примерный вид данной страницы приведен на Рис.3



The image shows a registration form with the title "Регистрация" in bold black text. Below the title are four input fields, each with a label to its left: "Логин:", "Пароль:", "Email:", and "Роль:". The "Логин:", "Пароль:", and "Email:" fields are empty text boxes. The "Роль:" field is a dropdown menu with "Эксперт" selected. At the bottom of the form is a blue button with the text "Зарегистрироваться" in white.

Рис. 3 Примерный вид формы регистрации

Страница профиля пользователя:

Эта страница представляет собой профиль пользователя. В верхней части страницы расположены поля для отображения информации о пользователе, такие как имя пользователя и email. Ниже предоставлены поля для ввода нового пароля и его подтверждения.

Пользователь может ввести новый пароль в соответствующие поля и нажать кнопку "Сохранить изменения", чтобы обновить свой профиль. При этом происходит проверка на совпадение введенных паролей. Если они не совпадают, появляется сообщение об ошибке. Если пароли совпадают, данные отправляются на сервер для сохранения, и профиль пользователя успешно обновляется.

Эта страница предоставляет возможность пользователям изменять свои учетные данные, обеспечивая безопасность и удобство использования веб-платформы.

Примерный вид данной страницы приведен на Рис.4

Профиль пользователя

Имя пользователя:

Email:

Новый пароль:

Подтвердите пароль:

Сохранить изменения

Рис.4 Примерный вид профиля пользователя

Страница ввода данных перед экзаменом.

Эта страница представляет собой форму для ввода данных. В ней содержатся следующие поля:

Текущая дата: Поле, отображающее текущую дату. Данное поле не предусматривает изменения даты пользователем, а лишь отображает текущую дату.

Текущий пользователь: Поле, предназначенное для ввода имени текущего пользователя системы. Это обязательное поле.

Медицинская специальность: Поле для ввода медицинской специальности текущего пользователя. Это обязательное поле.

Тип аккредитации: Выпадающий список, позволяющий выбрать тип аккредитации. Пользователь может выбрать либо "Первичная (ПА)", либо "Первичная специализированная (ПСА)".

Паспорт станции: Выпадающий список, содержащий названия паспортов станций. Пользователь может выбрать нужный паспорт.

Номер сценария: Выпадающий список с номерами сценариев. Пользователь может выбрать нужный номер сценария.

Описание сценария: Поле, отображающее описание выбранного сценария. Данное поле доступно только для чтения.

Имя, отчество, фамилия экзаменуемого: Поле для ввода имени, отчества и фамилии экзаменуемого.

Специализация экзаменуемого: Поле для ввода специализации экзаменуемого.

Сохранить: Кнопка, позволяющая сохранить введенные данные.

Эта страница предназначена для ввода информации перед проведением экзамена и сохранения её в системе.

Примерный вид данной страницы приведен на Рис.5

Введите данные

Текущая дата:

Текущая дата

Текущий пользователь:

Введите имя пользователя

Медицинская специальность:

Введите медицинскую специальность

Тип аккредитации:

Первичная (ПА) ▾

Паспорт станции:

▾

Номер сценария:

▾

Описание сценария:

Описание сценария

**Имя, отчество, фамилия
экзаменуемого:**

Введите ФИО экзаменуемого

Специализация экзаменуемого:

Введите специализацию экзаменуемого

Сохранить

Рис. 5 Примерный вид формы ввода данных

5.1.2 Реализация возможности экзаменатора оценивать выполнение экзаменуемыми этапов и формирование протокола экзамена

Главная страница:

Эта страница представляет собой интерфейс для оценки действий экзаменуемого в рамках экзамена. В верхней части страницы расположен информационный блок, в котором отображается сводная информация, введенная на предыдущей странице, такая как текущая дата, данные об экзаменаторе и экзаменуемом, а также описание сценария экзамена.

Справа от информационного блока расположен блок суммарного количества баллов, который отображает текущее количество баллов за все выполненные действия. Справа от него размещена кнопка "Сохранить протокол", которая предназначена для сохранения результатов оценки.

Основная часть страницы представляет собой список критериев оценки в виде таблицы. Каждая строка таблицы содержит описание критерия оценки и два чек-бокса для отметки выполненных и не выполненных действий. При выборе одного из чек-боксов в строке обновляется суммарное количество баллов.

Должна быть реализована проверка на то, чтобы в каждой строке был выбран только один чек-бокс. А также проверка если оба чек-бокса не отмечены, строка подсвечивается красным, и сохранение протокола невозможно.

После сохранения протокола интерфейс переключается на системный диалог сохранения файла с протоколом в формате pdf.

Примерный вид главной страницы приведен на Рис.6

Текущая дата: 01.01.2024

Текущий экзаменатор: Иванов Иван Иванович

Медицинская специальность: Терапевт

Тип аккредитации: Первичная

Название паспорта станции: "Экстренная медицинская помощь"

Номер сценария: 10.

Описание сценария: Острое нарушение мозгового кровообращения (ОНМК)

Имя отчество Фамилия экзаменуемого: Петров Петр Петрович

Специализация экзаменуемого: Хирург

Подтвердите действие

Вы уверены, что хотите начать новый экзамен? Введенные данные будут потеряны.

ОКОтмена

3

Суммарное количество баллов

Новый экзаменСохранить протокол

Критерии оценки	Выполнил	Не выполнил
Действие №1, выполняемое экзаменуемым, в рамках текущего сценария паспорта станции	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Действие №2, выполняемое экзаменуемым, в рамках текущего сценария паспорта станции	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Действие №3, выполняемое экзаменуемым, в рамках текущего сценария паспорта станции	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Действие №4, выполняемое экзаменуемым, в рамках текущего сценария паспорта станции	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Действие №5, выполняемое экзаменуемым, в рамках текущего сценария паспорта станции	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Рис. 6 Примерный вид главной страницы

5.2 Разработка базы данных MySQL для хранения информации о экзаменуемых, экзаменаторах, специальностях, этапах и баллах

На основе ER-диаграммы, разработанной в разделе 4 создадим базу данных MySQL MedicalExamDB.sql для хранения информации о экзаменуемых, экзаменаторах, специальностях, этапах и баллах.

Сущности и их атрибуты:

Разработаем модель базы данных, для этого опишем её сущности и их атрибуты:

1. Examiner:

- id (идентификатор)
- full_name (полное имя)
- medical_specialty (медицинская специальность)

2. Examinable:

- id (идентификатор)
- full_name (полное имя)
- specialization (специализация)

3. Accreditation_type:

- id (идентификатор)
- type (тип аккредитации)

4. Specialization:

- id (идентификатор)
- specialization_name (название специализации)

5. Station_passport:

- id (идентификатор)
- passport_name (название паспорта станции)

6. Situations_List:

- id (идентификатор)
- scenario_number (номер сценария)
- station_passport_id (идентификатор паспорта станции)
- description (текстовое описание)

7. Actions_list:

- id (идентификатор)
- action (действие)
- group_characteristic (групповая характеристика)

8. Protocol:

- id (идентификатор)
- exam_date (дата экзамена)
- examiner_id (идентификатор экзаменатора) [Внешний ключ]
- examinable_id (идентификатор экзаменуемого) [Внешний ключ]
- accreditation_type_id (идентификатор типа аккредитации) [Внешний ключ]
- specialization_id (идентификатор специализации) [Внешний ключ]
- station_passport_id (идентификатор паспорта станции) [Внешний ключ]
- scenario_number_id (идентификатор номера сценария) [Внешний ключ]
- total_points (общая сумма баллов)

9. Exam_info:

- id (идентификатор)
- protocol_id (идентификатор протокола) [Внешний ключ]
- action_id (идентификатор действия) [Внешний ключ]
- result (отметка экзаменатора)

10. Action_Scenario:

- id (идентификатор)
- action_id (идентификатор действия) [Внешний ключ]
- scenario_number_id (идентификатор номера сценария) [Внешний ключ]

11. Exam:

- id (идентификатор)
- protocol_id (идентификатор протокола) [Внешний ключ]
- examinable_id (идентификатор экзаменуемого) [Внешний ключ]

12. Authentication:

- id (идентификатор)
- examiner_id (идентификатор экзаменатора) [Внешний ключ]
- username (логин)
- password_hash (хешпароля)

Описание:

Examiner (Экзаменатор): Каждый экзаменатор представляет собой медицинского специалиста, проводящего экзамены. У него есть уникальный идентификатор (id), а также персональные данные, включая полное имя (full_name) и медицинскую специальность (medical_specialty). Экзаменаторы проводят экзамены, составляют итоговые протоколы и отмечают выполнение действий экзаменуемыми.

Examinable (Экзаменуемый): Экзаменуемые - это медицинские специалисты, проходящие экзамены для получения аккредитации. У каждого экзаменуемого есть уникальный идентификатор (id) и персональные данные, такие как полное имя (full_name) и специализация (specialization). Экзаменуемые проходят через несколько станций экзамена, где выполняют определенные действия.

Accreditation_type (Тип аккредитации):

Таблица "Тип аккредитации" содержит информацию о различных типах аккредитации, которые могут быть присвоены экзаменуемым. У каждого типа аккредитации есть уникальный идентификатор (id) и название типа аккредитации (type). Этот тип может быть, например, "Primary" (первичная аккредитация) или "Primary_specialized" (первичная специализированная аккредитация). Также может быть добавлено описание типа аккредитации для более подробного описания его назначения и цели.

Specialization (Специализация): «Лечебное дело» либо «Педиатрия» (Для типа аккредитации - «Первичная аккредитация»). Специализация определяет область медицинских знаний и навыков, на которую направлена аккредитация. У каждой специализации есть уникальный

идентификатор (id) и название (name). Каждый протокол экзамена и паспорт станции связан с определенной специализацией.

Station_passport (Паспорт станции): Паспорт станции содержит информацию о базовых сценариях, которые используются для проведения экзаменов. У каждого паспорта станции есть уникальный идентификатор (id) и название базового сценария (name). Паспорт станции также связан с определенной специализацией.

Situations_List (Перечень ситуаций): Перечень ситуаций содержит описание ситуаций или кейсов, которые могут воспроизводиться на станции в рамках экзамена. У каждой ситуации есть уникальный идентификатор (id) и номер сценария (scenario_number). Каждая ситуация связана с определенным паспортом станции (station_passport_id).

Actions_List (Перечень действий): Перечень действий содержит список действий, которые должен выполнить экзаменуемый в рамках каждой ситуации на станции. У каждого действия есть уникальный идентификатор (id), название действия (action) и характеристика группы (group_characteristic). Каждое действие может включаться в разные сценарии. Действия могут быть сгруппированы по группам.

Protocol (Итоговый протокол экзамена): Итоговый протокол экзамена содержит результаты экзамена для каждого экзаменуемого. У каждого протокола есть уникальный идентификатор (id), дата экзамена (date) и связи с экзаменатором (examiner_id), экзаменуемым (examinable_id), типом аккредитации (accreditation_type_id), специализацией (specialization_id), паспортом станции (station_passport_id) и номером сценария (scenario_number_id). В протоколе также указывается общая сумма баллов (total_points).

Exam_info (Детальная информация об экзамене): Детальная информация об экзамене содержит результаты выполнения каждого действия экзаменуемым в рамках конкретного протокола. У каждой записи есть связь с протоколом (protocol_id), действием (action_id) и отметкой о выполнении (result).

Action_Scenario (Действие - Сценарий): Сущность "Действие - Сценарий" представляет собой связь между действиями и сценариями, позволяя определить, какие действия входят в каждый сценарий. Таблица нужна в связи с тем, что каждое действие может быть связано с несколькими сценариями, и каждый сценарий может включать несколько действий.

Exam (Экзамен): Таблица "Экзамен" содержит информацию о прохождении аккредитации медицинскими специалистами. Каждый экзамен связан с уникальным идентификатором (id) и экзаменуемым (examinable_id), который проходил процесс аккредитации. В рамках экзамена создается несколько протоколов экзаменов.

Authentication (Авторизация): Таблица "Авторизация" содержит информацию о пользователях системы и их учетных данных для аутентификации. Каждая запись в этой таблице представляет собой одного пользователя. У каждого пользователя есть уникальный идентификатор (id), который является первичным ключом, идентификатор экзаменатора (examiner_id), логин (username) и хеш пароля (password_hash), который обеспечивает безопасное хранение пароля в базе данных.

Описание отношений между сущностями в базе данных:

1. Examiner (Экзаменатор):

- Связь с таблицей Protocol (Итоговый протокол экзамена): Это тип связи один-ко-многим (one-to-many), так как каждый экзаменатор может провести много экзаменов, но каждый протокол экзамена принадлежит только одному экзаменатору.

2. Examinable (Экзаменуемый):

- Связь с Protocol (Итоговый протокол экзамена): Это также тип связи один-ко-многим (one-to-many), так как каждый экзаменуемый может участвовать в нескольких экзаменах, но каждый протокол экзамена относится только к одному экзаменуемому.

- Связь с Accreditation (Аккредитация): Это также тип связи один-ко-многим (one-to-many), поскольку каждый экзаменуемый может иметь несколько аккредитаций, но каждая аккредитация относится только к одному экзаменуемому.

3. Specialization (Специализация):

- Связь с Protocol (Итоговый протокол экзамена): Это связь один-к-одному (one-to-one), так как каждый протокол экзамена связан с определенной специализацией.

- Связь с Station_passport (Паспорт станции): Это тип связи многие-ко-многим (many-to-many), так как каждый паспорт станции может быть связан с несколькими специализациями, и каждая специализация может быть связана с несколькими паспортами станции.

4. Station_passport (Паспорт станции):

- Связь с Protocol (Итоговый протокол экзамена): Это также тип связи один-ко-многим (one-to-many), так как каждый протокол экзамена связан с определенным паспортом станции.

- Связь с Situations_List (Перечень ситуаций): Это тип связи один-ко-многим (one-to-many), так как каждый паспорт станции может содержать несколько ситуаций.

5. Protocol (Итоговый протокол экзамена):

- Связь с Examiner (Экзаменатор): Это тип связи многие-к-одному (many-to-one), так как каждый протокол экзамена связан только с одним экзаменатором.

- Связь с Examinable (Экзаменуемый): Это также тип связи многие-ко-одному (many-to-one), так как каждый протокол экзамена связан только с одним экзаменуемым.

6. Exam_info (Детальная информация об экзамене):

- Связь с Protocol (Итоговый протокол экзамена): Это тип связи многие-ко-одному (many-to-one), так как каждая запись с детальной информацией об экзамене связана только с одним протоколом экзамена.

7. Action_Scenario (Действие - Сценарий):

- Связь с Actions_List (Перечень действий): Это тип связи многие-ко-многим (many-to-many), так как каждое действие может быть связано с несколькими сценариями, и каждый сценарий может быть связан с несколькими действиями.

8. Exam (Экзамен):

- Связь с Examinable (Экзаменуемый): Это также тип связи многие-ко-одному (many-to-one), так как каждый экзамен связан только с одним экзаменуемым.

- Связь с Protocol (Итоговый протокол экзамена): Это также тип связи многие-ко-одному (many-to-one), так как каждый экзамен связан только с одним протоколом экзамена.

Анализ типов данных атрибутов в базе данных важен для оптимизации использования ресурсов и обеспечения эффективного хранения данных. При разработке базы данных предусмотрим:

1. Использование подходящих типов данных:

- При определении типов данных для атрибутов выберем наиболее подходящие типы в зависимости от характера данных. Например, для атрибута Date в таблице Protocol будем использовать тип данных DATE или TIMESTAMP в зависимости от необходимой точности времени.

- Для числовых значений, таких как идентификаторы (например, id), используем целочисленные типы данных (INTEGER или BIGINT), которые обеспечивают эффективное хранение и индексацию.

2. Ограничение длины строковых типов данных:

- Для строковых атрибутов, таких как first_name, last_name и т. д., определим разумные ограничения длины, чтобы избежать избыточного использования памяти. Например, VARCHAR(50) для имен может быть достаточным, если предполагается, что длина имени не превысит 50 символов.

3. Использование булевых типов данных для логических значений:

- Для атрибутов с двоичными значениями, таких как отметка экзаменатора в таблице Exam_info, будем использовать булевый тип данных (BOOLEAN или TINYINT(1)), который занимает минимальное количество памяти.

4. Использование NULL атрибутов только там, где это необходимо:

- Определим атрибуты как NULL только в случае, если они могут принимать отсутствующие значения. В противном случае, если значение обязательно для каждой записи, следует определить атрибут как NOT NULL, чтобы избежать ненужного использования памяти и ускорить запросы.

Применение этих рекомендаций поможет оптимизировать использование ресурсов базы данных и обеспечить эффективное хранение данных.

Напишем SQL-код базы данных, учитывая рекомендации выше. SQL-код для базы данных MedicalExamDB.sql приведен в Приложении 2.

Проанализируем SQL-код на соответствие требованиям к нормализации базы данных:

Все таблицы имеют первичные ключи, нет повторяющихся групп атрибутов, что соответствует требованиям первой нормальной формы.

Внешние ключи добавлены для связи между таблицами, что обеспечивает соответствие второй нормальной форме.

Все атрибуты в таблицах полностью функционально зависят от первичного ключа, не имеют транзитивных зависимостей, что соответствует требованиям третьей нормальной формы.

В таблице Action_Scenario нет зависимости между атрибутами, кроме внешних ключей, что также соответствует требованиям.

Таким образом, предложенная модель базы данных соответствует требованиям нормализации и обеспечивает структурную целостность данных.

5.3 Создание серверной части с помощью JavaSpring

Разработка веб-приложения с использованием JavaSpring предполагает создание серверной части приложения, которая обрабатывает HTTP-запросы, взаимодействует с базой данных и обеспечивает бизнес-логику приложения. В данном разделе мы подробно опишем этапы создания серверной части.

