Министерство науки и высшего образования Российской Федерации

**Муромский институт (филиал)**

федерального государственного бюджетного образовательного учреждения высшего образования

**«Владимирский государственный университет   
имени Александра Григорьевича и Николая Григорьевича Столетовых»**

**(МИ ВлГУ)**

Факультет ИТР

Кафедра ИС

УТВЕРЖДАЮ

Заведующий кафедрой

Д.Е. Андрианов

(подпись)

«\_\_\_\_\_\_» 2025 г.

БАКАЛАВРСКАЯ

РАБОТА

Тема Разработка информационной системы для оценки знаний студентов при выполнении практических работ по информатике

МИВУ.09.03.02-00.000 БР

Руководитель

(фамилия, инициалы)

(подпись) (дата)

Студент

(группа)

(фамилия, инициалы)

(подпись) (дата)

Муром 2025

БЛАНК ЗАДАНИЯ

Бакалаврская работа посвящена разработке информационной системы для оценки знаний студентов при выполнении практических работ по информатике. В проекте используется сочетание технологий JavaFX для создания пользовательского интерфейса и Spring Boot для реализации серверной части. Основное внимание уделяется автоматизации процесса контроля и анализа результатов практических заданий, что позволяет повысить эффективность обучения и объективность оценки знаний. Разработанная система ориентирована на образовательные учреждения и преподавателей, обеспечивая удобный и интерактивный инструмент для проведения практических занятий и мониторинга успеваемости студентов.

Табл. 2. Ил. 19. Библ. 18

The bachelor’s thesis is dedicated to the development of an information system for assessing students’ knowledge during the completion of practical computer science assignments. The project employs a combination of JavaFX for the user interface and Spring Boot for backend implementation. The primary focus is on automating the process of monitoring and analyzing practical task results, which enhances learning efficiency and objectivity in knowledge assessment. The developed system is aimed at educational institutions and instructors, providing a convenient and interactive tool for conducting practical lessons and tracking student performance.

Tabl. 2. Fig. 19. Bibl. 18

Министерство науки и высшего образования Российской Федерации

**Муромский институт (филиал)**

федерального государственного бюджетного образовательного учреждения высшего образования

**«Владимирский государственный университет   
имени Александра Григорьевича и Николая Григорьевича Столетовых»**

**(МИ ВлГУ)**

Факультет ИТР

Кафедра ИС

ПОЯСНИТЕЛЬНАЯ

ЗАПИСКА

Тема: Разработка информационной системы оценки знаний

студентов при выполнении практических работ по информатике

МИВУ.09.03.02-00.000 ПЗ

Муром 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 7

1 Анализ технического задания 9

1.1 Общие сведения об объекте автоматизации 9

1.2 Обзор существующих аналогов 9

1.3 Формирование требований к информационной системе 11

1.4 Выбор средств реализации поставленной задачи 12

2 Проектирование системы 15

2.1 Проектирование баз данных 15

2.2 Разработка диаграмм информационной системы 19

3 Реализация информационной системы 35

3.1 Структура интерфейса системы 35

3.2 Программная архитектура системы 43

3.3 Реализация доступа к базе данных 46

3.4 Реализация моделей данных 48

4 Руководство по программному продукту 53

4.1 Руководство администратора 53

4.2 Руководство программиста 58

5 Тестирование 65

ЗАКЛЮЧЕНИЕ 69

# ВВЕДЕНИЕ

Современное образование стремительно развивается благодаря активному внедрению информационных технологий. Это особенно заметно в сфере подготовки специалистов, где важную роль играет качество и оперативность оценки знаний студентов. Практические работы по информатике являются одним из ключевых элементов учебного процесса, позволяя студентам закреплять теоретические знания на практике и развивать навыки программирования и решения задач.

Однако традиционные методы контроля знаний при выполнении практических заданий имеют ряд недостатков. Проверка часто требует значительных временных затрат у преподавателей и может содержать элемент субъективности, что влияет на объективность оценки. Кроме того, при большом количестве студентов становится трудно оперативно обработать и проанализировать результаты, что снижает эффективность учебного процесса.

В связи с этим актуальной задачей является создание информационных систем, которые автоматизируют процесс контроля знаний, обеспечивают прозрачность оценки и упрощают взаимодействие между преподавателем и студентами. Такие системы не только фиксируют выполнение заданий, но и способны автоматически или полуавтоматически проверять правильность решений, а также формировать статистику по результатам обучения.

В рамках данной работы рассматривается разработка информационной системы для оценки знаний студентов при выполнении практических работ по информатике. Использование современных технологий, таких как JavaFX для клиентской части и Spring Boot с применением gRPC для серверной, позволяет создать надёжное, удобное и масштабируемое приложение. Хранение данных организовано с помощью СУБД RedDatabase 3.0, что обеспечивает эффективное управление информацией.

Основная цель проекта — разработка клиент-серверной системы, которая обеспечит удобное взаимодействие между преподавателем и студентами, автоматизирует сбор и анализ результатов практических заданий, а также повысит качество и объективность оценки знаний.

Для достижения этой цели в работе решаются следующие задачи:

* проведение анализа существующих программных решений в области автоматизированной оценки знаний и выявление их преимуществ и недостатков;
* формулирование функциональных и нефункциональных требований к разрабатываемой системе с учётом специфики образовательного процесса;
* проектирование архитектуры клиент-серверного приложения, обеспечивающей надёжность и расширяемость;
* разработка клиентской части на базе JavaFX, обеспечивающей удобный и интуитивно понятный интерфейс для пользователей;
* создание серверной части с использованием Spring Boot и gRPC для эффективного взаимодействия компонентов системы;
* организация хранения и обработки данных в СУБД RedDatabase 3.0, учитывающей особенности работы с учебной информацией;
* проведение тестирования и оценки работоспособности разработанного программного продукта.

Объектом исследования является процесс оценки знаний студентов в условиях практико-ориентированного обучения информатике, что предполагает активное применение информационных технологий для повышения качества образования.

Предметом исследования выступает проектирование и реализация информационной системы, обеспечивающей автоматизацию оценки результатов выполнения студентами практических заданий с применением современных технологий JavaFX, Spring Boot, gRPC и СУБД RedDatabase 3.0.

# 1 Анализ технического задания

# Общие сведения об объекте автоматизации

Объектом автоматизации является процесс оценки знаний студентов при выполнении практических работ по информатике. В традиционной образовательной среде этот процесс включает формулирование преподавателем заданий, их раздачу студентам, сбор выполненных работ, ручную проверку и выставление итоговых оценок. Автоматизируемая система берёт на себя функции авторизации пользователей, индивидуальной генерации заданий на основе mn-кода, приёма и хранения результатов, их автоматической валидации по заранее заданным критериям. Дополнительно система формирует отчётные файлы и отправляет их на электронную почту для архивации и контроля. Внедрение такого решения обеспечит повышение объективности и оперативности оценивания, снижение нагрузки на преподавателя и исключит возможность списывания благодаря уникальным вариантам практических заданий.