Настройка проекта SpringBoot:

В данном разделе описывается настройка проекта Maven с учетом использования необходимых зависимостей и инструментов. Для настройки проекта используем онлайн-инструмент SpringInitializer, позволяющий легко выбрать все актуальные и совместимые версии зависимостей. В файл настроек приложения включим следующие зависимости:

Spring Boot Starter Data JPA:

Данная зависимость используется для работы с JavaPersistence API (JPA) в приложении SpringBoot.

Позволяет создавать и взаимодействовать с объектами базы данных с помощью ORM (Object-RelationalMapping).

MySQLConnector/J:

Этот драйвер JDBC обеспечивает взаимодействие приложения с базой данных MySQL.

Позволяет устанавливать соединение с базой данных, отправлять запросы и получать результаты.

Spring Boot Starter Security:

Данная зависимость интегрирует Spring Security в приложение Spring Boot.

Предоставляет механизмы аутентификации, авторизации и обеспечивает безопасность приложения.

SpringBootStarterValidation:

Этот набор зависимостей включает в себя HibernateValidator для валидации данных в приложении.

Позволяет проверять данные на соответствие определенным правилам перед сохранением.

SpringBootStarterWeb:

Данная зависимость предоставляет необходимые инструменты для разработки веб-приложений с использованием Spring MVC.

Включает в себя веб-сервер и другие компоненты для обработки HTTP-запросов.

SpringBootStarterTest:

Этот набор зависимостей предоставляет зависимости для тестирования приложения SpringBoot.

Включает в себя библиотеки для написания модульных и интеграционных тестов.

SpringSecurityTest:

Зависимость предоставляет инструменты для тестирования безопасности в приложении SpringSecurity.

Lombok:

Библиотека Lombok упрощает разработку в Java, автоматически генерируя методы доступа (геттеры, сеттеры) и другие стандартные методы.

Сокращает количество шаблонного кода и улучшает читаемость кода.

HibernateValidator:

Эта зависимость предоставляет инструменты для валидации данных в Java приложениях.

Позволяет определять правила валидации для JavaBeans и проверять их на соответствие.

Настройка проекта включает в себя подключение необходимых зависимостей для разработки веб-приложения с использованием SpringBoot. Указанные зависимости обеспечивают основные функциональные возможности, такие как взаимодействие с базой данных, обеспечение безопасности, валидация данных и тестирование приложения.

5.3.1 Создание RESTful API для взаимодействия с клиентской частью

Определение структуры базы данных и создание сущностей:

Для определения структуры базы данных и создания сущностей в соответствии с требованиями приложения используется подход объектно-реляционного отображения (ORM) с помощью Java Persistence API (JPA).

Создадим сущности (Entity) для каждой таблицы с помощью аннотаций JPA, определим поля и связи между ними. В рамках данной работы были созданы следующие сущности:

Examiner (Экзаменатор):

Отвечает за хранение информации о пользователях, проводящих экзамен.

Содержит поля, такие как идентификатор, полное имя, электронная почта и другие контактные данные.

Examinable (Экзаменуемый):

Хранит информацию о пользователях, которые проходят экзамен.

Включает атрибуты, такие как идентификатор, полное имя, дата рождения и т. д.

AccreditationType (Аккредитация):

Представляет типы аккредитации.

Содержит поля, описывающие тип аккредитации, его описание и дополнительные параметры

StationPassport (Паспорт станции):

Хранит информацию о паспорте станции.

Включает в себя данные о станции, такие как название, адрес, контактная информация и другие характеристики.

SituationsList (Перечень ситуаций):

Представляет собой список ситуаций для проведения экзамена.

Включает в себя описание ситуаций и дополнительную информацию.

Specialization (Специализация):

Хранит информацию о специализациях пользователей.

Включает в себя поля, определяющие название специализации и ее описание.

ActionsList (Перечень действий):

Содержит список действий, которые могут быть выполнены в рамках экзамена.

Включает в себя описание действий и другие характеристики.

Protocol (Итоговый протокол экзамена):

Представляет собой протокол экзамена с результатами.

Включает в себя информацию о проведенном экзамене, его участниках и результатах.

ExamInfo (Информация об экзамене):

Содержит информацию о проведенных экзаменах.

Включает в себя данные о дате, месте, типе экзамена и другие характеристики.

ActionScenario (Действие - Сценарий):

Описывает связь между действиями и сценариями экзамена.

Включает в себя данные о том, какие действия включены в каждый сценарий.

Exam (Экзамен):

Хранит информацию о проведенных экзаменах.

Включает в себя данные о дате, участниках, результатах и других параметрах экзамена.

Authentication (Аутентификация):

Содержит информацию о пользователях и их учетных записях для аутентификации.

Включает в себя данные о логинах, паролях и других учетных данных.

Для корректного маппинга объектов Java на таблицы в базе данных установим соответствующие аннотации, такие как: @Entity, @Id, @GeneratedValue, @Column.

Для обеспечения корректного хранения и обработки данных, кроме внешних ключей, в Java-коде бэкенда приложения, описывающем сущности (Entity), были добавлены ограничение уникальности, проверки целостности и ссылочные ограничения.

1. Ограничение уникальности:

- Для обеспечения уникальности в таблице «Examiner» по атрибуту «full_name» добавлено ограничение (проверка) уникальности, при вводе значений пользователями. То же самое ограничение применено и к таблице «Examinable» для атрибута «full_name».

Проверка уникальности реализована при помощи применения аннотации @Column с параметром unique = true.

Добавив аннотацию @Column(unique = true) к полю fullName, мы указываем, что этот столбец в базе данных должен содержать уникальные значения.

Обеспечение уникальности гарантирует, что каждый экзаменатор и каждый экзаменуемый будут иметь уникальное полное имя в системе.

2. Проверки целостности:

- Для атрибута Date в таблице Protocol при вводе значения было предусмотрено условие, чтобы дата экзамена не могла быть позже текущей даты. Это реализовано при помощи аннотации @PastOrPresent из библиотеки HibernateValidator.

- В таблице Exam_info для атрибута Result при вводе значения было предусмотрено условие, чтобы оно принимало значения только true или false. Для этого мы используем аннотацию @Column с атрибутом column Definition=BOOLEAN чтобы явно указать тип данных столбца в базе данных.

3. Ссылочные ограничения:

- Для всех внешних ключей, которые связывают таблицы между собой, были добавлены ссылочные ограничения, чтобы гарантировать существование соответствующих записей в связанных таблицах.

Например, для внешних ключей, указывающих на таблицы Examiner, Examinable, Accreditation_type, Specialization, Station_passport и Scenario_numbers, добавлены ссылочные ограничения, чтобы они указывали на существующие записи в соответствующих таблицах.

Добавление ссылочных ограничений (foreign key constraints) к внешним ключам в таблицах базы данных было сделано с использованием аннотаций Hibernate, таких как @ManyToOne, @JoinColumn, @OneToOne, @ManyToMany.

Эти ограничения помогут обеспечить целостность данных и предотвратить появление некорректных или неполных записей в базе данных.

Код базы данных приведен в Приложении 2.

Код сущностей приведен в Приложении 3.

Создание репозитория:

Напишем репозитории для каждой сущности, которые будут расширять интерфейс JpaRepository.

После описания сущностей приложения, необходимо создать репозитории, в которых будут описываться операции, производимые приложением с данными. Обычно это стандартные операции CRUD (Create, Read, Update, Delete), а также другие специфичные операции, такие как поиск по различным критериям или связанным данным.

Характеристики сущности (Entity) определяют, какие операции будут доступны в репозитории. При разработке нужно учитывать предполагаемые сценарии использования, чтобы определить, какие дополнительные операции могут понадобиться.

Функции, выполняемые репозиториями в отношении каждой сущности:

ExaminerRepository: Используется для сохранения, получения, обновления и удаления информации о экзаменаторах.

Методы CRUD (Create, Read, Update, Delete) позволяют выполнять операции с данными экзаменаторов. Методы для получения списка экзаменов, проведенных конкретным экзаменатором. Методы для получения списка всех экзаменаторов.

ExaminableRepository: Используется для сохранения, получения, обновления и удаления информации об экзаменуемых. Содержит методы для управления сущностью "Examinable", такие как создание, чтение, обновление и удаление, а также методы для получения информации о том, в каких экзаменах участвовал конкретный экзаменуемый.

Методы для получения информации о том, какие протоколы содержатся в экзаменах, которые сдал экзаменуемый, какие паспорта станций, сценарии, действия содержатся в протоколах, в которых участвовал экзаменуемый. Получение списка экзаменуемых для конкретного экзамена.

AccreditationTypeRepository: Используется для сохранения, получения, обновления и удаления информации о типах аккредитации. Включает операции CRUD для управления типами аккредитации.

StationPassportRepository: Включает методы для управления информацией о паспортах станций, такие как создание, чтение, обновление и удаление.

Используется для сохранения, получения, обновления и удаления информации о паспортах станций.

SituationsListRepository: Включает операции CRUD для управления данными о перечне ситуаций. Используется для сохранения, получения, обновления и удаления информации о ситуациях.

SpecializationRepository: Включает методы для управления информацией о специализациях, такие как создание, чтение, обновление и удаление. Используется для сохранения, получения, обновления и удаления информации о специализациях.

ActionsListRepository: Включает операции CRUD для управления данными о перечне действий. Используется для сохранения, получения, обновления и удаления информации о действиях.

ProtocolRepository: Включает методы для управления информацией о протоколах, такие как создание, чтение, обновление и удаление. Используется для сохранения, получения, обновления и удаления информации о протоколах экзамена..

Методы для получения информации о том, какой паспорт станции у заданного протокола, какой сценарий у протокола для данного паспорта станции. Методы для получения списка действий для заданного сценария и паспорта станции конкретного протокола. Методы для получения результата выполнения по каждому действию для заданного сценария, заданного паспорта станции, заданного протокола.

ExamInfoRepository: Включает операции CRUD для управления данными об информации об экзамене. Используется для сохранения, получения, обновления и удаления информации о экзаменах.

ActionScenarioRepository: Включает операции CRUD для управления данными о сценариях действий. Используется для сохранения, получения, обновления и удаления информации о сценариях действий.

ExamRepository: Содержит методы для управления информацией о экзаменах, такие как создание, чтение, обновление и удаление. Используется для сохранения, получения, обновления и удаления информации об экзаменах.

Методы для получения информации о том, какие протоколы содержит данный экзамен, какие результаты содержатся в протоколах, входящих в данный экзамен.

AuthenticationRepository: Включает операции CRUD для управления данными аутентификации. Используется для сохранения, получения, обновления и удаления информации об аутентификации.

Код созданных репозиториях приведен в Приложении 4.

Создание контроллеров:

Контроллеры: Контроллеры обрабатывают HTTP-запросы и взаимодействуют с клиентами приложения. Они принимают запросы от клиентов, вызывают соответствующие методы в сервисах для выполнения бизнес-логики и возвращают ответы обратно клиентам. Контроллеры также могут обрабатывать валидацию запросов, преобразование данных и управление исключениями.

Напишем контроллеры для обработки HTTP-запросов. Создадим классы контроллеров и определим в них методы для обработки различных типов запросов.

Зададим пути (endpoints) для каждого метода контроллера, определим типы запросов (GET, POST, PUT, DELETE) и параметры, передаваемые через URL или тело запроса.

ExaminerController:

GET /examiners: Получить список всех экзаменаторов.

GET /examiners/{id}: Получить информацию о конкретном экзаменаторе по его идентификатору.

POST /examiners: Создать нового экзаменатора.

PUT /examiners/{id}: Обновить информацию о конкретном экзаменаторе.

DELETE /examiners/{id}: Удалить экзаменатора.

GET /examiners/{id}/exams: Получить список экзаменов, проведенных конкретным экзаменатором.

ExaminableController:

GET /examinables: Получить список всех экзаменуемых.

GET /examinables/{id}: Получить информацию о конкретном экзаменуемом по его идентификатору.

POST /examinables: Создать нового экзаменуемого.

PUT /examinables/{id}: Обновить информацию о конкретном экзаменуемом.

DELETE /examinables/{id}: Удалить экзаменуемого.

GET /examinables/{id}/exams: Получить информацию о том, в каких экзаменах участвовал конкретный экзаменуемый.

AccreditationTypeController:

GET /accreditations: Получить список всех типов аккредитации.

GET /accreditations/{id}: Получить информацию о конкретном типе аккредитации по его идентификатору.

POST /accreditations: Создать новый тип аккредитации.

PUT /accreditations/{id}: Обновить информацию о конкретном типе аккредитации.

DELETE /accreditations/{id}: Удалить тип аккредитации.

StationPassportController:

GET /stations: Получить список всех паспортов станций.

GET /stations/{id}: Получить информацию о конкретном паспорте станции по его идентификатору.

POST /stations: Создать новый паспорт станции.

PUT /stations/{id}: Обновить информацию о конкретном паспорте станции.

DELETE /stations/{id}: Удалить паспорт станции.

SituationsListController:

GET /situations: Получить список всех ситуаций.

GET /situations/{id}: Получить информацию о конкретной ситуации по ее идентификатору.

POST /situations: Создать новую ситуацию.

PUT /situations/{id}: Обновить информацию о конкретной ситуации.

DELETE /situations/{id}: Удалить ситуацию.

ActionsListController:

GET /actions: Получить список всех действий.

GET /actions/{id}: Получить информацию о конкретном действии по его идентификатору.

POST /actions: Создать новое действие.

PUT /actions/{id}: Обновить информацию о конкретном действии.

DELETE /actions/{id}: Удалить действие.

ProtocolController:

GET /protocols: Получить список всех протоколов экзамена.

GET /protocols/{id}: Получить информацию о конкретном протоколе экзамена по его идентификатору.

POST /protocols: Создать новый протокол экзамена.

PUT /protocols/{id}: Обновить информацию о конкретном протоколе экзамена.

DELETE /protocols/{id}: Удалить протокол экзамена.

ExamInfoController:

GET /exams: Получить список всех экзаменов.

GET /exams/{id}: Получить информацию о конкретном экзамене по его идентификатору.

POST /exams: Создать новый экзамен.

PUT /exams/{id}: Обновить информацию о конкретном экзамене.

DELETE /exams/{id}: Удалить экзамен.

SpecializationController:

GET /specializations: Получение списка всех специализаций.

GET /specializations/{id}: Получение информации о конкретной специализации по её идентификатору.

POST /specializations: Создание новой специализации.

PUT /specializations/{id}: Обновление информации о специализации по её идентификатору.

DELETE /specializations/{id}: Удаление специализации по её идентификатору.

Аннотации Swagger в SpecializationController:

ActionScenarioController:

GET /actions-scenarios: Получение списка всех связей между действиями и сценариями.

GET /actions-scenarios/{id}: Получение информации о конкретной связи по её идентификатору.

POST /actions-scenarios: Создание новой связи между действием и сценарием.

PUT /actions-scenarios/{id}: Обновление информации о связи по её идентификатору.

DELETE /actions-scenarios/{id}: Удаление связи между действием и сценарием по её идентификатору.

ExamController:

GET /exams: Получение списка всех экзаменов.

GET /exams/{id}: Получение информации о конкретном экзамене по его идентификатору.

POST /exams: Создание нового экзамена.

PUT /exams/{id}: Обновление информации об экзамене по его идентификатору.

DELETE /exams/{id}: Удаление экзамена по его идентификатору.

AuthenticationController:

POST /auth/login: Аутентификация пользователя на основе предоставленных учетных данных.

POST /auth/logout: Завершение сеанса аутентификации и выход пользователя из системы.

В каждом методе контроллера опишем логику обработки запроса и формирования ответа. Это может включать в себя вызов соответствующих сервисов, обработку данных и формирование HTTP-ответа.

Обработка входных данных:

Проверим и валидируем входные данные, полученные от клиента, чтобы гарантировать их корректность и безопасность. Используем аннотации проверки ввода данных и обработку ошибок в контроллерах.

Формирование HTTP-ответов:

Подготовим структуру и формат HTTP-ответов для каждого метода API. Ответы должны быть читаемыми и содержательными, содержать необходимую информацию о результатах операции или ошибках, если таковые возникли.

Документирование API:

Создадим документацию для нашего API с помощью инструментов, таких как Swagger или Springfox. Описанием каждой конечной точки, параметров запроса и примеров использования.

Код контроллеров приведен в Приложении 6.

5.3.2 Реализация логики подсчета баллов и формирования протокола экзамена

Этот этап разработки включает в себя реализацию логики, которая отвечает за подсчет баллов, оценку экзамена и формирование протокола с результатами. Ниже приведены основные шаги этого процесса:

Определение требований к подсчету баллов:

Логика подсчета баллов заключается в суммировании баллов, за каждый правильный ответ. Причем каждый экзамен может включать в себя несколько протоколов. То есть, сначала суммируются баллы в рамках испытания, проводимого по каждому паспорту станции.

После чего результаты всех испытаний суммируются и выводится средний процент всех испытаний, а также процент правильных ответов по всем пройденным промежуточным испытаниям.

Вес каждого ответа принимается равным единице.

Разработка алгоритма подсчета баллов:

В рамках промежуточного испытания по какому-либо паспорту станции и конкретному сценарию, логику суммирования баллов логично осуществлять «на лету» на стороне фронтэнда, то есть клиентской части.

Экзаменатор, выставя и снимая чек-боксы может сразу видеть результаты своих выставленных оценок, но они не являются окончательными до тех пор, пока экзаменатор не нажмет кнопку сохранить.

После нажатия кнопки сохранения все результаты сохраняются в базу данных.