* 1. Обзор существующих аналогов

При проектировании информационной системы важно учитывать уже существующие решения, которые выполняют схожие функции. Анализ аналогов позволяет выявить их сильные и слабые стороны, что поможет избежать ошибок при разработке собственной системы. В рамках исследования были рассмотрены несколько популярных образовательных платформ и систем автоматизированной проверки знаний.

Одним из наиболее известных решений является Moodle, который широко используется в образовательных учреждениях. Он предоставляет инструменты для тестирования, проверки заданий и управления учебными материалами, однако сложность настройки и адаптации под конкретные задачи делает его не всегда удобным решением.

Stepik предлагает автоматизированную проверку программного кода, но ориентирован больше на онлайн-курсы, чем на офлайн-оценку практических работ.

Codeforces и LeetCode позволяют проверять код студентов в автоматическом режиме, однако их основная направленность – алгоритмические задачи, что не всегда применимо для практических работ по информатике.

Яндекс.Контест активно используется для автоматизированного тестирования студенческих работ, но его основной фокус – на спортивное программирование и олимпиадные задачи.

Codility применяется в основном для технических интервью и проверки алгоритмических навыков, что делает его менее подходящим для образовательного процесса.

Таблица 1- Сравнительный анализ программ аналогов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Программа | Автоматическая проверка кода | Генерация заданий | Интеграция с LMS | Защита от списывания | Поддержка офлайн-заданий |
| Moodle | Да | Частично | Да | Нет | Да |
| Stepik | Да | Нет | Ограниченная | Нет | Нет |
| Codeforces | Да | Нет | Нет | Частично | Нет |
| LeetCode | Да | Нет | Нет | Нет | Нет |
| Яндекс.Контест | Да | Нет | Нет | Частично | Нет |
| Codility | Да | Нет | Нет | Частично | Нет |

Большинство существующих решений либо не ориентированы на проверку практических лабораторных работ, либо требуют значительных доработок и адаптации. Это подтверждает необходимость создания специализированной системы, которая будет включать автоматизированную генерацию и проверку уникальных заданий

* 1. Формирование требований к информационной системе

Разрабатываемая информационная система предназначена для автоматизации процесса оценки знаний студентов при выполнении практических заданий по информатике. Основное внимание уделяется взаимодействию студента с системой: от авторизации до получения обратной связи по выполненной работе.

Каждый студент проходит авторизацию в системе, после чего получает доступ к практическим заданиям. Особенностью реализации является индивидуальная генерация заданий — каждое формируется уникально на основе специального идентификатора (mn-кода) студента. Это исключает возможность списывания и обеспечивает высокий уровень контроля.

После получения задания студент приступает к его выполнению непосредственно в пользовательском интерфейсе приложения. По завершении практической работы предусмотрена возможность загрузки результатов прямо через интерфейс клиента. Загрузка включает как данные выполнения, так и автоматическую отправку отчётного файла на электронную почту, что создаёт дополнительный уровень контроля и архивации.

Проверка заданий осуществляется автоматически. В отличие от традиционных систем, участие преподавателя в процессе проверки не требуется: система самостоятельно сопоставляет результат выполнения с заранее заданными критериями. Такой подход позволяет снизить нагрузку на преподавателя и обеспечить объективность.

Что касается нефункциональных требований, система должна обеспечивать удобный и интуитивно понятный графический интерфейс, реализованный средствами JavaFX, что делает её кросс-платформенной и доступной для использования как в аудиториях, так и на личных устройствах студентов. Высокая производительность системы критически важна в условиях массового одновременного использования, особенно при выполнении контрольных и зачётных работ. Обмен данными между клиентом и сервером осуществляется по протоколу gRPC, что обеспечивает надёжную, защищённую и эффективную передачу информации. Безопасность системы является приоритетом —идентификационные данные должны передаваться в зашифрованном виде. Кроме того, архитектура сервера на основе Spring Boot должна позволять гибко масштабировать решение, адаптируя его под увеличивающееся количество пользователей и расширение функциональности без полного рефакторинга кода.

* 1. Выбор средств реализации поставленной задачи

Выбор программных средств для реализации информационной системы основывался на необходимости создать надёжное, производительное и легко сопровождаемое решение, пригодное для работы в образовательной среде. При проектировании учитывались требования к модульности архитектуры, удобству пользовательского интерфейса, скорости отклика, а также доступности и совместимости инструментов разработки.

В качестве основного языка программирования была выбрана Java — зрелая и универсальная платформа, обеспечивающая стабильную работу на различных операционных системах и имеющая широкую экосистему библиотек и фреймворков. Это позволило использовать единый стек технологий как на клиентской, так и на серверной стороне системы.

Для реализации клиентского интерфейса использована JavaFX — библиотека для создания графических пользовательских интерфейсов. Она обеспечивает современный внешний вид, кросс-платформенность, возможность быстрой разработки и гибкость кастомизации интерфейсов. Это особенно важно в образовательных проектах, где требуется ясный и интуитивно понятный пользовательский опыт.

Серверная часть системы реализована с применением Spring Boot — одного из самых распространённых Java-фреймворков для создания микросервисов. Он обеспечивает высокую надёжность, удобную структуру проекта, встроенные средства безопасности и простоту настройки, что делает его оптимальным выбором для образовательных информационных систем.

В качестве механизма взаимодействия между клиентом и сервером выбран gRPC — современный протокол удалённого вызова процедур, работающий поверх HTTP/2. Использование gRPC позволяет добиться высокой скорости и компактности передачи данных за счёт сериализации с помощью Protocol Buffers, а также реализовать потоковую двустороннюю передачу сообщений, необходимую для интерактивного взаимодействия в реальном времени.

Сборка и управление зависимостями проекта осуществляется с помощью Gradle — гибкой системы сборки, которая позволяет точно управлять конфигурацией проекта и автоматически подтягивать необходимые библиотеки. Gradle обеспечивает быструю сборку, поддержку модульных проектов и простоту интеграции с современными инструментами разработки.

Разработка велась в среде IntelliJ IDEA — одной из наиболее мощных интегрированных сред разработки для Java. Эта среда предоставляет средства рефакторинга, отладки, профилирования и интеграции с системами контроля версий, что позволяет существенно ускорить процесс программирования и обеспечить высокое качество кода.