После прохождения экзаменуемым всех станций, экзаменатор нажимает кнопку формирования итогового протокола. И вот эта логика формирования итогового протокола осуществляется на стороне бэкенда, так как для этого потребуется получить из базы данных из разных таблиц данные по заданным критериям, обработать их и выдать результат экзаменатору в виде итогового протокола.

Сервисы: Сервисы являются промежуточным уровнем между контроллерами и репозиториями. Они содержат бизнес-логику приложения и обрабатывают сложные операции, связанные с сущностями.

Например, сервисы могут содержать логику для создания новых сущностей, обновления существующих, выполнения сложных запросов к базе данных и т. д.

Разработка сервисов:

Напишем сервисы, которые будут обеспечивать бизнес-логику приложения. Создадим классы сервисов и определим в них методы для выполнения различных операций с данными.

В сервисах также реализуем логику обработки данных, включая валидацию, проверку прав доступа и другие бизнес-процессы.

ExaminerService (Сервис экзаменаторов):

Описание: Этот сервис отвечает за управление экзаменаторами, включая создание, получение, обновление и удаление информации об экзаменаторах. Также он предоставляет функции для получения списка экзаменов, проведенных конкретным экзаменатором, и списка всех экзаменаторов.

ExaminableService (Сервис экзаменуемых):

Описание: Данный сервис управляет экзаменуемыми, включая создание, получение, обновление и удаление информации об экзаменуемых. Он также предоставляет функции для получения списка экзаменуемых для конкретного экзамена и информации о проведенных экзаменуемым экзаменах.

AccreditationTypeService (Сервис типов аккредитации):

Описание: Этот сервис обеспечивает управление типами аккредитации, включая создание, получение, обновление и удаление информации о типах аккредитации.

SpecializationService (Сервис специализаций):

Описание: Специализационный сервис отвечает за управление специализациями, включая создание, получение, обновление и удаление информации о специализациях.

StationPassportService (Сервис паспортов станций):

Описание: Этот сервис предоставляет функциональность для работы с паспортами станций, включая создание, получение, обновление и удаление информации о паспортах станций.

SituationsListService (Сервис перечня ситуаций):

Описание: Сервис перечня ситуаций обеспечивает управление перечнями ситуаций, включая создание, получение, обновление и удаление информации о перечнях ситуаций.

ActionsListService (Сервис перечня действий):

Описание: Этот сервис отвечает за управление перечнями действий, включая создание, получение, обновление и удаление информации о перечнях действий.

ProtocolService (Сервис протокола экзамена):

Описание: Сервис протокола экзамена обеспечивает управление протоколами экзамена, включая создание, получение, обновление и удаление информации о протоколах экзамена.

ExamInfoService (Сервис информации об экзамене):

Описание: Данный сервис управляет информацией об экзамене, включая создание, получение, обновление и удаление информации об экзамене.

ActionScenarioService (Сервис действия - сценария):

Описание: Сервис действия - сценария отвечает за управление связями между действиями и сценариями, включая создание, получение, обновление и удаление информации о связях между действиями и сценариями.

ExamService (Сервис экзамена):

Описание: Этот сервис предоставляет функциональность для работы с экзаменами, включая создание, получение, обновление и удаление информации об экзаменах.

AuthenticationService (Сервис аутентификации):

Описание: Сервис аутентификации обеспечивает управление аутентификацией пользователей, включая создание, получение, обновление и удаление информации об аутентификации.

Код сервисов приведен в Приложении 5.

5.4. Интеграция клиентской и серверной частей приложения

Интеграция клиентской и серверной частей приложения – это процесс объединения фронтенда и бэкенда таким образом, чтобы они взаимодействовали между собой для обеспечения функциональности всего приложения. Этот процесс включает в себя несколько ключевых шагов:

Определение API: На этом этапе определяются эндпоинты (URL-адреса) и методы (GET, POST, PUT, DELETE), которые будут использоваться для обмена данными между клиентом и сервером. API должно быть хорошо спроектировано, учитывая все необходимые операции и данные, которые требуются клиентскому приложению.

Разработка бэкенда: Разработка серверной части приложения, которая будет предоставлять API для взаимодействия с клиентом. Здесь создаются контроллеры, сервисы и репозитории для обработки запросов, выполнения бизнес-логики и работы с базой данных.

Разработка фронтенда: Создание пользовательского интерфейса (UI) и клиентской логики, которая будет взаимодействовать с бэкендом через определенные API. В этом процессе используются HTML, CSS и JavaScript (или фреймворки, такие как React, Angular или Vue.js) для создания пользовательского интерфейса и реализации логики взаимодействия с сервером.

Тестирование: После разработки как клиентской, так и серверной частей приложения проводится тестирование для проверки их корректной работы. Это включает модульное тестирование для каждой из частей, а также интеграционное тестирование для проверки взаимодействия между ними.

Развертывание и настройка: Готовое приложение разворачивается на сервере, а клиентская часть загружается на клиентские устройства. Настройка серверной инфраструктуры также может включать в себя настройку базы данных, веб-сервера, облачных сервисов и прочего.

Интеграция: Непосредственно процесс объединения клиентской и серверной частей приложения. Здесь настраивается взаимодействие между ними, например, клиентский код отправляет запросы на серверные эндпоинты, а сервер обрабатывает эти запросы и возвращает соответствующие ответы.

Отладка и оптимизация: После интеграции производится отладка и оптимизация приложения для устранения возможных проблем и повышения его производительности. Это может включать в себя исправление ошибок, улучшение интерфейса пользователя, оптимизацию работы сети и прочее.

Поддержка и обновление: После запуска приложения на продакшн поддержка и обновление приложения, включая исправление ошибок, добавление новых функций и улучшение производительности, являются важной частью разработки.

6. Тестирование и отладка

6.1. Проведение модульных и интеграционных тестов

Для написания модульных тестов для контроллеров следует использовать специализированные фреймворки для тестирования веб-приложений, такие как JUnit.

Общий подход к написанию модульных тестов для контроллеров.

Подготовка тестового окружения: Создание фиктивных или временных баз данных, подготовка тестовых данных, создание объектов-моков для сервисов, которые используются в контроллерах.

Написание тестовых методов: Написание отдельных тестовых методов для каждого метода контроллера. Каждый тест должен проверять определенный сценарий использования или функциональность.

Вызов контроллеров с имитированными запросами: В каждом тесте следует имитировать HTTP-запросы к контроллерам с помощью специальных библиотек для тестирования веб-приложений. Запросы должны содержать необходимые параметры и данные.

Проверка результата: После вызова метода контроллера проверяется возвращаемый им результат. Это может быть HTTP-статус ответа, тело ответа или другие аспекты ответа, в зависимости от ожидаемого поведения.

Оценка корректности ответа: Сравнение полученного ответа с ожидаемым результатом с учетом конкретных условий и ожидаемых результатов.

Очистка ресурсов и завершение теста: По окончании каждого теста необходимо освободить используемые ресурсы и выполнить другие завершающие действия.

Пример кода модульного теста на Java с использованием фреймворка JUnit:

```
import static org.junit.Assert.assertEquals;
import static org.mockito.Mockito.*;
import org.junit.Before;
import org.junit.Test;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
```



```

public class SpecializationControllerTest {

    private SpecializationController specializationController;
    private SpecializationService specializationServiceMock;

    @Before
    public void setUp() {
        specializationServiceMock = mock(SpecializationService.class);
        specializationController = new
SpecializationController(specializationServiceMock);
    }

    @Test
    public void testGetSpecializations() {
        // Arrange
        List<Specialization> expectedSpecializations = Arrays.asList(
            new Specialization("1", "Specialization 1"),
            new Specialization("2", "Specialization 2")
        );

        when(specializationServiceMock.getAllSpecializations()).thenReturn(expectedSp
ecializations);

        // Act
        ResponseEntity<List<Specialization>> response =
specializationController.getSpecializations();

        // Assert
        assertEquals(HttpStatus.OK, response.getStatusCode());
        assertEquals(expectedSpecializations, response.getBody());
    }
    // Другие тесты для остальных методов контроллера
}

```

Этот пример демонстрирует написание модульного теста для метода `getSpecializations()` контроллера `SpecializationController`. Таким же образом можно написать тесты для остальных методов контроллеров.

6.2. Отладка ошибок и исправление дефектов

После завершения тестирования неизбежно возникают ситуации, когда обнаруживаются ошибки или дефекты в приложении. Этот этап включает в себя процесс отладки и исправления обнаруженных проблем. Вот основные шаги этого процесса:

Идентификация проблемы: В случае возникновения ошибки необходимо в первую очередь выяснить ее причину. Это может включать в себя анализ сообщений об ошибках, просмотр логов, воспроизведение проблемы и т. д.

Воспроизведение ошибки: Для того чтобы эффективно исправить ошибку, важно воспроизвести ее на тестовом или разработческом окружении. Это позволит лучше понять контекст и условия, при которых возникает проблема.

Использование инструментов отладки: Для эффективной отладки ошибок можно использовать различные инструменты, такие как отладчики, профилировщики, инструменты мониторинга производительности и т. д. Они помогают выявить и проанализировать причины возникновения проблем.

Исправление ошибок: После того как причина ошибки была идентифицирована, можно приступить к ее исправлению. Это может включать в себя изменение кода, обновление зависимостей, конфигурационных файлов и т. д.

Тестирование исправлений: После внесения изменений необходимо протестировать приложение, чтобы убедиться, что исправления успешно устранили проблему. Это может включать в себя повторное тестирование ранее обнаруженных проблемных сценариев, а также создание новых тестов для проверки исправленного функционала.

Документация изменений: Важно внести соответствующие изменения в документацию проекта, чтобы описать обнаруженные ошибки, принятые меры по их устранению и версии приложения, в которых они были исправлены.

Выпуск обновления: После успешного исправления ошибок и прохождения тестирования необходимо подготовить и выпустить обновленную версию приложения. Это обеспечит пользователям доступ к исправленной функциональности и повысит общую стабильность и надежность приложения.

Мониторинг и обратная связь: После выпуска обновления важно продолжать мониторинг работы приложения и собирать обратную связь от пользователей. Это позволит быстро реагировать на новые проблемы и обеспечить высокий уровень удовлетворенности пользователей.

7. Разработка документации

7.1 Описание функциональности приложения:

Приложение представляет собой систему управления процессом проведения экзаменов и аттестаций. Оно предназначено для организации и автоматизации всех этапов проведения экзаменов, включая создание экзаменаторов и экзаменуемых, определение типов аккредитации, управление специализациями, составление протоколов экзаменов и многое другое.

Функциональные возможности приложения включают в себя:

1. Управление экзаменаторами: создание, получение, обновление и удаление информации о экзаменаторах.
2. Управление экзаменуемыми: создание, получение, обновление и удаление информации об экзаменуемых.
3. Управление типами аккредитации: создание, получение, обновление и удаление информации о типах аккредитации.
4. Управление специализациями: создание, получение, обновление и удаление информации о специализациях.
5. Управление паспортами станций: создание, получение, обновление и удаление информации о паспортах станций.
6. Управление перечнями ситуаций: создание, получение, обновление и удаление информации о перечнях ситуаций.
7. Управление перечнями действий: создание, получение, обновление и удаление информации о перечнях действий.
8. Управление протоколами экзамена: создание, получение, обновление и удаление информации о протоколах экзамена.
9. Управление информацией об экзамене: создание, получение, обновление и удаление информации об экзамене.
10. Управление связями между действиями и сценариями: создание, получение, обновление и удаление информации о связях между действиями и сценариями.
11. Управление экзаменами: создание, получение, обновление и удаление информации об экзаменах.
12. Управление аутентификацией: создание, получение, обновление и удаление информации об аутентификации.

7.2. Инструкция по установке и запуску приложения:

Для развертывания приложения на Vue.js на сервере под управлением Windows Server 2012 выполните следующие шаги:

Сборка приложения для продакшена: Перейдите в директорию вашего Vue.js приложения и выполните команду `npm run build`. Это создаст готовые для развертывания файлы в папке `dist`.

Создание веб-сайта в IIS: Откройте "Internet Information Services (IIS) Manager" через "Server Manager". Создайте новый сайт, укажите имя, путь к папке с собранными файлами вашего Vue.js приложения и порт, по которому сайт будет доступен.

Настройка параметров веб-сайта: При необходимости настройте параметры веб-сайта, такие как привязки и SSL-сертификаты.

Настройка разрешений и доступа: Убедитесь, что разрешения на файлы и папки вашего сайта настроены корректно для пользователя, под которым работает веб-сервер IIS.

Проверка работоспособности: После развертывания приложения проверьте его работоспособность, перейдя по URL-адресу вашего веб-сайта в браузере.

Для развертывания приложения на Java Spring с использованием Spring Boot на сервере под управлением Windows Server 2012 следует выполнить следующие шаги:

Скомпилируйте приложение в исполняемый JAR файл: Используйте средства сборки проекта, такие как Maven или Gradle, для компиляции приложения. Укажите, что вы используете Spring Boot и определите основной класс вашего приложения.

Перенесите JAR файл на сервер: Скопируйте полученный JAR файл вашего приложения на сервер Windows Server 2012 в удобное для вас место.

Запустите приложение: Откройте командную строку на сервере и выполните команду `java -jar your-application.jar`, где `your-application.jar` - имя вашего JAR файла. Это запустит встроенный сервер Spring Boot и автоматически развернет ваше приложение на указанном порту.

Настройте правила брандмауэра: Если приложение должно быть доступно извне, убедитесь, что правила брандмауэра на сервере разрешают входящие соединения на используемый вами порт.

Проверьте работоспособность: После запуска приложения проверьте его работоспособность, перейдя по URL-адресу вашего сервера и используемому порту.

7.3. Техническая документация:

Техническая документация представляет собой подробное описание архитектуры, структуры и функциональности приложения, предназначенное для разработчиков, администраторов и других заинтересованных лиц. Включает в себя следующие разделы:

Архитектура приложения: Описание общей архитектуры приложения, используемых технологий и инструментов разработки.

Структура проекта: Обзор структуры файлов и каталогов проекта, включая иерархию компонентов и модулей.

Описание функциональности: Подробное описание основной функциональности приложения, включая описание всех основных функций, их использование и взаимодействие с пользователем.

API документация: Описание всех эндпоинтов API, используемых в приложении, включая их описание, параметры запросов и примеры ответов.

Инструкции по установке и запуску: Подробные инструкции по установке и запуску приложения на различных платформах и серверах.

Инструкции по развертыванию: Подробные инструкции по развертыванию приложения на продакшн-сервере, включая настройку сервера, базы данных и других внешних зависимостей.

Руководство пользователя: Описание основных возможностей приложения и инструкции по их использованию для конечного пользователя.

Инструкции по обновлению и поддержке: Рекомендации по обновлению приложения до новых версий, а также инструкции по поддержке и решению проблем.

Техническая документация помогает разработчикам лучше понять приложение, его структуру и функциональность, что облегчает поддержку, развитие и сопровождение приложения в будущем.

7.4. Форматирование и оформление документации:

Для создания читаемой, понятной и профессионально выглядящей документации рекомендуется следующее форматирование и оформление:

Использование ясной структуры: Разделить документацию на логические разделы и подразделы, такие как "Введение", "Установка", "Настройка", "Использование" и т.д.

Использование заголовков и подзаголовков: Использование заголовков различных уровней для выделения основных разделов и подразделов.

Описание шагов и инструкций: Предоставление пошаговых инструкций для установки, настройки и использования вашего приложения.

Использование списков: Использование маркированных и нумерованных списков для перечисления шагов, параметров, настроек и другой важной информации.

Примеры кода и сниппеты: Вставка примеров кода, конфигураций и запросов API для наглядного представления функциональности и использования приложения.

Использование форматирования текста: Выделение важных терминов, кода, URL-адресов и других ключевых элементов с помощью форматирования текста, такого как жирный, курсив или моноширинный шрифт.

Использование графиков и изображений: Для наглядной демонстрации концепций, архитектуры или диаграмм использования необходимо вставлять графики, схемы и скриншоты.

Создание ссылок и перекрестных ссылок: Для облегчения навигации нужно добавлять ссылки на другие разделы документации, внешние ресурсы и дополнительную информацию.

Обеспечение доступности и актуальности: Документация должна быть доступна для всех членов команды разработки и должна обновляться при внесении изменений в приложение.

Проверка правописания и грамматики: Перед публикацией документации нужно проверить документацию на наличие ошибок и опечаток.

Соблюдение этих принципов форматирования и оформления поможет создать информативную и удобную в использовании документацию для приложения.

8. Основные результаты проекта

8.1. Основные результаты проекта:

В результате выполнения проекта были достигнуты следующие основные результаты:

Разработано полноценное веб-приложение на базе Java Spring и Vue.js для управления экзаменационным процессом, включая создание экзаменаторов, экзаменуемых, типов аккредитации и паспортов станций.

Приложение предоставляет удобный интерфейс для взаимодействия с базой данных, а также обеспечивает безопасность и аутентификацию пользователей.

Разработано RESTful API для взаимодействия с клиентской частью приложения, что обеспечивает гибкость и расширяемость системы.

Проведено тестирование и оптимизация API и клиентской части приложения для обеспечения высокой производительности и устойчивости.

Создана документация для API и инструкции по установке и запуску приложения, что облегчает его использование и поддержку.

8.2. Выводы и рекомендации по дальнейшему развитию приложения:

После разработки и развертывания приложения, можно сделать следующие выводы и рекомендации для его дальнейшего развития:

Обратная связь пользователей: Следует активно собирать обратную связь от пользователей приложения для выявления и устранения возможных проблем, а также для понимания их потребностей и предпочтений.