Для хранения и обработки данных выбрана RedDatabase 3.0 — отечественная СУБД, оптимизированная для работы с большими объёмами информации и совместимая с Firebird. Доступ к базе данных осуществляется через мультиплатформенный менеджер СУБД — Ред Эксперт, который поддерживает подключение как к RedDatabase, так и к Firebird. Преимуществом решения является полная поддержка всех версий Red Базы Данных и Firebird "из коробки", что обеспечивает гибкость при развёртывании системы на различных конфигурациях серверов. Это также облегчает сопровождение и перенос системы в случае необходимости смены версии или среды.

Интеграция с RedDatabase и Firebird осуществляется через стандартный JDBC-интерфейс, что обеспечивает высокую совместимость с Java-приложениями и позволяет использовать всю мощь SQL для работы с данными: от простых выборок до сложных триггеров и хранимых процедур.

Таким образом, совокупность выбранных программных средств обеспечивает надёжную архитектуру системы, совместимую с различными платформами, легко расширяемую и готовую к эксплуатации в условиях учебного процесса. Использование современных технологий, таких как gRPC и Spring Boot, в сочетании с проверенными инструментами управления и разработки, позволяет обеспечить высокую производительность, безопасность и масштабируемость решения.

# 2 Проектирование системы

# 2.1 Проектирование баз данных

Общая структура базы данных построена на основе объектно-реляционного отображения (ORM) с использованием JPA моделей, что обеспечивает удобное взаимодействие между объектной моделью приложения и реляционной моделью базы данных. В центре модели находится таблица студентов (STUDENTS), которая содержит основную информацию о пользователях — студентам, участвующих в учебном процессе. Каждая запись этой таблицы включает индивидуальные данные, такие как фамилия, имя, отчество, уникальный код студента в группе, а также ссылку на группу, к которой он принадлежит. Группы студентов представлены отдельной таблицей GROUPS, которая содержит идентификаторы и наименования групп, позволяя структурировать студентов по учебным подразделениям. Помимо этого, в системе предусмотрены сущности, отражающие учебные практики (PRACTICES), которые объединяют множество учебных заданий (TASKS), хранящихся в таблице TASKS. Каждое задание связано с конкретной практикой и может быть использовано для оценки знаний студентов. Ответы студентов на эти задания фиксируются в таблице STUDENT\_ANSWERS, где каждая запись ссылается на конкретного студента и конкретное задание, что позволяет детально отслеживать и анализировать процесс обучения. Для управления ролями пользователей в системе существует отдельная таблица ROLES. Кроме того, профили пользователей дополнены сущностью USER\_AVATAR, которая хранит данные об их аватарах, а также отдельной таблицей PROFILE с расширенной информацией о пользователях.

При проектировании базы данных была проведена детальная декомпозиция предметной области — выделены сущности пользователей, студентов, учебных групп, типов и запусков практик, а также заданий, примеров и ответов студентов. Каждая сущность оформлена в отдельную таблицу со строго определённым первичным ключом типа BIGINT и набором атрибутов, обеспечивающих полноту и однозначность данных. Связи между таблицами, которые наглядно отражены на рисунке 1, реализованы через внешние ключи и уникальные ограничения: таблица STUDENTS ссылается на USERS и GROUPS, таблица PRACTICES — на PRACTICE\_TYPES и конкретного студента по mn-коду, а таблица STUDENT\_ANSWERS одновременно ссылается на текущую практику, определение задания и пример. Это позволяет обеспечить целостность и непротиворечивость данных при выполнении операций добавления, изменения и удаления.

Связи между сущностями реализованы с учетом логики предметной области и требований к целостности данных. Основной связью является отношение многие-к-одному между студентом и группой — каждый студент принадлежит ровно одной группе, а группа может содержать множество студентов. В таблице STUDENTS для этого используется внешний ключ GROUP\_ID, который ссылается на первичный ключ в таблице GROUPS. Аналогично, каждая практика относится к одной группе, что позволяет разграничивать учебные материалы по группам и обеспечивать соответствие заданий целевой аудитории. Задания (TASKS) связаны с практиками отношением один-ко-многим, где в таблице TASKS хранится внешний ключ PRACTICE\_ID, указывающий на соответствующую практику. Такая структура облегчает навигацию от практики к набору её заданий. Отношения между студентами и их ответами на задания реализованы через связь один-ко-многим: студент может дать множество ответов, каждый из которых привязан к определённой задаче. В таблице STUDENT\_ANSWERS хранятся ссылки как на студента (STUDENT\_ID), так и на задачу (TASK\_ID), что обеспечивает возможность построения подробных отчетов об успеваемости. Для пользователей и ролей предусмотрена связь многие-ко-многим через ROLES, что обеспечивает гибкость в распределении прав и доступов в системе. Связь один-к-одному реализована между пользователем и его аватаром, позволяя хранить изображения, ассоциированные с каждым конкретным пользователем.

Все атрибуты в базе данных нормализованы до третьей нормальной формы, что исключает избыточность и обеспечивает однозначность данных. Параметры генератора заданий и дополнительные метаданные хранятся в отдельной JSON-колонке PARAMS\_JSON в справочнике PRACTICE\_TYPE\_TASK\_DEFS, что позволяет гибко добавлять новые типы вопросов и параметры без необходимости вносить изменения в структуру таблиц. Таблица TASK\_EXAMPLES обеспечивает хранение вариантов входных данных для каждого задания, предоставляя возможность многообразия и вариативности упражнений. Важным элементом является наличие статусов сущностей (таблица STATUSES) как в таблице TASKS, так и в PRACTICES. Статус, который закрепляется за определенной практикой или заданием и позволяет быстро определять на каком этапе находится сущность и какие действия с ней можно выполнять:

* STATUSID = 1 (Новый): Практическая работа или задание еще не были пройдены студентом;
* STATUSID = 3 (Не пройдена проверка): Практическая работа или задание студентом выполнены не правильно;
* STATUSID = 13 (Завершенный): Практическая работа или задание успешно завершены студентом (все задания должны быть верными);
* STATUSID = 14 (Возможна ошибка): Практическая работа или задание могут иметь в себе ошибку, не связанную с ответом студента.

Особое внимание уделялось индексации и созданию уникальных ограничений для обеспечения эффективности и целостности данных. В таблице STUDENTS проиндексировано поле FULL\_NAME, что существенно ускоряет поиск и фильтрацию студентов по полному имени. Для уникальной идентификации студента в пределах группы введено ограничение уникальности на пару столбцов GROUP\_ID и MN\_CODE, что исключает дублирование кодов и облегчает внутренние операции идентификации. В таблице GROUPS индексировано поле NAME, что оптимизирует операции выборки и сортировки по наименованию групп. В таблице STUDENT\_ANSWERS созданы индексы по полям STUDENT\_ID и TASK\_ID, что позволяет быстро получать все ответы конкретного студента и ускоряет обработку статистических данных и отчетов.