Дальнейшее улучшение интерфейса: Важно продолжать работу над улучшением пользовательского интерфейса, сделав его более интуитивно понятным и удобным для использования.

Расширение функциональности: Исходя из обратной связи пользователей и анализа рынка, можно рассмотреть возможность добавления новых функциональных возможностей, которые могут улучшить опыт пользователей или расширить спектр предлагаемых услуг.

Оптимизация производительности: При необходимости следует провести оптимизацию производительности приложения для обеспечения его эффективной работы даже при высоких нагрузках.

Обновление безопасности: Необходимо регулярно обновлять компоненты и зависимости приложения для обеспечения его защиты от возможных уязвимостей и атак.

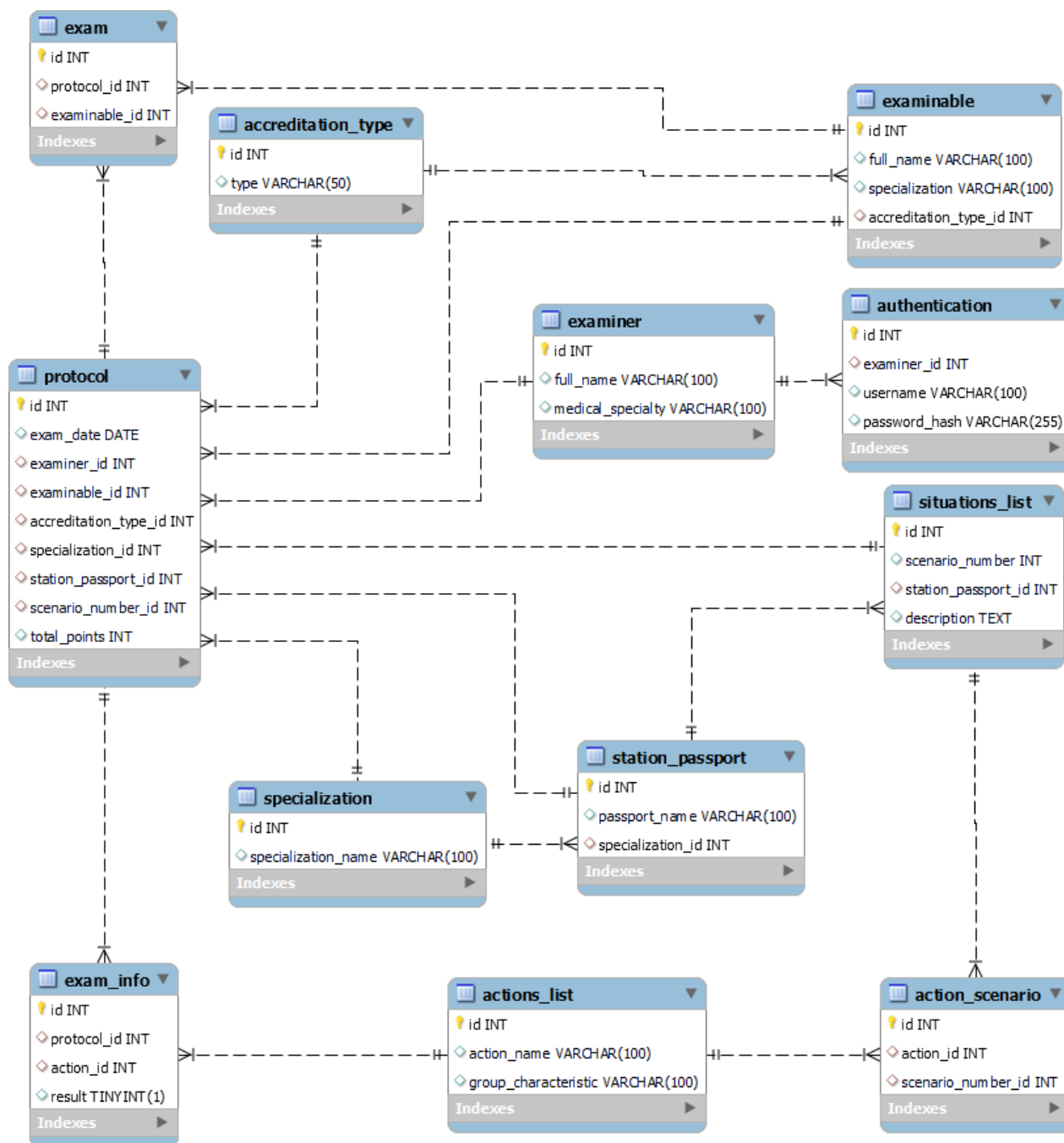
Интеграция с другими сервисами: Необходимо рассмотреть возможность интеграции приложения с другими сервисами или платформами для расширения его функциональности и увеличения числа потенциальных пользователей.

9. Список используемой литературы

1. Фаулер Мартин. "Рефакторинг: Улучшение существующего кода". Пер. с англ. - СПб.: Питер, 2019. - 464 с.
2. Стебловская Е.А. "Разработка Web-приложений на Java с использованием Spring Framework". - СПб.: БХВ-Петербург, 2019. - 256 с.
3. Уоллс Крейг. "Spring в действии". - СПб.: Питер, 2017. - 496 с.
4. Паланик Чак. "Vue.js в действии". - М.: ДМК Пресс, 2019. - 448 с.
5. Тарасов Владимир. "Vue.js 2: Практическое руководство по разработке современных веб-приложений". - СПб.: Питер, 2020. - 384 с.
6. Харроп Тим. "Vue.js. Полное руководство". - СПб.: БХВ-Петербург, 2019. - 328 с.
7. Исаченко А.Г., Горбунова И.В. "Разработка веб-приложений на JavaScript и HTML5". - М.: ДМК Пресс, 2019. - 544 с.
8. Лоуренс Маркус, Роббинс Райан. "Программирование на Java для начинающих". - М.: ДМК Пресс, 2018. - 480 с.
9. Гусейнов Рустам. "Java Spring Framework для профессионалов". - М.: ДМК Пресс, 2020. - 768 с.
10. Жуковский Д.В. "Vue.js. Разработка компонентов на JavaScript". - М.: ДМК Пресс, 2019. - 256 с.
11. Vue.js Documentation. [Онлайн]. Доступно: <https://vuejs.org/v2/guide/>. - Официальная документация Vue.js, содержащая подробную информацию о фреймворке Vue.js, его возможностях и способах использования.
12. "Spring Framework Documentation." [Онлайн]. Доступно: <https://spring.io/projects/spring-framework>. - Официальная документация Spring Framework, содержащая информацию о различных модулях и возможностях фреймворка Spring.
13. Вагин, Сергей. "Spring. Подробное руководство." БХВ-Петербург, 2016. - книга представляет подробное руководство по использованию Spring Framework для разработки Java-приложений.
14. "Apache Tomcat Documentation." [Онлайн]. Доступно: <https://tomcat.apache.org/tomcat-9.0-doc/index.html>. - Официальная документация Apache Tomcat, содержащая информацию о настройке и использовании веб-контейнера Apache Tomcat.



ERD (Entity-Relationship Diagram)



**SQL-код для базы данных MedicalExamDB.sql**

```
CREATE DATABASE IF NOT EXISTS MedicalExamDB;  
USE MedicalExamDB;
```

```
CREATE TABLE Examiner ( -- Таблица для хранения информации об экзаменаторах  
id INT PRIMARY KEY, -- Идентификатор экзаменатора  
full_name VARCHAR(100), -- Полное имя экзаменатора  
medical_specialty VARCHAR(100) -- Медицинская специальность экзаменатора  
);
```

```
CREATE TABLE Examinable ( -- Таблица для хранения информации об экзаменуемых  
id INT PRIMARY KEY, -- Идентификатор экзаменуемого  
full_name VARCHAR(100), -- Полное имя экзаменуемого  
specialization VARCHAR(100) -- Специализация экзаменуемого  
accreditation_id INT, -- Внешний ключ для связи с таблицей Accreditation  
FOREIGN KEY (accreditation_id) REFERENCES Accreditation(id)  
);
```

```
CREATE TABLE Accreditation_type ( -- Таблица для хранения типов аккредитации  
id INT PRIMARY KEY, -- Идентификатор типа аккредитации  
type VARCHAR(50) -- Тип аккредитации  
);
```

```
CREATE TABLE Specialization ( -- Таблица для хранения информации о специализациях  
id INT PRIMARY KEY, -- Идентификатор специализации  
specialization_name VARCHAR(100) -- Название специализации  
);
```

```
CREATE TABLE Station_passport ( -- Таблица для хранения информации о паспортах станций  
id INT PRIMARY KEY, -- Идентификатор паспорта станции  
passport_name VARCHAR(100) -- Название паспорта станции  
);
```

```
CREATE TABLE Situations_List (-- Таблица для хранения информации о ситуациях
id INT PRIMARY KEY, -- Идентификатор ситуации
scenario_number INT, -- Номер сценария
station_passport_id INT, -- Идентификатор паспорта станции (внешний ключ)
description TEXT, -- Описание ситуации

FOREIGN KEY (station_passport_id) REFERENCES Station_passport(id) - Ссылка на паспорт
станции
);

CREATE TABLE Actions_list (-- Таблица для хранения информации о действиях
id INT PRIMARY KEY, -- Идентификатор действия
action_name VARCHAR(100), -- Название действия
group_characteristic VARCHAR(100) - Групповая характеристика действия
);

CREATE TABLE Protocol (-- Таблица для хранения информации о протоколах экзамена
id INT PRIMARY KEY, -- Идентификатор протокола
exam_date DATE, -- Дата экзамена
examiner_id INT, -- Идентификатор экзаменатора (внешний ключ)
examinable_id INT, -- Идентификатор экзаменуемого (внешний ключ)
accreditation_type_id INT, -- Идентификатор типа аккредитации (внешний ключ)
specialization_id INT, -- Идентификатор специализации (внешний ключ)
station_passport_id INT, -- Идентификатор паспорта станции (внешний ключ)
scenario_number_id INT, -- Идентификатор номера сценария (внешний ключ)
total_points INT, -- Общая сумма баллов
FOREIGN KEY (examiner_id) REFERENCES Examiner(id), -- Ссылка на экзаменатора
FOREIGN KEY (examinable_id) REFERENCES Examable(id), -- Ссылка на экзаменуемого
FOREIGN KEY (accreditation_type_id) REFERENCES Accreditation_type(id), -- Ссылка на тип
аккредитации
FOREIGN KEY (specialization_id) REFERENCES Specialization(id), -- Ссылка на
специализацию
FOREIGN KEY (station_passport_id) REFERENCES Station_passport(id), -- Ссылка на паспорт
станции
FOREIGN KEY (scenario_number_id) REFERENCES Scenario_numbers(id) - Ссылка на номер
сценария
);
```

```
CREATE TABLE Action_Scenario (-- Таблица для хранения информации о действиях в сценарии
id INT PRIMARY KEY, -- Идентификатор действия в сценарии
protocol_id INT, -- Идентификатор протокола (внешний ключ)
action_id INT, -- Идентификатор действия (внешний ключ)
result BOOLEAN, -- Результат действия
FOREIGN KEY (protocol_id) REFERENCES Protocol(id), -- Ссылка на протокол экзамена
FOREIGN KEY (action_id) REFERENCES Actions_list(id) -- Ссылка на действие
);
```

```
CREATE TABLE Exam (-- Таблица для хранения информации о деталях экзамена
id INT PRIMARY KEY, -- Идентификатор экзамена
protocol_id INT, -- Идентификатор протокола (внешний ключ)
examinable_id INT, -- Идентификатор экзаменуемого (внешний ключ)
FOREIGN KEY (protocol_id) REFERENCES Protocol(id), -- Ссылка на протокол экзамена
FOREIGN KEY (examinable_id) REFERENCES Examable(id) -- Ссылка на экзаменуемого
);
```

```
CREATE TABLE Exam_info (-- Таблица для хранения информации об экзамене
protocol_id INT, -- Идентификатор протокола (внешний ключ)
action_id INT, -- Идентификатор действия (внешний ключ)
result TINYINT(1) CHECK (result IN (0,1)), -- Результат экзамена
FOREIGN KEY (protocol_id) REFERENCES Protocol(id), -- Ссылка на протокол экзамена
FOREIGN KEY (action_id) REFERENCES Actions_list(id) -- Ссылка на действие
);
```

```
CREATE TABLE Authentication (-- Таблица для хранения информации об аутентификации
id INT PRIMARY KEY, -- Идентификатор аутентификации
examiner_id INT, -- Идентификатор экзаменатора (внешний ключ)
username VARCHAR(100), -- Логин пользователя
password_hash VARCHAR(255), -- Хеш пароля
FOREIGN KEY (examiner_id) REFERENCES Examiner(id) -- Ссылка на экзаменатора
);
```

**Java-код (Spring boot) сущностей (Entity)****1. Сущность Экзаменатор:**

```
package ru.surgu.medexambbackend.entity;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.AllArgsConstructor;
import lombok.Setter;

import java.util.List;

@Entity(name = "examiner")
@NoArgsConstructor
@Getter
@Setter
public class Examiner {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(name = "full_name", unique = true) // Ограничение уникальности
    для full_name
    private String fullName;

    @Column(name = "medical_specialty")
    private String medicalSpecialty;

    @OneToMany(mappedBy = "examinerId")
    private List<Protocol> protocols;

    public Examiner(String fullName, String medicalSpecialty) {
        this.fullName = fullName;
        this.medicalSpecialty = medicalSpecialty;
    }

    // Пустой конструктор и конструктор со всеми аргументами,
    // а также геттеры и сеттеры создаются при помощи lombok
}
```

2. Сущность Экзаменуемый:

```
package ru.surgu.medexambackend.entity;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Entity(name = "examinable")
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
public class Examinable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(name = "full_name", unique = true) // Ограничение уникальности
    для full_name
    private String fullName;

    private String specialization;

    @Column(name = "accreditation_type_id") //
    private int accreditationTypeId;

    public Examinable(String fullName, String specialization, int
accreditationTypeId) {
        this.fullName = fullName;
        this.specialization = specialization;
        this.accreditationTypeId = accreditationTypeId;
    }

    // Пустой конструктор и конструктор со всеми аргументами,
    // а также геттеры и сеттеры создаются при помощи lombok
}
```

3. AccreditationType

```
package ru.surgu.medexambbackend.entity;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import lombok.Data;

@Entity(name = "accreditation_type")
@Data
public class AccreditationType {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private String type;
    // Конструкторы, геттеры и сеттеры создаются при помощи lombok
}
```

4. Specialization

```
package ru.surgu.medexambbackend.entity;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Column;
import jakarta.persistence.OneToOne;
import jakarta.persistence.ManyToMany;
import lombok.*;

import java.util.Set;

@Entity(name = "specialization")
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
public class Specialization {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(name = "specialization_name ")
    private String specializationName; // название специализации

    // Пустой конструктор и конструктор со всеми аргументами,
    // а также геттеры и сеттеры создаются при помощи lombok
}
```


5. StationPassport

```
package ru.surgu.medexambbackend.entity;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Column;
import jakarta.persistence.OneToMany;
import lombok.Data;

import java.util.Set;

@Entity(name = "station_passport")
@Data
public class StationPassport {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(name = "passport_name")
    private String passportName; // название паспорта станции

    @OneToMany(mappedBy = "stationPassportId")
    private Set<Protocol> protocols;

    @OneToMany(mappedBy = "stationPassportId")
    private Set<SituationsList> situationsLists;

    // Конструкторы, геттеры и сеттеры создаются при помощи lombok
}
```

6. SituationsList

```
package ru.surgu.medexambackend.entity;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Column;
import jakarta.persistence.ManyToOne;
import jakarta.persistence.JoinColumn;
import lombok.Data;

@Entity(name = "situations_list")
@Data
public class SituationsList {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(name = "scenario_number")
    private int scenarioNumber;

    @ManyToOne
    @JoinColumn(name = "station_passport_id")
    private StationPassport stationPassportId;

    private String description;

    // Конструкторы, геттеры и сеттеры создаются при помощи lombok
}
```

7. ActionsList

```
package ru.surgu.medexambackend.entity;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Column;
import lombok.Data;

@Entity(name = "actions_list")
@Data
public class ActionsList {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(name = "action_name")
    private String actionName;

    @Column(name = "group_characteristic")
    private String groupCharacteristic;

    // Конструкторы, геттеры и сеттеры создаются при помощи lombok
}
```

8. Protocol

```
package ru.surgu.medexambackend.entity;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Column;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.ManyToOne;
import jakarta.validation.constraints.PastOrPresent;

import java.time.LocalDate;
import lombok.Data;

@Entity(name = "protocol")
@Data
public class Protocol {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(name = "exam_date")
    @PastOrPresent //условие, чтобы дата экзамена не могла быть позже текущей
    даты
    private LocalDate examDate;

    @ManyToOne
    @JoinColumn(name = "examiner_id", referencedColumnName = "id")
    private Examiner examinerId;

    @ManyToOne
    @JoinColumn(name = "examinable_id", referencedColumnName = "id")
    private Examable examinableId;

    @ManyToOne
    @JoinColumn(name = "accreditation_type_id", referencedColumnName = "id")
    private AccreditationType accreditationTypeId;

    @ManyToOne
    @JoinColumn(name = "specialization_id", referencedColumnName = "id")
    private Specialization specializationId;

    @ManyToOne
    @JoinColumn(name = "station_passport_id", referencedColumnName = "id")
    private StationPassport stationPassportId;

    @ManyToOne
    @JoinColumn(name = "scenario_number_id", referencedColumnName = "id")
    private SituationsList scenarioNumberId;

    @Column(name = "total_points")
    private int totalPoints;

    // Конструкторы, геттеры и сеттеры создаются при помощи lombok
}
```