Для автоматизации процессов в таблице STUDENTS реализованы два критически важных триггера, обеспечивающих поддержку согласованности данных и облегчение работы с ними. Первый триггер TRG\_STUDENTS\_FULL\_NAME автоматически формирует поле FULL\_NAME, объединяя компоненты фамилии, имени и отчества в единое строковое представление при вставке или обновлении записи. Это уменьшает вероятность ошибок ввода и обеспечивает единообразие данных без необходимости дополнительной обработки на уровне приложения. Второй триггер TRG\_STUDENTS\_MN\_CODE автоматически генерирует уникальный двузначный код MN\_CODE для студента в рамках одной группы, если код не был указан вручную. Триггер реализует цикл генерации двух случайных цифр от 1 до 9, исключая совпадение цифр между собой, с последующей проверкой уникальности полученного кода в группе. Если код уже существует, генерация повторяется до получения уникального значения. Данный механизм освобождает разработчиков и пользователей от необходимости ручного контроля уникальности MN\_CODE, одновременно повышая надёжность данных.

При проектировании базы данных также учитывались требования по производительности и надёжности хранения данных. Выбранная система управления базами данных — RedDatabase 3.0 с совместимостью Firebird — предоставляет механизмы транзакционной целостности, что гарантирует корректность выполнения операций в условиях конкурентного доступа. Кроме того, СУБД поддерживает масштабируемость и встроенные инструменты репликации, доступные через платформу «Ред Эксперт», что обеспечивает возможность расширения и устойчивость к отказам. Доступ к базе организован через единый JDBC-слой, что обеспечивает кроссплатформенную совместимость приложения и упрощает управление соединениями, настройку индексов, а также миграции схемы базы данных. Такое решение оптимизирует работу с большими объёмами данных, ускоряет выборки при массовой загрузке ответов и облегчает сопровождение системы при её дальнейшем развитии.

Таким образом, спроектированная база данных обеспечивает полноту, непротиворечивость и эффективность хранения информации, необходимой для функционирования системы. Декомпозиция предметной области, нормализация данных, правильное распределение связей, а также использование индексов и триггеров создают устойчивую архитектуру, позволяющую масштабировать систему и адаптировать её к изменяющимся требованиям без существенных затрат на поддержку и доработку.

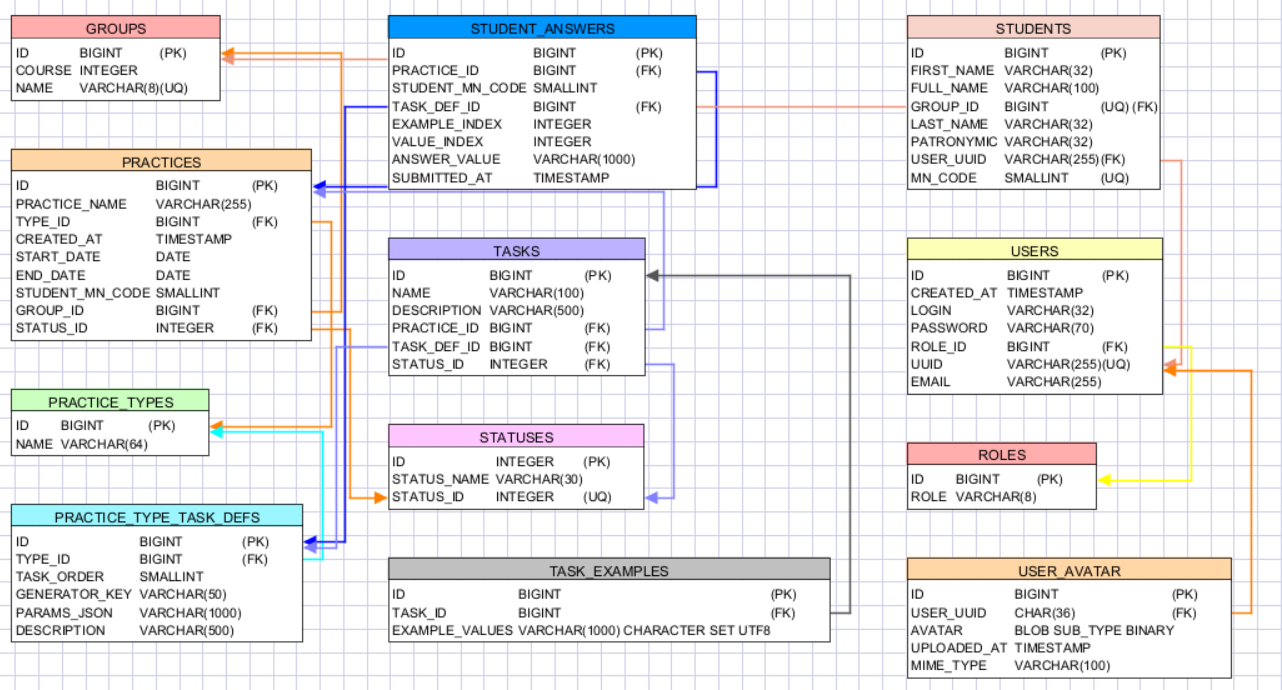


Рисунок 1 – ER-диаграмма базы данных

# 2.2 Разработка диаграмм информационной системы

В процессе проектирования информационной системы была проведена всесторонняя визуализация архитектуры и бизнес-логики, что позволило сформировать целостное и детализированное представление о структуре, функционировании и взаимодействиях между всеми элементами будущего решения. Такой подход обеспечивает прозрачность и управляемость на всех этапах разработки.

Для каждой ключевой подсистемы и основных бизнес-процессов были созданы графические модели, которые позволили выявить взаимосвязи между компонентами системы, определить точки взаимодействия пользователей с системой, проанализировать потоки данных и управление процессами, а также документировать функциональные и нефункциональные требования.

В ходе работы применялись разнообразные типы диаграмм, отражающие разные аспекты системы. Диаграммы прецедентов (use case) определяли основные сценарии использования с позиции различных категорий пользователей, задавая чёткие границы и функциональные возможности системы. Последовательности взаимодействий между объектами и компонентами фиксировались на диаграммах последовательностей, позволяя подробно описать порядок действий и обмен сообщениями в рамках конкретных сценариев.

Для отображения поведения элементов системы использовались диаграммы состояний, фиксирующие возможные состояния объектов и условия переходов между ними, что было важно для описания жизненных циклов ключевых сущностей. Архитектурные решения и распределение ответственности между модулями документировались с помощью диаграмм компонентов, а физическое развертывание системы на аппаратных ресурсах — с помощью диаграмм развертывания.

Каждая из этих моделей была построена с особенностей предметной области. Кроме того, поддержание актуальности диаграмм в процессе жизненного цикла системы значительно упростило внесение изменений и масштабирование, минимизируя риски ошибок и повышая адаптивность решения к новым требованиям.

Таким образом, разработка диаграмм информационной системы обеспечила системный и структурированный подход к проектированию, способствующий прозрачности, полноте охвата функционала и эффективному управлению процессом разработки и поддержки программного продукта.

1 Диаграмма прецедентов (Use Case diagram)

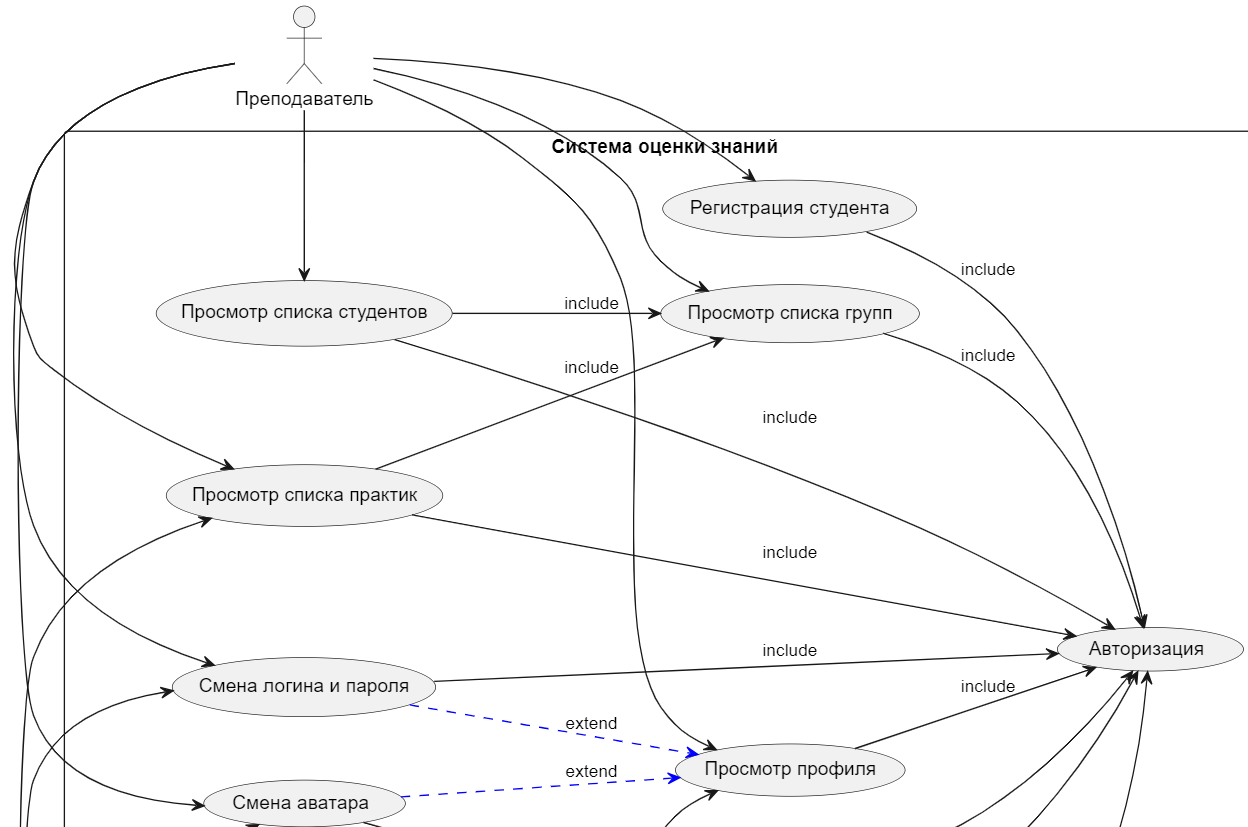


Рисунок 2.1 – Диаграмма прецедентов (преподаватель)

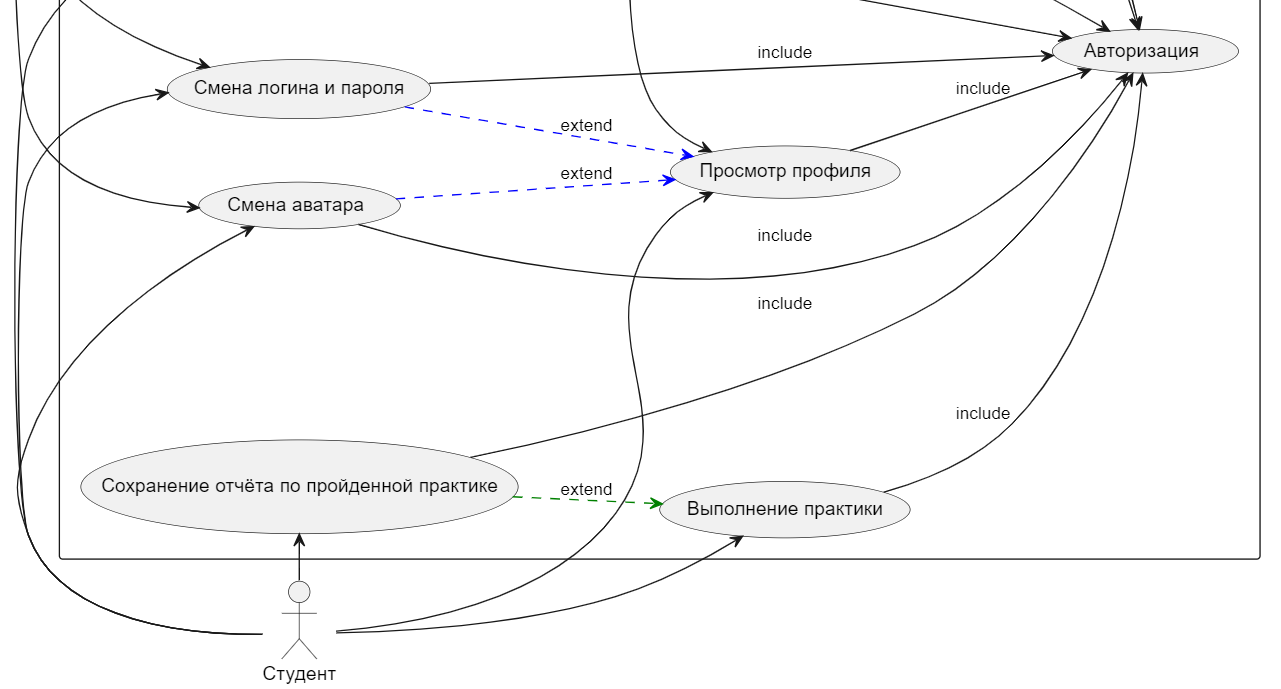


Рисунок 2.2 – Диаграмма прецедентов (студент)

В центре внимания — процесс, где каждый пользователь системы — будь то преподаватель или студент — начинает взаимодействие с системой через прецедент Авторизация. Эта операция обязательна и является основой доступа ко всем последующим функциям, что отражено использованием связи include. Такой подход подчёркивает, что без успешного входа в систему остальные действия невозможны, обеспечивая безопасность и контроль доступа.