9. ExamInfo

```
package ru.surgu.medexambackend.entity;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.ManyToOne;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.Column;
import lombok.Data;

@Entity(name = "exam_info")
@Data
public class ExamInfo {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @ManyToOne
    @JoinColumn(name = "protocol_id", referencedColumnName = "id")
    private Protocol protocolId;

    @ManyToOne
    @JoinColumn(name = "action_id", referencedColumnName = "id")
    private ActionsList actionId;

    @Column(columnDefinition = "BOOLEAN") // Это позволит базе данных
    ограничивать значения атрибута result только логическими (true или false)
    private boolean result;

    // Конструкторы, геттеры и сеттеры создаются при помощи lombok
}
```

10. ActionScenario

```
package ru.surgu.medexambackend.entity;

import jakarta.persistence.*;
import lombok.Data;

@Entity(name = "action_scenario")
@Data
public class ActionScenario {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    // Связь с действием
    @ManyToOne
    @JoinColumn(name = "action_id", referencedColumnName = "id")
    private ActionsList actionId;

    // Связь с номером сценария
    @ManyToOne
    @JoinColumn(name = "scenario_number_id", referencedColumnName = "id")
    private SituationsList scenarioNumberId;

    // Конструкторы, геттеры и сеттеры создаются при помощи lombok
}
```

11. Exam

```
package ru.surgu.medexambackend.entity;

import jakarta.persistence.*;
import lombok.Data;

@Entity(name = "exam")
@Data
public class Exam {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @ManyToOne
    @JoinColumn(name = "protocol_id", referencedColumnName = "id")
    private Protocol protocolId;

    @ManyToOne
    @JoinColumn(name = "examinable_id", referencedColumnName = "id")
    private Examinable examinableId;

    // Конструкторы, геттеры и сеттеры создаются при помощи lombok
}
```

12. Authentication

```
package ru.surgu.medexambackend.entity;

import jakarta.persistence.*;
import lombok.Data;

@Entity(name = "authentication")
@Data
public class Authentication {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @ManyToOne
    @JoinColumn(name = "examiner_id", referencedColumnName = "id")
    private Examiner examinerId;

    private String username;

    @Column(name = "password_hash")
    private String passwordHash;

    // Конструкторы, геттеры и сеттеры создаются при помощи lombok
}
```

**Java-код (Spring boot) репозитория (repository)****1. ExaminerRepository**

```
package ru.surgu.medexambackend.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import ru.surgu.medexambackend.entity.Examiner;

import java.util.List;
import java.util.Optional;

@Repository
public interface ExaminerRepository extends JpaRepository<Examiner, Integer> {

    // методы для стандартных операций CRUD предоставляются по умолчанию через
    // интерфейс JpaRepository

    // Получение списка экзаменов, проведенных конкретным экзаменатором
    List<Examiner> findExaminersByProtocolIsNotNull();

    // Получение списка всех экзаменаторов
    List<Examiner> findAll();
}
```

2. ExamableRepository

```
package ru.surgu.medexambackend.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;
import ru.surgu.medexambackend.entity.*;

import java.util.List;

@Repository
public interface ExamableRepository extends JpaRepository<Examable, Integer> {

    // методы для стандартных операций CRUD предоставляются по умолчанию через
    // интерфейс JpaRepository

    // Получение информации о том, в каких экзаменах участвовал конкретный
    // экзаменуемый
    @Query("SELECT e FROM exam e JOIN e.protocolId p JOIN p.examableId ex WHERE ex.id = :examableId")
    List<Exam> findExamsByExamableId(@Param("examableId") int examableId);

    // Получение информации о том, какие протоколы содержатся в экзаменах,
    // которые сдал экзаменуемый
    @Query("SELECT DISTINCT pr FROM protocol pr JOIN examable ex ON ex.id = pr.examableId.id")
    List<Protocol> findProtocolsByExamableId(@Param("examableId") int examableId);
}
```



```

// Получение информации о паспортах станций, в которых участвовал
экзаменуемый
@Query("SELECT sp FROM station_passport sp JOIN sp.protocols p JOIN
p.examinableId ex WHERE ex.id = :examinableId")
List<StationPassport>
findStationPassportsByExaminableId(@Param("examinableId") int examinableId);

// Получение информации о сценариях, в которых участвовал экзаменуемый
@Query("SELECT DISTINCT sp.passportName, sl.scenarioNumber " +
"FROM station_passport sp " +
"JOIN sp.protocols p " +
"JOIN p.examinableId ex " +
"JOIN situations_list sl ON sl.stationPassportId.id = sp.id " +
"WHERE ex.id = :examinableId")
List<Object[]> findScenarioNumbersByExaminableId(@Param("examinableId") int
examinableId);

// Получение информации о действиях, в которых участвовал экзаменуемый
@Query("SELECT al.actionName " +
"FROM protocol p " +
"JOIN exam_info ei ON p.id = ei.protocolId.id " +
"JOIN actions_list al ON ei.actionId.id = al.id " +
"WHERE p.examinableId.id = :examinableId")
List<String> findActionsByExaminableId(@Param("examinableId") int
examinableId);

// // Получение списка экзаменуемых для конкретного экзамена
@Query("SELECT ex.fullName " +
"FROM exam e " +
"JOIN protocol p ON e.protocolId.id = p.id " +
"JOIN examinable ex ON p.examinableId.id = ex.id " +
"WHERE e.id = :examId")
List<String> findExamineesByExamId(@Param("examId") int examId);
}

// Получение информации о том, какие протоколы содержатся в экзаменах, которые
сдал экзаменуемый
@Query("SELECT p FROM Protocol p WHERE p.exam.examinable.id
= :examinableId")
List<Protocol> findProtocolsByExaminableId(Long examinableId);

// Получение информации о паспортах станций, в которых участвовал
экзаменуемый
@Query("SELECT DISTINCT s.stationPassport FROM Protocol p JOIN
p.situationsLists s WHERE p.exam.examinable.id = :examinableId")
List<StationPassport> findStationPassportsByExaminableId(Long examinableId);

// Получение информации о сценариях, в которых участвовал экзаменуемый
@Query("SELECT DISTINCT s.scenarioNumber FROM Protocol p JOIN
p.situationsLists s WHERE p.exam.examinable.id = :examinableId")
List<SituationsList> findSituationsListsByExaminableId(Long examinableId);

// Получение информации о действиях, в которых участвовал экзаменуемый
@Query("SELECT DISTINCT a.action FROM Protocol p JOIN p.situationsLists s
JOIN s.actionScenarios a WHERE p.exam.examinable.id = :examinableId")
List<ActionsList> findActionsByExaminableId(Long examinableId);
}

```

3. AccreditationTypeRepository

```
package ru.surgu.medexambackend.repository;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import ru.surgu.medexambackend.entity.AccreditationType;

@Repository
public interface AccreditationTypeRepository extends
JpaRepository<AccreditationType, Integer> {

    // методы для стандартных операций CRUD предоставляются по умолчанию через
    // интерфейс JpaRepository
    // пользовательские методы (если будут добавлены в будущем) могут быть
    // добавлены здесь
}
```

4. StationPassportRepository

```
package ru.surgu.medexambackend.repository;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import ru.surgu.medexambackend.entity.StationPassport;

@Repository
public interface StationPassportRepository extends
JpaRepository<StationPassport, Integer> {
    // методы для стандартных операций CRUD предоставляются по умолчанию
    // через интерфейс JpaRepository
    // пользовательские методы (если будут добавлены в будущем) могут быть
    // добавлены здесь
}
```

5. SituationsListRepository

```
package ru.surgu.medexambackend.repository;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;
import ru.surgu.medexambackend.entity.SituationsList;

import java.util.List;

@Repository
public interface SituationsListRepository extends JpaRepository<SituationsList,
Integer> {

    // методы для стандартных операций CRUD предоставляются по умолчанию
    // через интерфейс JpaRepository

    // Получение списка номеров сценариев по заданным id протокола и id паспорта
    // станции
    @Query("SELECT s FROM situations_list s " +
        "JOIN protocol p ON p.scenarioNumberId = s " +
        "WHERE p.id = :protocolId " +
        "AND p.stationPassportId.id = :stationPassportId")
    List<SituationsList>
    findScenariosByProtocolIdAndStationPassportId(@Param("protocolId") int
protocolId, @Param("stationPassportId") int stationPassportId);
}
```

6. SpecializationRepository

```
package ru.surgu.medexambackend.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import ru.surgu.medexambackend.entity.Specialization;

@Repository
public interface SpecializationRepository extends JpaRepository<Specialization, Integer> {
    // методы для стандартных операций CRUD предоставляются по умолчанию
    // через интерфейс JpaRepository
    // пользовательские методы (если будут добавлены в будущем) могут быть
    // добавлены здесь
}
```

7. ActionsListRepository

```
package ru.surgu.medexambackend.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import ru.surgu.medexambackend.entity.ActionsList;

@Repository
public interface ActionsListRepository extends JpaRepository<ActionsList, Integer> {
    // методы для стандартных операций CRUD предоставляются по умолчанию
    // через интерфейс JpaRepository
    // пользовательские методы (если будут добавлены в будущем) могут быть
    // добавлены здесь
}
```

8. ProtocolRepository

```
package ru.surgu.medexambackend.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;
import ru.surgu.medexambackend.entity.*;

import java.util.List;

@Repository
public interface ProtocolRepository extends JpaRepository<Protocol, Integer> {

    // методы для стандартных операций CRUD предоставляются по умолчанию
    // через интерфейс JpaRepository

    // Получение информации о том, какой паспорт станции у заданного протокола
    @Query("SELECT p.stationPassportId FROM protocol p WHERE p.id = :protocolId")
    StationPassport findStationPassportByProtocolId(@Param("protocolId") int protocolId);
}
```

```
// Получение информации о том, какой сценарий у протокола для данного паспорта
станции
@Query("SELECT s FROM situations_list s " +
        "JOIN protocol p ON p.scenarioNumberId = s " +
        "WHERE p.id = :protocolId " +
        "AND p.stationPassportId.id = :stationPassportId")
List<SituationsList>
findScenariosByProtocolIdAndStationPassportId(@Param("protocolId") int
protocolId, @Param("stationPassportId") int stationPassportId);

// Получение списка действий для заданного id сценария, id паспорта станции,
id протокола
@Query("SELECT al FROM actions_list al " +
        "JOIN action_scenario as ON as.actionId = al " +
        "JOIN situations_list sl ON sl = as.scenarioNumberId " +
        "JOIN protocol p ON p.scenarioNumberId = sl " +
        "WHERE p.id = :protocolId " +
        "AND p.stationPassportId.id = :stationPassportId " +
        "AND sl.id = :scenarioId")
List<ActionsList>
findActionsByProtocolAndStationPassportAndScenario(@Param("protocolId") int
protocolId, @Param("stationPassportId") int stationPassportId,
@Param("scenarioId") int scenarioId);

// Получение результата выполнения по каждому действию для заданного
сценария, заданного паспорта станции, заданного протокола
@Query("SELECT ei FROM exam_info ei " +
        "WHERE ei.actionId IN :actionsList " +
        "AND ei.protocolId.id = :protocolId")
List<ExamInfo> findResultsByActionsAndProtocol(@Param("actionsList")
List<ActionsList> actionsList, @Param("protocolId") int protocolId);
}
```

9. ExamInfoRepository

```
package ru.surgu.medexambbackend.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import ru.surgu.medexambbackend.entity.ExamInfo;

@Repository
public interface ExamInfoRepository extends JpaRepository<ExamInfo, Integer> {
    // методы для стандартных операций CRUD предоставляются по умолчанию через
    интерфейс JpaRepository
    // пользовательские методы (если будут добавлены в будущем) могут быть
    добавлены здесь
}
```

10. ActionScenarioRepository

```
package ru.surgu.medexambackend.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import ru.surgu.medexambackend.entity.ActionScenario;

@Repository
public interface ActionScenarioRepository extends JpaRepository<ActionScenario, Integer> {

    // методы для стандартных операций CRUD предоставляются по умолчанию через
    // интерфейс JpaRepository
    // пользовательские методы (если будут добавлены в будущем) могут быть
    // добавлены здесь
}
```

11. ExamRepository

```
package ru.surgu.medexambackend.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;
import ru.surgu.medexambackend.entity.Exam;
import ru.surgu.medexambackend.entity.Protocol;

import java.util.List;

@Repository
public interface ExamRepository extends JpaRepository<Exam, Integer> {

    // методы для стандартных операций CRUD предоставляются по умолчанию
    // через интерфейс JpaRepository

    // Получение информации о том, какие протоколы содержит данный экзамен
    @Query("SELECT e.protocolId FROM exam e WHERE e.id = :examId")
    List<Protocol> findProtocolsByExamId(@Param("examId") int examId);

    // Получение информации о том, какие результаты содержатся в протоколах,
    // входящих в данный экзамен
    @Query("SELECT ei.result, ei.actionId FROM exam e JOIN exam_info ei ON e.protocolId.id = ei.protocolId.id WHERE e.id = :examId")
    List<Object[]> findResultsByExamId(@Param("examId") int examId);
}
```

12. AuthenticationRepository

```
package ru.surgu.medexambackend.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import ru.surgu.medexambackend.entity.Authentication;

@Repository
public interface AuthenticationRepository extends JpaRepository<Authentication,
Integer> {
    // методы для стандартных операций CRUD предоставляются по умолчанию
    // через интерфейс JpaRepository
    // пользовательские методы (если будут добавлены в будущем) могут быть
    // добавлены здесь
}
```

**Java-код (Spring boot) сервисов (services)****1. ExaminerService**

```
package ru.surgu.medexambackend.service;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import ru.surgu.medexambackend.entity.Examiner;
import ru.surgu.medexambackend.repository.ExaminerRepository;
import java.util.List;

@Service
public class ExaminerService {
    private final ExaminerRepository examinerRepository;
    @Autowired
    public ExaminerService(ExaminerRepository examinerRepository) {
        this.examinerRepository = examinerRepository;
    }
    // Сохранение экземпляра Examiner
    public Examiner saveExaminer(Examiner examiner) {
        return examinerRepository.save(examiner);
    }
    // Получение экземпляра Examiner по его идентификатору
    public Examiner getExaminerById(int id) {
        return examinerRepository.findById(id).orElse(null);
    }
    // Получение списка всех экземпляров Examiner
    public List<Examiner> getAllExaminers() {
        return examinerRepository.findAll();
    }
    // Удаление экземпляра Examiner по его идентификатору
    public void deleteExaminer(int id) {
        examinerRepository.deleteById(id);
    }
    // Получение списка экзаменов, проведенных конкретным экзаменатором
    public List<Examiner> findExaminersByProtocolsIsNotNull() {
        return examinerRepository.findExaminersByProtocolsIsNotNull();
    }
}
```

2. ExaminableService

```
package ru.surgu.medexambackend.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import ru.surgu.medexambackend.entity.Examinable;
import ru.surgu.medexambackend.entity.Exam;
import ru.surgu.medexambackend.entity.Protocol;
import ru.surgu.medexambackend.entity.StationPassport;
import ru.surgu.medexambackend.repository.ExaminableRepository;

import java.util.List;

@Service
public class ExaminableService {

    private final ExaminableRepository examinableRepository;
```

```
@Autowired

public ExamService(ExamRepository examRepository) {
    this.examRepository = examRepository;
}

// Сохранение экземпляра Exam
public Exam saveExam(Exam exam) {
    return examRepository.save(exam);
}

// Получение экземпляра Exam по его идентификатору
public Exam getExamById(int id) {
    return examRepository.findById(id).orElse(null);
}

// Получение списка всех экземпляров Exam
public List<Exam> getAllExams() {
    return examRepository.findAll();
}

// Удаление экземпляра Exam по его идентификатору
public void deleteExam(int id) {
    examRepository.deleteById(id);
}

// Получение списка экзаменов для заданного экзаменуемого
public List<Exam> findExamsByExamId(int examId) {
    return examRepository.findExamsByExamId(examId);
}

// Получение списка протоколов для заданного экзаменуемого
public List<Protocol> findProtocolsByExamId(int examId) {
    return examRepository.findProtocolsByExamId(examId);
}

// Получение списка паспортов станций для заданного экзаменуемого
public List<StationPassport> findStationPassportsByExamId(int
examId) {
    return
examRepository.findStationPassportsByExamId(examId);
}

// Получение списка сценариев для заданного экзаменуемого
public List<Object[]> findScenarioNumbersByExamId(int examId) {
    return
examRepository.findScenarioNumbersByExamId(examId);
}

// Получение списка действий для заданного экзаменуемого
public List<String> findActionsByExamId(int examId) {
    return examRepository.findActionsByExamId(examId);
}

// Получение списка экзаменуемых для конкретного экзамена
public List<String> findExamineesByExamId(int examId) {
    return examRepository.findExamineesByExamId(examId);
}
}
```


3. AccreditationTypeService

```
package ru.surgu.medexambbackend.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import ru.surgu.medexambbackend.entity.AccreditationType;
import ru.surgu.medexambbackend.repository.AccreditationTypeRepository;

import java.util.List;
import java.util.Optional;

@Service
public class AccreditationTypeService {

    private final AccreditationTypeRepository accreditationTypeRepository;

    @Autowired
    public AccreditationTypeService(AccreditationTypeRepository accreditationTypeRepository) {
        this.accreditationTypeRepository = accreditationTypeRepository;
    }

    // Сохранение нового типа аккредитации
    public AccreditationType saveAccreditationType(AccreditationType accreditationType) {
        return accreditationTypeRepository.save(accreditationType);
    }

    // Получение типа аккредитации по его ID
    public Optional<AccreditationType> getAccreditationTypeById(int id) {
        return accreditationTypeRepository.findById(id);
    }

    // Получение всех типов аккредитации
    public List<AccreditationType> getAllAccreditationTypes() {
        return accreditationTypeRepository.findAll();
    }

    // Удаление типа аккредитации по его ID
    public void deleteAccreditationType(int id) {
        accreditationTypeRepository.deleteById(id);
    }
}
```

4. StationPassportService

```
package ru.surgu.medexambackend.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import ru.surgu.medexambackend.entity.StationPassport;
import ru.surgu.medexambackend.repository.StationPassportRepository;

import java.util.List;
import java.util.Optional;

@Service
public class StationPassportService {

    private final StationPassportRepository stationPassportRepository;

    @Autowired
    public StationPassportService(StationPassportRepository
stationPassportRepository) {
        this.stationPassportRepository = stationPassportRepository;
    }

    // Стандартный метод для поиска всех паспортов станций
    public List<StationPassport> findAll() {
        return stationPassportRepository.findAll();
    }

    // Стандартный метод для поиска паспорта станции по его ID
    public Optional<StationPassport> findById(int id) {
        return stationPassportRepository.findById(id);
    }

    // Стандартный метод для сохранения паспорта станции
    public StationPassport save(StationPassport stationPassport) {
        return stationPassportRepository.save(stationPassport);
    }

    // Стандартный метод для удаления паспорта станции по его ID
    public void deleteById(int id) {
        stationPassportRepository.deleteById(id);
    }

    // Дополнительные методы, если такие есть в репозитории, могут быть
    добавлены здесь
}
```

5. SituationsListService

```
package ru.surgu.medexambackend.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import ru.surgu.medexambackend.entity.SituationsList;
import ru.surgu.medexambackend.repository.SituationsListRepository;

import java.util.List;
import java.util.Optional;

@Service
public class SituationsListService {

    private final SituationsListRepository situationsListRepository;

    @Autowired
    public SituationsListService(SituationsListRepository
situationsListRepository) {
        this.situationsListRepository = situationsListRepository;
    }

    // Получение всех сценариев
    public List<SituationsList> findAllSituationsLists() {
        return situationsListRepository.findAll();
    }

    // Получение сценария по его ID
    public Optional<SituationsList> findSituationsListById(int id) {
        return situationsListRepository.findById(id);
    }

    // Сохранение сценария
    public SituationsList saveSituationsList(SituationsList situationsList) {
        return situationsListRepository.save(situationsList);
    }

    // Удаление сценария по его ID
    public void deleteSituationsListById(int id) {
        situationsListRepository.deleteById(id);
    }

    // Получение списка сценариев по заданным id протокола и id паспорта станции
    public List<SituationsList>
findScenariosByProtocolIdAndStationPassportId(int protocolId, int
stationPassportId) {
        return
situationsListRepository.findScenariosByProtocolIdAndStationPassportId(protocolI
d, stationPassportId);
    }
}
```

6. SpecializationService

```
package ru.surgu.medexambackend.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import ru.surgu.medexambackend.entity.Specialization;
import ru.surgu.medexambackend.repository.SpecializationRepository;

import java.util.List;
import java.util.Optional;

@Service
public class SpecializationService {

    private final SpecializationRepository specializationRepository;

    @Autowired
    public SpecializationService(SpecializationRepository
specializationRepository) {
        this.specializationRepository = specializationRepository;
    }

    //Получение всех специализаций
    public List<Specialization> getAllSpecializations() {
        return specializationRepository.findAll();
    }

    //Получение специализации по ее ID
    public Optional<Specialization> getSpecializationById(int id) {
        return specializationRepository.findById(id);
    }

    //Сохранение специализации
    public Specialization saveSpecialization(Specialization specialization) {
        return specializationRepository.save(specialization);
    }

    //Удаление специализации по ее ID
    public void deleteSpecializationById(int id) {
        specializationRepository.deleteById(id);
    }
}
```

7. ActionsListService

```
package ru.surgu.medexambackend.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import ru.surgu.medexambackend.entity.ActionsList;
import ru.surgu.medexambackend.repository.ActionsListRepository;

import java.util.List;
import java.util.Optional;

@Service
public class ActionsListService {

    private final ActionsListRepository actionsListRepository;

    @Autowired
    public ActionsListService(ActionsListRepository actionsListRepository) {
        this.actionsListRepository = actionsListRepository;
    }

    // Сохранение действия
    public ActionsList saveAction(ActionsList action) {
        return actionsListRepository.save(action);
    }

    // Получение действия по его ID
    public ActionsList getActionById(int id) {
        Optional<ActionsList> optionalAction =
actionsListRepository.findById(id);
        return optionalAction.orElse(null);
    }

    // Получение всех действий
    public List<ActionsList> getAllActions() {
        return actionsListRepository.findAll();
    }

    // Удаление действия по его ID
    public void deleteAction(int id) {
        actionsListRepository.deleteById(id);
    }
}
```

8. ProtocolService

```
package ru.surgu.medexambbackend.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import ru.surgu.medexambbackend.entity.*;
import ru.surgu.medexambbackend.repository.ProtocolRepository;

import java.util.List;

@Service
public class ProtocolService {

    private final ProtocolRepository protocolRepository;

    @Autowired
    public ProtocolService(ProtocolRepository protocolRepository) {
        this.protocolRepository = protocolRepository;
    }

    // Стандартный метод для сохранения протокола
    public Protocol saveProtocol(Protocol protocol) {
        return protocolRepository.save(protocol);
    }

    // Стандартный метод для поиска протокола по его ID
    public Protocol findProtocolById(int id) {
        return protocolRepository.findById(id).orElse(null);
    }

    // Метод для поиска паспорта станции по ID протокола
    public StationPassport findStationPassportByProtocolId(int protocolId) {
        return protocolRepository.findStationPassportByProtocolId(protocolId);
    }

    // Метод для поиска сценариев по ID протокола и ID паспорта станции
    public List<SituationsList>
    findScenariosByProtocolIdAndStationPassportId(int protocolId, int
    stationPassportId) {
        return
        protocolRepository.findScenariosByProtocolIdAndStationPassportId(protocolId,
        stationPassportId);
    }

    // Метод для поиска действий по ID протокола, ID паспорта станции и ID
    сценария
    public List<ActionsList>
    findActionsByProtocolAndStationPassportAndScenario(int protocolId, int
    stationPassportId, int scenarioId) {
        return
        protocolRepository.findActionsByProtocolAndStationPassportAndScenario(protocolId
        , stationPassportId, scenarioId);
    }

    // Метод для поиска результатов выполнения действий по ID протокола и списку
    действий
    public List<ExamInfo> findResultsByActionsAndProtocol(List<ActionsList>
    actionsList, int protocolId) {
        return protocolRepository.findResultsByActionsAndProtocol(actionsList,
        protocolId);
    }
}
```

9. ExamInfoService

```
package ru.surgu.medexambackend.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import ru.surgu.medexambackend.entity.ExamInfo;
import ru.surgu.medexambackend.repository.ExamInfoRepository;

import java.util.List;
import java.util.Optional;

@Service
public class ExamInfoService {

    private final ExamInfoRepository examInfoRepository;

    @Autowired
    public ExamInfoService(ExamInfoRepository examInfoRepository) {
        this.examInfoRepository = examInfoRepository;
    }

    // Сохранение информации о экзамене
    public ExamInfo saveExamInfo(ExamInfo examInfo) {
        return examInfoRepository.save(examInfo);
    }

    // Получение информации о экзамене по его ID
    public ExamInfo getExamInfoById(int id) {
        Optional<ExamInfo> optionalExamInfo = examInfoRepository.findById(id);
        return optionalExamInfo.orElse(null);
    }

    // Получение списка всех записей о экзамене
    public List<ExamInfo> getAllExamInfos() {
        return examInfoRepository.findAll();
    }

    // Удаление информации о экзамене по его ID
    public void deleteExamInfo(int id) {
        examInfoRepository.deleteById(id);
    }
}
```

10. ActionScenarioService

```
package ru.surgu.medexambackend.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import ru.surgu.medexambackend.entity.ActionScenario;
import ru.surgu.medexambackend.repository.ActionScenarioRepository;

import java.util.List;
import java.util.Optional;

@Service
public class ActionScenarioService {

    private final ActionScenarioRepository actionScenarioRepository;

    @Autowired
    public ActionScenarioService(ActionScenarioRepository
actionScenarioRepository) {
        this.actionScenarioRepository = actionScenarioRepository;
    }

    // Сохранение экземпляра ActionScenario
    public ActionScenario saveActionScenario(ActionScenario actionScenario) {
        return actionScenarioRepository.save(actionScenario);
    }

    // Получение экземпляра ActionScenario по его идентификатору
    public ActionScenario getActionScenarioById(int id) {
        Optional<ActionScenario> optionalActionScenario =
actionScenarioRepository.findById(id);
        return optionalActionScenario.orElse(null);
    }

    // Получение списка всех экземпляров ActionScenario
    public List<ActionScenario> getAllActionScenarios() {
        return actionScenarioRepository.findAll();
    }

    // Удаление экземпляра ActionScenario по его идентификатору
    public void deleteActionScenario(int id) {
        actionScenarioRepository.deleteById(id);
    }
}
```


11. ExamService

```
package ru.surgu.medexambackend.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import ru.surgu.medexambackend.entity.Exam;
import ru.surgu.medexambackend.entity.Protocol;
import ru.surgu.medexambackend.repository.ExamRepository;

import java.util.List;

@Service
public class ExamService {

    private final ExamRepository examRepository;

    @Autowired
    public ExamService(ExamRepository examRepository) {
        this.examRepository = examRepository;
    }

    // Стандартный метод сохранения экзамена
    public Exam saveExam(Exam exam) {
        return examRepository.save(exam);
    }

    // Стандартный метод поиска экзамена по его ID
    public Exam getExamById(int id) {
        return examRepository.findById(id).orElse(null);
    }

    // Стандартный метод получения всех экзаменов
    public List<Exam> getAllExams() {
        return examRepository.findAll();
    }

    // Стандартный метод удаления экзамена по его ID
    public void deleteExam(int id) {
        examRepository.deleteById(id);
    }

    // Получение протоколов для заданного экзамена
    public List<Protocol> findProtocolsByExamId(int examId) {
        return examRepository.findProtocolsByExamId(examId);
    }

    // Получение результатов для заданного экзамена
    public List<Object[]> findResultsByExamId(int examId) {
        return examRepository.findResultsByExamId(examId);
    }
}
```

12. AuthenticationService

```
package ru.surgu.medexambackend.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import ru.surgu.medexambackend.entity.Authentication;
import ru.surgu.medexambackend.repository.AuthenticationRepository;

import java.util.List;
import java.util.Optional;

@Service
public class AuthenticationService {

    private final AuthenticationRepository authenticationRepository;

    @Autowired
    public AuthenticationService(AuthenticationRepository authenticationRepository) {
        this.authenticationRepository = authenticationRepository;
    }

    // Сохранение сущности Authentication
    public Authentication saveAuthentication(Authentication authentication) {
        return authenticationRepository.save(authentication);
    }

    // Получение сущности Authentication по ее идентификатору
    public Optional<Authentication> getAuthenticationById(int id) {
        return authenticationRepository.findById(id);
    }

    // Получение списка всех сущностей Authentication
    public List<Authentication> getAllAuthentications() {
        return authenticationRepository.findAll();
    }

    // Удаление сущности Authentication по ее идентификатору
    public void deleteAuthentication(int id) {
        authenticationRepository.deleteById(id);
    }
}
```

**Java-код (Spring boot) контроллеров (controller)****1. ExaminerController**

```
package ru.surgu.medexambackend.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import ru.surgu.medexambackend.entity.Examiner;
import ru.surgu.medexambackend.repository.ExaminerRepository;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.responses.ApiResponses;

import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/api")
public class ExaminerController {

    private final ExaminerRepository examinerRepository;

    @Autowired
    public ExaminerController(ExaminerRepository examinerRepository) {
        this.examinerRepository = examinerRepository;
    }

    // Получение всех экзаменаторов
    @Operation(summary = "Get all examiners")
    @GetMapping("/examiners")
    public ResponseEntity<List<Examiner>> getAllExaminers() {
        try {
            List<Examiner> examiners = examinerRepository.findAll();
            if (examiners.isEmpty()) {
                return new ResponseEntity<>(HttpStatus.NO_CONTENT);
            }
            return new ResponseEntity<>(examiners, HttpStatus.OK);
        } catch (Exception e) {
```

```
        return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}

// Получение экзаменатора по его ID
@Operation(summary = "Get examiner by ID")
@ApiResponses(value = {
    @ApiResponse(responseCode = "200", description = "Found the
examiner"),
    @ApiResponse(responseCode = "404", description = "Examiner not
found")
})
@GetMapping("/examiners/{id}")
public ResponseEntity<Examiner> getExaminerById(@PathVariable("id") int id)
{
    Optional<Examiner> optionalExaminer = examinerRepository.findById(id);
    return optionalExaminer.map(examiner -> new ResponseEntity<>(examiner,
HttpStatus.OK))
        .orElseGet(() -> new ResponseEntity<>(HttpStatus.NOT_FOUND));
}

// Получение списка экзаменаторов, проведенных как минимум одним экзаменом
@Operation(summary = "Get examiners with protocols")
@GetMapping("/examiners/with-protocols")
public ResponseEntity<List<Examiner>> getExaminersWithProtocols() {
    try {
        List<Examiner> examiners =
examinerRepository.findExaminersByProtocolsIsNotNull();
        if (examiners.isEmpty()) {
            return new ResponseEntity<>(HttpStatus.NO_CONTENT);
        }
        return new ResponseEntity<>(examiners, HttpStatus.OK);
    } catch (Exception e) {
        return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

```
// Создание нового экзаменатора
@Operation(summary = "Create a new examiner")
@PostMapping("/examiners")
public ResponseEntity<Examiner> createExaminer(@RequestBody Examiner
examiner) {
    try {
        Examiner createdExaminer = examinerRepository.save(examiner);
        return new ResponseEntity<>(createdExaminer, HttpStatus.CREATED);
    } catch (Exception e) {
        return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}

// Обновление информации об экзаменаторе по его ID
@Operation(summary = "Update examiner by ID")
@PutMapping("/examiners/{id}")
public ResponseEntity<Examiner> updateExaminer(@PathVariable("id") int id,
@RequestBody Examiner examinerDetails) {
    try {
        Optional<Examiner> optionalExaminer =
examinerRepository.findById(id);
        if (optionalExaminer.isPresent()) {
            Examiner updatedExaminer = optionalExaminer.get();
            updatedExaminer.setFullName(examinerDetails.getFullName());

updatedExaminer.setMedicalSpecialty(examinerDetails.getMedicalSpecialty());
            return new
ResponseEntity<>(examinerRepository.save(updatedExaminer), HttpStatus.OK);
        } else {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    } catch (Exception e) {
        return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

```
// Удаление экзаменатора по его ID
@Operation(summary = "Delete examiner by ID")
@DeleteMapping("/examiners/{id}")
public ResponseEntity<HttpStatus> deleteExaminer(@PathVariable("id") int id)
{
    try {
        examinerRepository.deleteById(id);
        return new ResponseEntity<>(HttpStatus.NO_CONTENT);
    } catch (Exception e) {
        return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

```
//GET /api/examiners - Получение списка всех экзаменаторов.
//GET /api/examiners/{id} - Получение информации об экзаменаторе по его ID.
//GET /api/examiners/with-protocols - Получение списка экзаменаторов,
проведенных как минимум одним экзаменом.
//POST /api/examiners - Создание нового экзаменатора.
//PUT /api/examiners/{id} - Обновление информации об экзаменаторе по его ID.
//DELETE /api/examiners/{id} - Удаление экзаменатора по его ID.
```

2. ExaminableController

```
package ru.surgu.medexambackend.controller;

import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.responses.ApiResponses;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import ru.surgu.medexambackend.entity.*;
import ru.surgu.medexambackend.service.ExaminableService;

import java.util.List;

@RestController
@RequestMapping("/examinable")
public class ExaminableController {

    private final ExaminableService examinableService;

    @Autowired
    public ExaminableController(ExaminableService examinableService) {
        this.examinableService = examinableService;
    }

    // Эндпоинт для сохранения экземпляра Examinable
    @Operation(summary = "Save an Examinable")
    @PostMapping("/save")
    public ResponseEntity<Examinable> saveExaminable(@RequestBody Examinable
examinable) {
        Examinable savedExaminable =
examinableService.saveExaminable(examinable);
        return ResponseEntity.ok(savedExaminable);
    }

    // Эндпоинт для получения экземпляра Examinable по его идентификатору
    @Operation(summary = "Get an Examinable by ID")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Found the
Examinable"),
        @ApiResponse(responseCode = "404", description = "Examinable not
found")
    })
    @GetMapping("/{id}")
    public ResponseEntity<Examinable> getExaminableById(@PathVariable int id) {
        Examinable examinable = examinableService.getExaminableById(id);
        if (examinable != null) {
            return ResponseEntity.ok(examinable);
        } else {
            return ResponseEntity.notFound().build();
        }
    }

    // Эндпоинт для получения списка всех экземпляров Examinable
    @Operation(summary = "Get all Examinables")
    @GetMapping("/all")
    public ResponseEntity<List<Examinable>> getAllExaminables() {
        List<Examinable> examinables = examinableService.getAllExaminables();
    }
}
```

```
        return ResponseEntity.ok(examinables);
    }

    // Эндпоинт для удаления экземпляра Examinable по его идентификатору
    @Operation(summary = "Delete an Examinable by ID")
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteExaminable(@PathVariable int id) {
        examinableService.deleteExaminable(id);
        return ResponseEntity.noContent().build();
    }