Для преподавателя предусмотрены возможности, связанные с управлением учебным процессом. Он может регистрировать новых студентов, что является стартовым действием для расширения базы пользователей системы. Помимо этого, преподаватель имеет доступ к просмотру списков учебных групп и студентов, что позволяет организовать и контролировать учебный процесс на уровне групп и отдельных лиц. При этом просмотр списков студентов и практик взаимосвязан с просмотром групп — эти связи реализованы через include, поскольку информация о группах необходима для структурирования данных о студентах и практиках. Все эти функции и их взаимосвязи подробно отражены на рисунке 2.1.

Преподаватель также имеет возможность просматривать список практик, однако не выполняет их. Вся активность преподавателя направлена на управление и мониторинг, что чётко отражает разграничение ролей в системе.

Студент, в свою очередь, взаимодействует с системой несколько иначе. Он получает доступ к просмотру практических заданий и непосредственно выполняет практики. При выполнении практики возможна дополнительная операция — сохранение отчёта по пройденной практике, которая расширяет основной процесс выполнения. Эта связь обозначена как extend, что означает: сохранение отчёта — не обязательный шаг, но возможный и логически связанный с выполнением практики. Это отражает гибкость системы — студент может сохранить отчёт для контроля и архивации. Данный функционал и его связи представлены на рисунке 2.2.

Общий прецедент Просмотр профиля доступен обоим типам пользователей. Эта функция позволяет видеть личную информацию, что является важной частью пользовательского опыта. От неё в диаграмме идут две расширяющие связи (extend): Смена аватара и Смена логина и пароля. Это значит, что изменения профиля — опциональные действия, выполняемые по желанию пользователя, дополняющие основной просмотр. Такое разделение логично, так как просмотр профиля необходим, а модификации — второстепенны и зависят от пользовательского запроса.

Выбор include и extend в диаграмме грамотно отражает логику взаимосвязей и возможных вариантов поведения пользователей. Include подчёркивает обязательность вызова вспомогательных прецедентов для успешного выполнения основных, тогда как extend описывает дополнительные опциональные шаги, которые могут, но не обязаны, быть выполнены.

Таким образом, вся диаграмма демонстрирует стройную архитектуру взаимодействия, учитывающую безопасность (через авторизацию), разграничение прав и обязанностей преподавателя и студента, а также удобство и полноту пользовательского функционала, включая управление личными данными и отчётностью. Такой подход обеспечивает прозрачность и простоту в понимании бизнес-логики, а также служит надёжным руководством для последующей реализации и тестирования системы.

2 Диаграмма последовательности (Sequence diagrams)

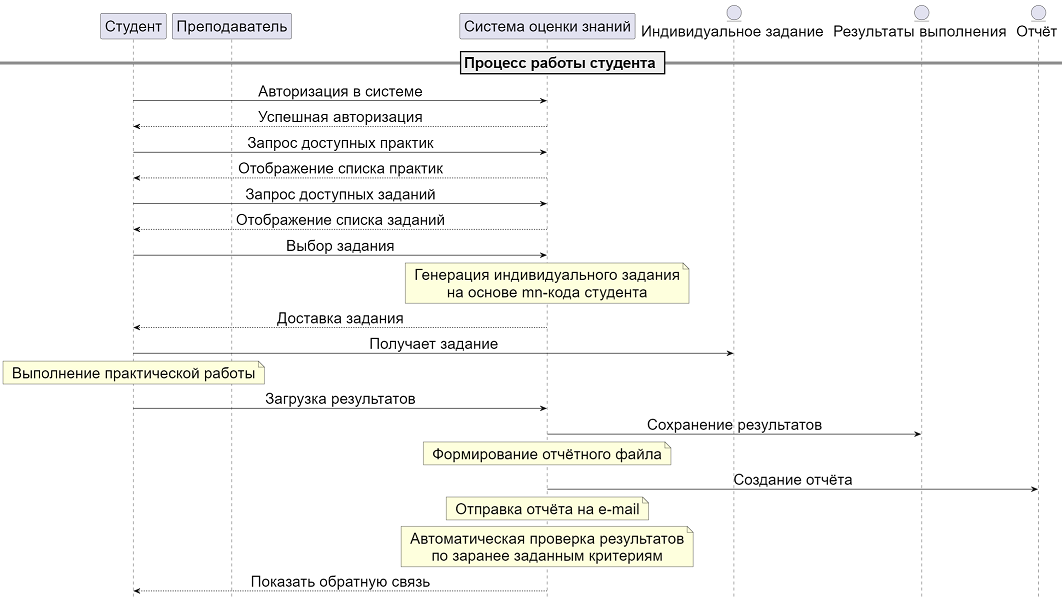


Рисунок 3.1 – Диаграмма последовательности (студент)

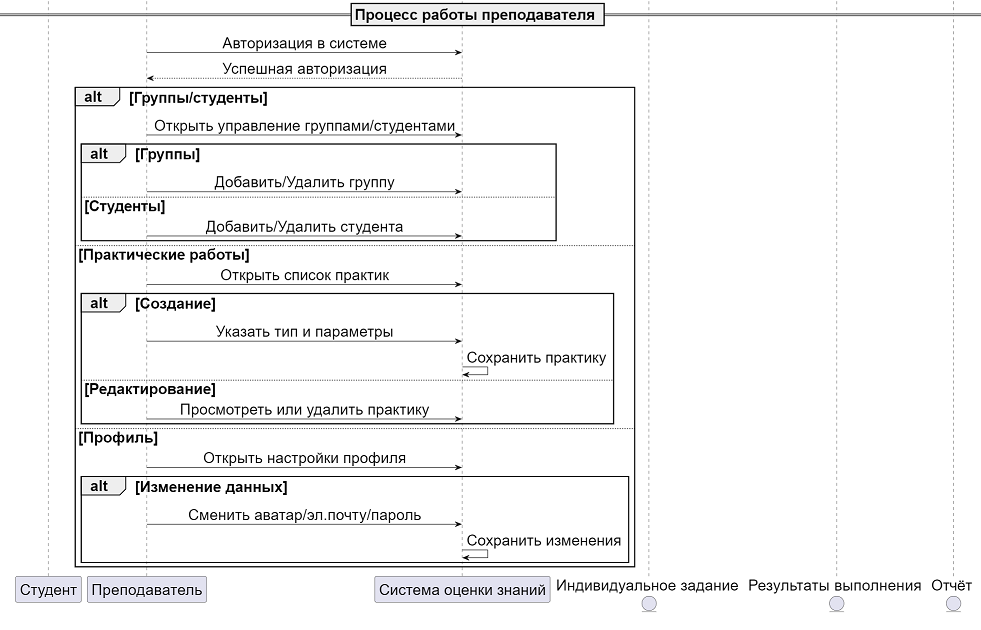


Рисунок 3.2 – Диаграмма последовательности (преподаватель)

Диаграмма последовательности, представленная на рисунке 3.1, детально иллюстрирует взаимодействие студента с системой оценки знаний на протяжении всего цикла выполнения практической работы.

Начинается процесс с авторизации студента в системе, где студент отправляет запрос на вход, а система подтверждает успешный доступ. Это обеспечивает контроль безопасности и персонализацию дальнейшей работы.

Далее студент запрашивает у системы список доступных практик, получая их отображение, что позволяет ориентироваться в текущих учебных заданиях. После этого происходит запрос списка заданий, что раскрывает конкретные варианты практик, доступные для выполнения.

Выбор конкретного задания инициирует процесс индивидуальной генерации практической работы на основе mn-кода студента — уникального идентификатора, который гарантирует, что задание будет персонализированным и исключит возможность списывания. Эта операция визуально отделена с помощью комментария, подчеркивая важность данного этапа.

После получения задания студент приступает к его выполнению, после чего загружает результаты обратно в систему. Эти результаты сохраняются в базе данных как объект «Результаты выполнения», что фиксирует прогресс и дает основу для оценки.

В системе автоматически формируется отчёт, отражающий ход и результаты выполнения, который затем отправляется на электронную почту студента. Кроме того, система проводит автоматическую проверку результатов по заданным критериям, что повышает объективность оценивания и снижает нагрузку на преподавателей.

В финале студент получает обратную связь с результатами проверки, завершая цикл взаимодействия.

Такой подробный сценарий обеспечивает прозрачность и полноту процесса с учётом всех ключевых операций, важных для качественной реализации автоматизированной системы.

На рисунке 3.2 отображён процесс взаимодействия преподавателя с системой, начиная также с этапа авторизации, что обеспечивает безопасность и разграничение доступа.

После успешного входа преподаватель может управлять учебными группами и списками студентов. Здесь используется конструкция alt, которая иллюстрирует выбор между управлением группами (добавление и удаление групп) и управлением студентами (добавление и удаление студентов). Такая логика отражает многообразие административных функций и показывает гибкость интерфейса.

Другой ветвью в условной конструкции является работа с практическими заданиями: преподаватель может просматривать список практик, создавать новые, указывая тип и параметры, либо редактировать или удалять существующие. Это охватывает полный цикл администрирования практик, позволяя оперативно управлять учебным процессом.

Важной частью диаграммы является управление профилем преподавателя, где через alt показаны действия по изменению личных данных: смена аватара, электронной почты и пароля. Все изменения сохраняются в системе, что обеспечивает актуальность и безопасность личной информации.

Диаграмма последовательно и ясно показывает основные сценарии работы преподавателя с системой, акцентируя внимание на административных функциях и персонализации.

3 Диаграмма состояний (State diagrams)

Определяет возможные состояния объектов или системы в целом и условия перехода между ними. Это важно для описания поведения сложных элементов, таких как жизненный цикл практики.