    // Эндпоинт для получения списка экзаменов для заданного экзаменуемого
    @Operation(summary = "Get Exams by Examinable ID")
    @GetMapping("/{examinableId}/exams")
    public ResponseEntity<List<Exam>> getExamsByExaminableId(@PathVariable int
examinableId) {
        List<Exam> exams =
examinableService.findExamsByExaminableId(examinableId);
        return ResponseEntity.ok(exams);
    }

    // Эндпоинт для получения списка протоколов для заданного экзаменуемого
    @Operation(summary = "Get Protocols by Examinable ID")
    @GetMapping("/{examinableId}/protocols")
    public ResponseEntity<List<Protocol>>
getProtocolsByExaminableId(@PathVariable int examinableId) {
        List<Protocol> protocols =
examinableService.findProtocolsByExaminableId(examinableId);
        return ResponseEntity.ok(protocols);
    }

    // Эндпоинт для получения списка паспортов станций для заданного
экзаменуемого
    @Operation(summary = "Get Station Passports by Examinable ID")
    @GetMapping("/{examinableId}/station-passports")
    public ResponseEntity<List<StationPassport>>
getStationPassportsByExaminableId(@PathVariable int examinableId) {
        List<StationPassport> stationPassports =
examinableService.findStationPassportsByExaminableId(examinableId);
        return ResponseEntity.ok(stationPassports);
    }

    // Эндпоинт для получения списка сценариев для заданного экзаменуемого
    @Operation(summary = "Get Scenarios by Examinable ID")
    @GetMapping("/{examinableId}/scenarios")
    public ResponseEntity<List<Object[]>>
getScenarioNumbersByExaminableId(@PathVariable int examinableId) {
        List<Object[]> scenarios =
examinableService.findScenarioNumbersByExaminableId(examinableId);
        return ResponseEntity.ok(scenarios);
    }

    // Эндпоинт для получения списка действий для заданного экзаменуемого
    @Operation(summary = "Get Actions by Examinable ID")
    @GetMapping("/{examinableId}/actions")
    public ResponseEntity<List<String>> getActionsByExaminableId(@PathVariable
int examinableId) {
        List<String> actions =
examinableService.findActionsByExaminableId(examinableId);
        return ResponseEntity.ok(actions);
    }
}
```



```
// Эндпоинт для получения списка экзаменуемых для конкретного экзамена
@Operation(summary = "Get Examinees by Exam ID")
@GetMapping("/exams/{examId}/examinees")
public ResponseEntity<List<String>> getExamineesByExamId(@PathVariable int
examId) {
    List<String> examinees =
examinableService.findExamineesByExamId(examId);
    return ResponseEntity.ok(examinees);
}
}
```

```
//Сохранение экземпляра Examinaable:
//POST /examinable/save
//Принимает POST-запрос с телом, содержащим экземпляр Examinaable, сохраняет его
в базе данных и возвращает сохраненный экземпляр.
```

```
//Получение экземпляра Examinaable по его идентификатору:
//GET /examinable/{id}
//Принимает GET-запрос с идентификатором экзаменуемого в качестве пути,
// возвращает экземпляр Examinaable с соответствующим идентификатором,
// если он существует.
```

```
//Получение списка всех экземпляров Examinaable:
//GET /examinable/all
//Возвращает список всех экземпляров Examinaable.
```

```
//Удаление экземпляра Examinaable по его идентификатору:
//DELETE /examinable/{id}
//Принимает DELETE-запрос с идентификатором экзаменуемого в качестве пути,
//удаляет соответствующий экземпляр из базы данных.
```

```
//Получение списка экзаменов для заданного экзаменуемого:
//GET /examinable/{examinableId}/exams
//Возвращает список экзаменов для заданного экзаменуемого.
```

```
//Получение списка протоколов для заданного экзаменуемого:
//GET /examinable/{examinableId}/protocols
//Возвращает список протоколов для заданного экзаменуемого.
```

```
//Получение списка паспортов станций для заданного экзаменуемого:
//GET /examinable/{examinableId}/station-passports
//Возвращает список паспортов станций для заданного экзаменуемого.
```

```
//Получение списка сценариев для заданного экзаменуемого:
//GET /examinable/{examinableId}/scenarios
//Возвращает список сценариев для заданного экзаменуемого.
```

```
//Получение списка действий для заданного экзаменуемого:
//GET /examinable/{examinableId}/actions
//Возвращает список действий для заданного экзаменуемого.
```

```
//Получение списка экзаменуемых для конкретного экзамена:
//GET /examinable/exams/{examId}/examinees
//Возвращает список экзаменуемых для конкретного экзамена.
```

3. **AccreditationTypeController**

```
package ru.surgu.medexambackend.controller;

import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.responses.ApiResponses;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import ru.surgu.medexambackend.entity.AccreditationType;
import ru.surgu.medexambackend.service.AccreditationTypeService;

import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/api/accreditation-types")
public class AccreditationTypeController {

    private final AccreditationTypeService accreditationTypeService;

    @Autowired
    public AccreditationTypeController(AccreditationTypeService accreditationTypeService) {
        this.accreditationTypeService = accreditationTypeService;
    }

    // Эндпоинт для сохранения нового типа аккредитации
    @Operation(summary = "Save a new accreditation type")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "201", description = "Accreditation type saved"),
        @ApiResponse(responseCode = "400", description = "Invalid input")
    })
    @PostMapping
    public ResponseEntity<AccreditationType> saveAccreditationType(@RequestBody AccreditationType accreditationType) {
        AccreditationType savedAccreditationType = accreditationTypeService.saveAccreditationType(accreditationType);
        return new ResponseEntity<>(savedAccreditationType, HttpStatus.CREATED);
    }

    // Эндпоинт для получения всех типов аккредитации
    @Operation(summary = "Get all accreditation types")
    @GetMapping
    public ResponseEntity<List<AccreditationType>> getAllAccreditationTypes() {
        List<AccreditationType> accreditationTypes = accreditationTypeService.getAllAccreditationTypes();
        return ResponseEntity.ok(accreditationTypes);
    }

    // Эндпоинт для получения типа аккредитации по его ID
    @Operation(summary = "Get accreditation type by ID")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Found the accreditation type"),
        @ApiResponse(responseCode = "404", description = "Accreditation type not found")
    })
```

```

    })
    @GetMapping("/{id}")
    public ResponseEntity<AccreditationType>
getAccreditationTypeById(@PathVariable int id) {
    Optional<AccreditationType> accreditationType =
accreditationTypeService.getAccreditationTypeById(id);
    return accreditationType.map(ResponseEntity::ok)
        .orElseGet(() -> ResponseEntity.notFound().build());
}

// Эндпоинт для удаления типа аккредитации по его ID
@Operation(summary = "Delete accreditation type by ID")
@ApiResponses(value = {
    @ApiResponse(responseCode = "204", description = "Accreditation type
deleted"),
    @ApiResponse(responseCode = "404", description = "Accreditation type
not found")
})
@DeleteMapping("/{id}")
public ResponseEntity<Void> deleteAccreditationType(@PathVariable int id) {
    accreditationTypeService.deleteAccreditationType(id);
    return ResponseEntity.noContent().build();
}
}

//POST /api/accreditation-types: Создание нового типа аккредитации.
// Принимает объект типа AccreditationType в теле запроса и сохраняет его в базе
данных.
// Возвращает созданный объект типа AccreditationType и статус ответа 201
(Created).

//GET /api/accreditation-types: Получение всех типов аккредитации.
// Возвращает список всех типов аккредитации, сохраненных в базе данных, и
статус ответа 200 (OK).

//GET /api/accreditation-types/{id}: Получение типа аккредитации по его ID.
// Принимает ID типа аккредитации в качестве переменной пути и возвращает объект
типа AccreditationType с указанным ID,
// если такой существует. В случае отсутствия объекта возвращает статус ответа
404 (Not Found).

//DELETE /api/accreditation-types/{id}: Удаление типа аккредитации по его ID.
// Принимает ID типа аккредитации в качестве переменной пути и удаляет
соответствующий объект из базы данных.
// Возвращает статус ответа 204 (No Content).

```

4. **StationPassportController**

```
package ru.surgu.medexambackend.controller;

import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.responses.ApiResponses;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import ru.surgu.medexambackend.entity.StationPassport;
import ru.surgu.medexambackend.service.StationPassportService;

import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/api/station-passports")
public class StationPassportController {

    private final StationPassportService stationPassportService;

    @Autowired
    public StationPassportController(StationPassportService
stationPassportService) {
        this.stationPassportService = stationPassportService;
    }

    // Обработка запроса на получение всех паспортов станций
    @Operation(summary = "Get all station passports")
    @GetMapping
    public ResponseEntity<List<StationPassport>> getAllStationPassports() {
        List<StationPassport> stationPassports =
stationPassportService.findAll();
        return ResponseEntity.ok(stationPassports);
    }

    // Обработка запроса на получение паспорта станции по его ID
    @Operation(summary = "Get a station passport by ID")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Found the station
passport"),
        @ApiResponse(responseCode = "404", description = "Station passport
not found")
    })
    @GetMapping("/{id}")
    public ResponseEntity<StationPassport>
getStationPassportById(@PathVariable("id") int id) {
        Optional<StationPassport> stationPassportOptional =
stationPassportService.findById(id);
        return stationPassportOptional.map(ResponseEntity::ok).orElseGet(() ->
ResponseEntity.notFound().build());
    }

    // Обработка запроса на создание нового паспорта станции
    @Operation(summary = "Create a new station passport")
    @PostMapping
```

```
public ResponseEntity<StationPassport> createStationPassport(@RequestBody
StationPassport stationPassport) {

    StationPassport savedStationPassport =
stationPassportService.save(stationPassport);
    return
ResponseEntity.status(HttpStatus.CREATED).body(savedStationPassport);
}

// Обработка запроса на удаление паспорта станции по его ID
@Operation(summary = "Delete a station passport by ID")
@ApiResponses(value = {
    @ApiResponse(responseCode = "204", description = "Station passport
deleted"),
    @ApiResponse(responseCode = "404", description = "Station passport
not found")
})
>DeleteMapping("/{id}")
public ResponseEntity<Void> deleteStationPassport(@PathVariable("id") int
id) {
    stationPassportService.deleteById(id);
    return ResponseEntity.noContent().build();
}
}

//GET /api/station-passports - Получение списка всех паспортов станций.
//GET /api/station-passports/{id} - Получение информации о конкретном паспорте
станции по его идентификатору.
//POST /api/station-passports - Создание нового паспорта станции.
//DELETE /api/station-passports/{id} - Удаление паспорта станции по его
идентификатору.
```

5. **SituationsListController**

```

package ru.surgu.medexambackend.controller;

import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.responses.ApiResponses;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import ru.surgu.medexambackend.entity.SituationsList;
import ru.surgu.medexambackend.service.SituationsListService;

import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/api/situations-lists")
public class SituationsListController {

    private final SituationsListService situationsListService;

    @Autowired
    public SituationsListController(SituationsListService situationsListService)
    {
        this.situationsListService = situationsListService;
    }

    // Получение всех сценариев
    @Operation(summary = "Get all situations lists")
    @GetMapping
    public ResponseEntity<List<SituationsList>> getAllSituationsLists() {
        List<SituationsList> situationsLists =
situationsListService.findAllSituationsLists();
        return ResponseEntity.ok(situationsLists);
    }

    // Получение сценария по его ID
    @Operation(summary = "Get a situations list by ID")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Found the
situations list"),
        @ApiResponse(responseCode = "404", description = "Situations list
not found")
    })
    @GetMapping("/{id}")
    public ResponseEntity<SituationsList>
getSituationsListById(@PathVariable("id") int id) {
        Optional<SituationsList> situationsList =
situationsListService.findSituationsListById(id);
        return situationsList.map(ResponseEntity::ok).orElseGet(() ->
ResponseEntity.notFound().build());
    }

    // Создание нового сценария
    @Operation(summary = "Create a new situations list")
    @PostMapping
    public ResponseEntity<SituationsList> createSituationsList(@RequestBody
SituationsList situationsList) {

```

```

        SituationsList createdSituationsList =
situationsListService.saveSituationsList(situationsList);
        return new ResponseEntity<>(createdSituationsList, HttpStatus.CREATED);
    }

    // Обновление информации о сценарии по его ID
    @Operation(summary = "Update a situations list by ID")
    @PutMapping("/{id}")
    public ResponseEntity<SituationsList>
updateSituationsList(@PathVariable("id") int id, @RequestBody SituationsList
updatedSituationsList) {
        if (!situationsListService.findSituationsListById(id).isPresent()) {
            return ResponseEntity.notFound().build();
        }
        updatedSituationsList.setId(id);
        SituationsList savedSituationsList =
situationsListService.saveSituationsList(updatedSituationsList);
        return ResponseEntity.ok(savedSituationsList);
    }

    // Удаление сценария по его ID
    @Operation(summary = "Delete a situations list by ID")
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteSituationsList(@PathVariable("id") int id)
{
        if (!situationsListService.findSituationsListById(id).isPresent()) {
            return ResponseEntity.notFound().build();
        }
        situationsListService.deleteSituationsListById(id);
        return ResponseEntity.noContent().build();
    }

    // Получение списка сценариев по заданным id протокола и id паспорта станции
    @Operation(summary = "Get scenarios by protocol ID and station passport ID")
    @GetMapping("/protocol/{protocolId}/station-passport/{stationPassportId}")
    public ResponseEntity<List<SituationsList>>
getScenariosByProtocolIdAndStationPassportId(@PathVariable int protocolId,
@PathVariable int stationPassportId) {
        List<SituationsList> scenarios =
situationsListService.findScenariosByProtocolIdAndStationPassportId(protocolId,
stationPassportId);
        return ResponseEntity.ok(scenarios);
    }
}

//GET /api/situations-lists: Получение списка всех сценариев.
//GET /api/situations-lists/{id}: Получение сведений о конкретном сценарии по
его ID.
//POST /api/situations-lists: Создание нового сценария.
//PUT /api/situations-lists/{id}: Обновление информации о существующем сценарии
по его ID.
//DELETE /api/situations-lists/{id}: Удаление сценария по его ID.

//GET
/api/situations-lists/protocol/{protocolId}/station-passport/{stationPassportId}
:
// Получение списка сценариев по заданным id протокола и id паспорта станции.

```

6. **SpecializationController**

```

package ru.surgu.medexambackend.controller;

import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.responses.ApiResponses;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import ru.surgu.medexambackend.entity.Specialization;
import ru.surgu.medexambackend.service.SpecializationService;

import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/specializations")
public class SpecializationController {

    private final SpecializationService specializationService;

    @Autowired
    public SpecializationController(SpecializationService specializationService)
    {
        this.specializationService = specializationService;
    }

    // Эндпоинт для получения всех специализаций
    @Operation(summary = "Get all specializations")
    @GetMapping
    public ResponseEntity<List<Specialization>> getAllSpecializations() {
        List<Specialization> specializations =
specializationService.getAllSpecializations();
        return ResponseEntity.ok(specializations);
    }

    // Эндпоинт для получения специализации по ID
    @Operation(summary = "Get specialization by ID")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Found the
specialization"),
        @ApiResponse(responseCode = "404", description = "Specialization not
found")
    })
    @GetMapping("/{id}")
    public ResponseEntity<Specialization> getSpecializationById(@PathVariable
int id) {
        Optional<Specialization> specialization =
specializationService.getSpecializationById(id);
        return specialization.map(ResponseEntity::ok).orElseGet(() ->
ResponseEntity.notFound().build());
    }

    // Эндпоинт для сохранения специализации
    @Operation(summary = "Save specialization")
    @PostMapping
    public ResponseEntity<Specialization> saveSpecialization(@RequestBody
Specialization specialization) {
        Specialization savedSpecialization =
specializationService.saveSpecialization(specialization);

```



```
        return
        ResponseEntity.status(HttpStatus.CREATED).body(savedSpecialization);
    }

    // Эндпоинт для удаления специализации по ID
    @Operation(summary = "Delete specialization by ID")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "204", description = "Specialization
deleted"),
        @ApiResponse(responseCode = "404", description = "Specialization not
found")
    })
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteSpecializationById(@PathVariable int id) {
        specializationService.deleteSpecializationById(id);
        return ResponseEntity.noContent().build();
    }
}

//GET /specializations - возвращает все специализации.
//GET /specializations/{id} - возвращает специализацию по ее ID.
//POST /specializations - сохраняет новую специализацию.
//DELETE /specializations/{id} - удаляет специализацию по ее ID.
```

7. **ActionsListController**

```

package ru.surgu.medexambackend.controller;

import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.responses.ApiResponses;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import ru.surgu.medexambackend.entity.ActionsList;
import ru.surgu.medexambackend.service.ActionsListService;

import java.util.List;

@RestController
@RequestMapping("/api/actions")
public class ActionsListController {

    private final ActionsListService actionsListService;

    @Autowired
    public ActionsListController(ActionsListService actionsListService) {
        this.actionsListService = actionsListService;
    }

    // Создание нового действия
    @Operation(summary = "Create a new action")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "201", description = "Action created"),
        @ApiResponse(responseCode = "400", description = "Invalid input")
    })
    @PostMapping("/create")
    public ResponseEntity<ActionsList> createAction(@RequestBody ActionsList
action) {
        ActionsList createAction = actionsListService.saveAction(action);
        return new ResponseEntity<>(createAction, HttpStatus.CREATED);
    }

    // Получение действия по его ID
    @Operation(summary = "Get action by ID")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Found the
action"),
        @ApiResponse(responseCode = "404", description = "Action not found")
    })
    @GetMapping("/{id}")
    public ResponseEntity<ActionsList> getActionById(@PathVariable("id") int id)
{
        ActionsList action = actionsListService.getActionById(id);
        if (action != null) {
            return new ResponseEntity<>(action, HttpStatus.OK);
        } else {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    }

    // Получение всех действий
    @Operation(summary = "Get all actions")

```

```
@GetMapping("/all")
public ResponseEntity<List<ActionsList>> getAllActions() {
    List<ActionsList> actions = actionsListService.getAllActions();
    return new ResponseEntity<>(actions, HttpStatus.OK);
}

// Удаление действия по его ID
@Operation(summary = "Delete action by ID")
@ApiResponses(value = {
    @ApiResponse(responseCode = "204", description = "Action deleted"),
    @ApiResponse(responseCode = "404", description = "Action not found")
})
@DeleteMapping("/{id}")
public ResponseEntity<Void> deleteAction(@PathVariable("id") int id) {
    actionsListService.deleteAction(id);
    return new ResponseEntity<>(HttpStatus.NO_CONTENT);
}
}

//POST /api/actions/create: Создание нового действия.
//GET /api/actions/{id}: Получение действия по его ID.
//GET /api/actions/all: Получение всех действий.
//DELETE /api/actions/{id}: Удаление действия по его ID.
```

8. ProtocolController

```

package ru.surgu.medexambackend.controller;

import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.responses.ApiResponses;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import ru.surgu.medexambackend.entity.*;
import ru.surgu.medexambackend.service.ProtocolService;

import java.util.List;

@RestController
@RequestMapping("/protocols")
public class ProtocolController {

    private final ProtocolService protocolService;

    @Autowired
    public ProtocolController(ProtocolService protocolService) {
        this.protocolService = protocolService;
    }

    // Эндпоинт для сохранения протокола
    @Operation(summary = "Save protocol")
    @PostMapping
    public ResponseEntity<Protocol> saveProtocol(@RequestBody Protocol protocol)
{
        Protocol savedProtocol = protocolService.saveProtocol(protocol);
        return new ResponseEntity<>(savedProtocol, HttpStatus.CREATED);
    }

    // Эндпоинт для поиска протокола по его ID
    @Operation(summary = "Find protocol by ID")
    @GetMapping("/{id}")
    public ResponseEntity<Protocol> findProtocolById(@PathVariable int id) {
        Protocol protocol = protocolService.findProtocolById(id);
        return protocol != null ? ResponseEntity.ok(protocol) :
ResponseEntity.notFound().build();
    }

    // Эндпоинт для поиска паспорта станции по ID протокола
    @Operation(summary = "Find station passport by protocol ID")
    @GetMapping("/{protocolId}/station-passport")
    public ResponseEntity<StationPassport>
findStationPassportByProtocolId(@PathVariable int protocolId) {
        StationPassport stationPassport =
protocolService.findStationPassportByProtocolId(protocolId);
        return stationPassport != null ? ResponseEntity.ok(stationPassport) :
ResponseEntity.notFound().build();
    }

    // Эндпоинт для поиска сценариев по ID протокола и ID паспорта станции
    @Operation(summary = "Find scenarios by protocol ID and station passport
ID")
    @GetMapping("/{protocolId}/scenarios/{stationPassportId}")

```

```
        public ResponseEntity<List<SituationsList>>
findScenariosByProtocolIdAndStationPassportId(@PathVariable int protocolId,
@PathVariable int stationPassportId) {
    List<SituationsList> scenarios =
protocolService.findScenariosByProtocolIdAndStationPassportId(protocolId,
stationPassportId);
    return ResponseEntity.ok(scenarios);
}

// Эндпоинт для поиска действий по ID протокола, ID паспорта станции и ID
сценария
@Operation(summary = "Find actions by protocol ID, station passport ID, and
scenario ID")
@GetMapping("/{protocolId}/actions/{stationPassportId}/{scenarioId}")
public ResponseEntity<List<ActionsList>>
findActionsByProtocolAndStationPassportAndScenario(@PathVariable int protocolId,
@PathVariable int stationPassportId, @PathVariable int scenarioId) {
    List<ActionsList> actions =
protocolService.findActionsByProtocolAndStationPassportAndScenario(protocolId,
stationPassportId, scenarioId);
    return ResponseEntity.ok(actions);
}

// Эндпоинт для поиска результатов выполнения действий по ID протокола и
списку действий
@Operation(summary = "Find results by actions and protocol ID")
@PostMapping("/{protocolId}/results")
public ResponseEntity<List<ExamInfo>>
findResultsByActionsAndProtocol(@PathVariable int protocolId, @RequestBody
List<ActionsList> actionsList) {
    List<ExamInfo> results =
protocolService.findResultsByActionsAndProtocol(actionsList, protocolId);
    return ResponseEntity.ok(results);
}
}

//Создание протокола:
//Метод: POST
//Путь: /protocols

//Получение протокола по его ID:
//Метод: GET
//Путь: /protocols/{id}

//Получение паспорта станции по ID протокола:
//Метод: GET
//Путь: /protocols/{protocolId}/station-passport

//Получение сценариев по ID протокола и ID паспорта станции:
//Метод: GET
//Путь: /protocols/{protocolId}/scenarios/{stationPassportId}

//Получение действий по ID протокола, ID паспорта станции и ID сценария:
//Метод: GET
//Путь: /protocols/{protocolId}/actions/{stationPassportId}/{scenarioId}

//Получение результатов выполнения действий по ID протокола и списку действий:
//Метод: POST
//Путь: /protocols/{protocolId}/results
```

9. ExamInfoController

```

package ru.surgu.medexambbackend.controller;

import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.Parameter;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.responses.ApiResponses;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import ru.surgu.medexambbackend.entity.ExamInfo;
import ru.surgu.medexambbackend.service.ExamInfoService;

import java.util.List;

@RestController
@RequestMapping("/exam-info")
public class ExamInfoController {

    private final ExamInfoService examInfoService;

    @Autowired
    public ExamInfoController(ExamInfoService examInfoService) {
        this.examInfoService = examInfoService;
    }

    // Эндпоинт для сохранения информации о экзамене
    @Operation(summary = "Save exam information")
    @PostMapping("/save")
    public ResponseEntity<ExamInfo> saveExamInfo(@RequestBody ExamInfo examInfo)
{
        ExamInfo savedExamInfo = examInfoService.saveExamInfo(examInfo);
        return ResponseEntity.ok(savedExamInfo);
    }

    // Эндпоинт для получения информации о экзамене по его ID
    @Operation(summary = "Get exam information by ID")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Found the exam
information", content = @Content(schema = @Schema(implementation =
ExamInfo.class))),
        @ApiResponse(responseCode = "404", description = "Exam information
not found")
    })
    @GetMapping("/{id}")
    public ResponseEntity<ExamInfo> getExamInfoById(@PathVariable int id) {
        ExamInfo examInfo = examInfoService.getExamInfoById(id);
        if (examInfo != null) {
            return ResponseEntity.ok(examInfo);
        } else {
            return ResponseEntity.notFound().build();
        }
    }

    // Эндпоинт для получения списка всех записей о экзамене
    @Operation(summary = "Get all exam information")
    @GetMapping("/all")
    public ResponseEntity<List<ExamInfo>> getAllExamInfos() {

```

```
List<ExamInfo> examInfos = examInfoService.getAllExamInfos();
return ResponseEntity.ok(examInfos);
}

// Эндпоинт для удаления информации о экзамене по его ID
@Operation(summary = "Delete exam information by ID")
@DeleteMapping("/{id}")
public ResponseEntity<Void> deleteExamInfo(@PathVariable int id) {
    examInfoService.deleteExamInfo(id);
    return ResponseEntity.noContent().build();
}
}

//Сохранение информации о экзамене:
//Метод: POST
//Путь: /exam-info/save
//Принимаемые данные: объект ExamInfo в теле запроса
//Ответ: сохраненный объект ExamInfo

//Получение информации о экзамене по его ID:
//Метод: GET
//Путь: /exam-info/{id}
//Параметры: id - идентификатор экзамена
//Ответ: информация о экзамене с указанным id

//Получение списка всех записей о экзаменах:
//Метод: GET
//Путь: /exam-info/all
//Ответ: список всех записей о экзаменах

//Удаление информации о экзамене по его ID:
//Метод: DELETE
//Путь: /exam-info/{id}
//Параметры: id - идентификатор экзамена
//Ответ: статус ответа без содержимого (204 No Content)
```

10. **ActionScenarioController**

```

package ru.surgu.medexambackend.controller;

import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.responses.ApiResponses;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import ru.surgu.medexambackend.entity.ActionScenario;
import ru.surgu.medexambackend.service.ActionScenarioService;

import java.util.List;

@RestController
@RequestMapping("/api/action-scenarios")
public class ActionScenarioController {

    private final ActionScenarioService actionScenarioService;

    @Autowired
    public ActionScenarioController(ActionScenarioService actionScenarioService)
    {
        this.actionScenarioService = actionScenarioService;
    }

    // Создание нового экземпляра ActionScenario
    @Operation(summary = "Create a new ActionScenario")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "201", description = "ActionScenario
created"),
        @ApiResponse(responseCode = "400", description = "Invalid
ActionScenario request")
    })
    @PostMapping
    public ResponseEntity<ActionScenario> createActionScenario(@RequestBody
ActionScenario actionScenario) {
        ActionScenario createdActionScenario =
actionScenarioService.saveActionScenario(actionScenario);
        return
ResponseEntity.status(HttpStatus.CREATED).body(createdActionScenario);
    }

    // Получение экземпляра ActionScenario по его ID
    @Operation(summary = "Get an ActionScenario by ID")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Found the
ActionScenario"),
        @ApiResponse(responseCode = "404", description = "ActionScenario not
found")
    })
    @GetMapping("/{id}")
    public ResponseEntity<ActionScenario> getActionScenarioById(@PathVariable
int id) {
        ActionScenario actionScenario =
actionScenarioService.getActionScenarioById(id);
        if (actionScenario != null) {
            return ResponseEntity.ok(actionScenario);
        } else {
            return ResponseEntity.notFound().build();
        }
    }
}

```



```

    }
}

// Получение списка всех экземпляров ActionScenario
@Operation(summary = "Get all ActionScenarios")
@GetMapping
public ResponseEntity<List<ActionScenario>> getAllActionScenarios() {
    List<ActionScenario> actionScenarios =
actionScenarioService.getAllActionScenarios();
    return ResponseEntity.ok(actionScenarios);
}

// Удаление экземпляра ActionScenario по его ID
@Operation(summary = "Delete an ActionScenario by ID")
@ApiResponses(value = {
    @ApiResponse(responseCode = "204", description = "ActionScenario
deleted"),
    @ApiResponse(responseCode = "404", description = "ActionScenario not
found")
})
@DeleteMapping("/{id}")
public ResponseEntity<Void> deleteActionScenario(@PathVariable int id) {
    actionScenarioService.deleteActionScenario(id);
    return ResponseEntity.noContent().build();
}
}

//POST /api/action-scenarios: Создает новый экземпляр ActionScenario.
//GET /api/action-scenarios/{id}: Получает экземпляр ActionScenario по его ID.
//GET /api/action-scenarios: Получает список всех экземпляров ActionScenario.
//DELETE /api/action-scenarios/{id}: Удаляет экземпляр ActionScenario по его ID.

```

11. **ExamController**

```

package ru.surgu.medexambackend.controller;

import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.responses.ApiResponses;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import ru.surgu.medexambackend.entity.Exam;
import ru.surgu.medexambackend.entity.Protocol;
import ru.surgu.medexambackend.service.ExamService;

import java.util.List;

@RestController
@RequestMapping("/api/exams") // Базовый URL для всех запросов к контроллеру
public class ExamController {

    private final ExamService examService;

    @Autowired
    public ExamController(ExamService examService) {
        this.examService = examService;
    }

    // Эндпоинт для сохранения экзамена
    @Operation(summary = "Сохранить экзамен")
    @PostMapping("/save")
    public ResponseEntity<Exam> saveExam(@RequestBody Exam exam) {
        Exam savedExam = examService.saveExam(exam);
        return ResponseEntity.ok(savedExam);
    }

    // Эндпоинт для получения экзамена по его ID
    @Operation(summary = "Получить экзамен по его ID")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Экзамен найден"),
        @ApiResponse(responseCode = "404", description = "Экзамен не
найдено")
    })
    @GetMapping("/{id}")
    public ResponseEntity<Exam> getExamById(@PathVariable int id) {
        Exam exam = examService.getExamById(id);
        if (exam != null) {
            return ResponseEntity.ok(exam);
        } else {
            return ResponseEntity.notFound().build();
        }
    }

    // Эндпоинт для получения всех экзаменов
    @Operation(summary = "Получить все экзамены")
    @GetMapping("/all")
    public ResponseEntity<List<Exam>> getAllExams() {
        List<Exam> exams = examService.getAllExams();
        return ResponseEntity.ok(exams);
    }

    // Эндпоинт для удаления экзамена по его ID
    @Operation(summary = "Удалить экзамен по его ID")

```

```
@DeleteMapping("/{id}")
public ResponseEntity<Void> deleteExam(@PathVariable int id) {
    examService.deleteExam(id);
    return ResponseEntity.noContent().build();
}

// Эндпоинт для получения протоколов для заданного экзамена
@Operation(summary = "Получить протоколы для заданного экзамена")
@GetMapping("/{examId}/protocols")
public ResponseEntity<List<Protocol>> getProtocolsByExamId(@PathVariable int
examId) {
    List<Protocol> protocols = examService.findProtocolsByExamId(examId);
    return ResponseEntity.ok(protocols);
}

// Эндпоинт для получения результатов для заданного экзамена
@Operation(summary = "Получить результаты для заданного экзамена")
@GetMapping("/{examId}/results")
public ResponseEntity<List<Object[]>> getResultsByExamId(@PathVariable int
examId) {
    List<Object[]> results = examService.findResultsByExamId(examId);
    return ResponseEntity.ok(results);
}
}

//POST /api/exams/save: Этот эндпоинт используется для сохранения нового
экзамена.
// Он принимает экземпляр экзамена в формате JSON в теле запроса и возвращает
сохраненный экзамен.

//GET /api/exams/{id}: Этот эндпоинт позволяет получить экзамен по его
уникальному идентификатору (ID).
// Он принимает ID экзамена в качестве переменной пути и возвращает экзамен в
формате JSON.

//GET /api/exams/all: Этот эндпоинт возвращает список всех экзаменов.
// Он не принимает параметров и возвращает список всех доступных экзаменов в
формате JSON.

//DELETE /api/exams/{id}: Этот эндпоинт используется для удаления экзамена по
его ID.
// Он принимает ID экзамена в качестве переменной пути и не возвращает никаких
данных.

//GET /api/exams/{examId}/protocols: Этот эндпоинт возвращает список протоколов,
связанных с определенным экзаменом.
// Он принимает ID экзамена в качестве переменной пути и возвращает список
протоколов в формате JSON.

//GET /api/exams/{examId}/results: Этот эндпоинт возвращает результаты,
связанные с определенным экзаменом.
// Он принимает ID экзамена в качестве переменной пути и возвращает список
результатов в формате JSON.
```

12. **AuthenticationController**

```

package ru.surgu.medexambackend.controller;

import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.responses.ApiResponses;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import ru.surgu.medexambackend.entity.Authentication;
import ru.surgu.medexambackend.service.AuthenticationService;

import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/api/authentication")
public class AuthenticationController {

    private final AuthenticationService authenticationService;

    @Autowired
    public AuthenticationController(AuthenticationService authenticationService)
    {
        this.authenticationService = authenticationService;
    }

    // Создание новой сущности Authentication
    @Operation(summary = "Create a new Authentication")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "201", description = "Authentication
created"),
        @ApiResponse(responseCode = "400", description = "Invalid request
body")
    })
    @PostMapping("/create")
    public ResponseEntity<Authentication> createAuthentication(@RequestBody
Authentication authentication) {
        Authentication createdAuthentication =
authenticationService.saveAuthentication(authentication);
        return
ResponseEntity.status(HttpStatus.CREATED).body(createdAuthentication);
    }

    // Получение сущности Authentication по ее ID
    @Operation(summary = "Get Authentication by ID")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Found the
Authentication"),
        @ApiResponse(responseCode = "404", description = "Authentication not
found")
    })
    @GetMapping("/{id}")
    public ResponseEntity<Authentication> getAuthenticationById(@PathVariable
int id) {
        Optional<Authentication> authentication =
authenticationService.getAuthenticationById(id);

```

```

        return authentication.map(ResponseEntity::ok).orElseGet(() ->
ResponseEntity.notFound().build());
    }

    // Получение списка всех сущностей Authentication
    @Operation(summary = "Get all Authentications")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Found all
Authentications"),
        @ApiResponse(responseCode = "204", description = "No Authentications
found")
    })
    @GetMapping("/all")
    public ResponseEntity<List<Authentication>> getAllAuthentications() {
        List<Authentication> authentications =
authenticationService.getAllAuthentications();
        if (authentications.isEmpty()) {
            return ResponseEntity.noContent().build();
        } else {
            return ResponseEntity.ok(authentications);
        }
    }

    // Удаление сущности Authentication по ее ID
    @Operation(summary = "Delete Authentication by ID")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "204", description = "Authentication
deleted"),
        @ApiResponse(responseCode = "404", description = "Authentication not
found")
    })
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteAuthentication(@PathVariable int id) {
        authenticationService.deleteAuthentication(id);
        return ResponseEntity.noContent().build();
    }
}

//POST /api/authentication/create: Создание новой сущности Authentication.
//GET /api/authentication/{id}: Получение сущности Authentication по ее ID.
//GET /api/authentication/all: Получение списка всех сущностей Authentication.
//DELETE /api/authentication/{id}: Удаление сущности Authentication по ее ID.

```