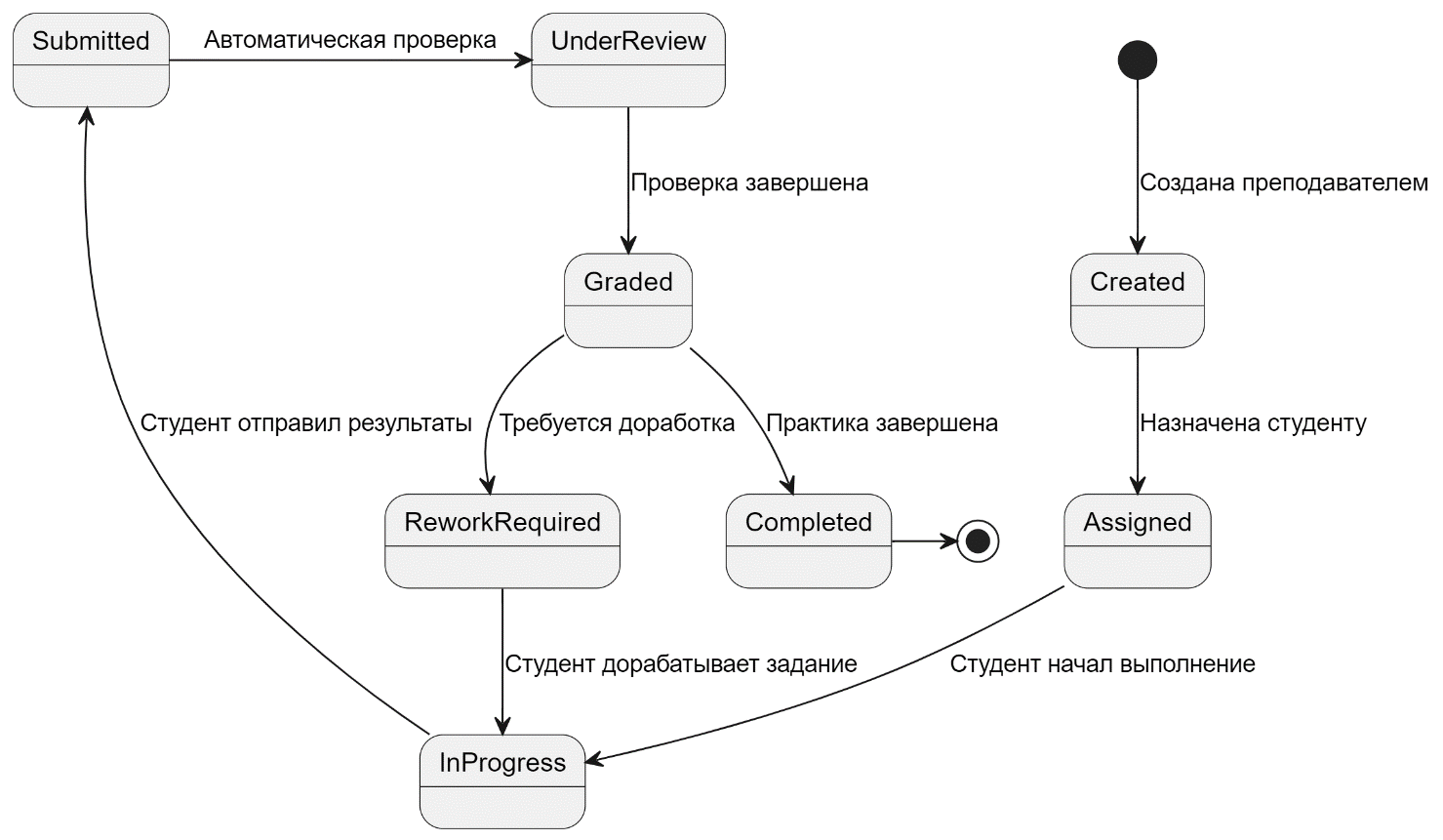


Рисунок 4 – Диаграмма состояний для практической работы

Диаграмма состояний для сущности «Практика», которая изображена на рисунке 4, в системе оценки знаний студентов отражает полный жизненный цикл выполнения практической работы от момента её создания до окончательного завершения. В начале объект практики находится в исходном состоянии, что означает, что он ещё не существует в системе.

Первым реальным состоянием является «Создана преподавателем». На этом этапе преподаватель формирует практику, задавая необходимые параметры, однако студент ещё не приступал к её выполнению. После этого практика переходит в состояние «Назначена студенту», что означает, что конкретному студенту, идентифицированному по уникальному mn-коду, предоставлен доступ к выполнению данной практической работы.

Далее практика переходит в состояние «Выполняется», когда студент начинает активно работать над заданием — получает его, формирует ответы и взаимодействует с системой. После завершения работы студент отправляет результаты в систему, и практика переходит в состояние «Отправлена». Это сигнализирует об окончании этапа выполнения с точки зрения студента.

Следующий этап — «На проверке». В этом состоянии происходит автоматическая (или частично ручная) оценка результатов работы по заданным критериям. На основе проверки практика может перейти в одно из двух состояний: «Оценена», если работа выполнена успешно, либо «Требуется доработка», если выявлены ошибки, и студенту необходимо внести исправления.

В случае необходимости доработки практика возвращается в состояние «Выполняется», позволяя студенту внести изменения и повторно отправить результаты. Если оценка выставлена, и все требования выполнены, практика переходит в конечное состояние — «Завершена». Это означает, что работа окончательно закрыта, а результаты доступны для просмотра и анализа.

Данная диаграмма состояний позволяет чётко структурировать процесс работы с практическими заданиями, обеспечивая прозрачность всех этапов, автоматизацию проверки, а также гибкость взаимодействия через возможность возврата к доработке. Она служит важным инструментом при проектировании и реализации функционала автоматизированной оценки, облегчая сопровождение и развитие системы.

4 Диаграмма компонентов (Component diagrams)

Отражают организацию и зависимости между основными модулями и подсистемами, обеспечивая понимание архитектурных решений и распределения ответственности.

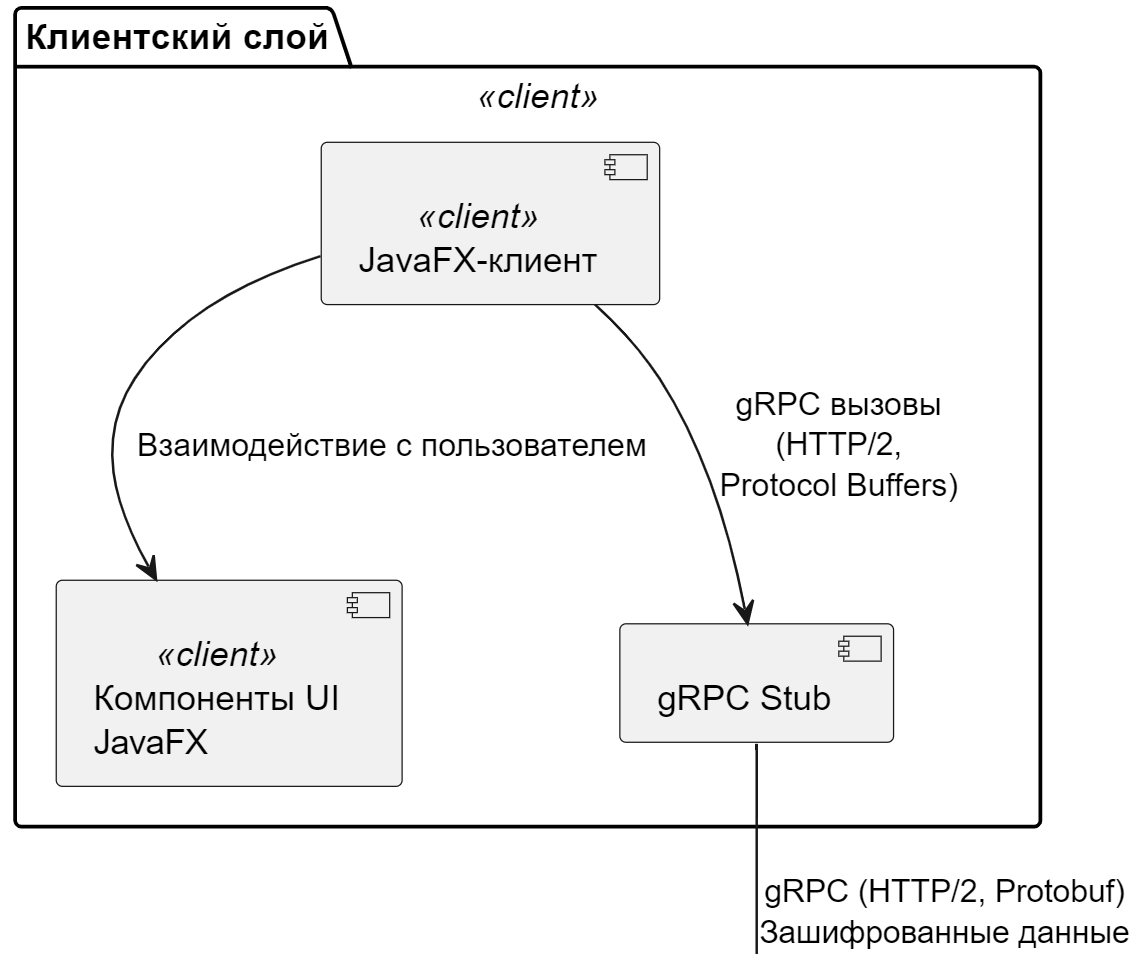


Рисунок 5.1 – Диаграмма компонентов клиентской части системы

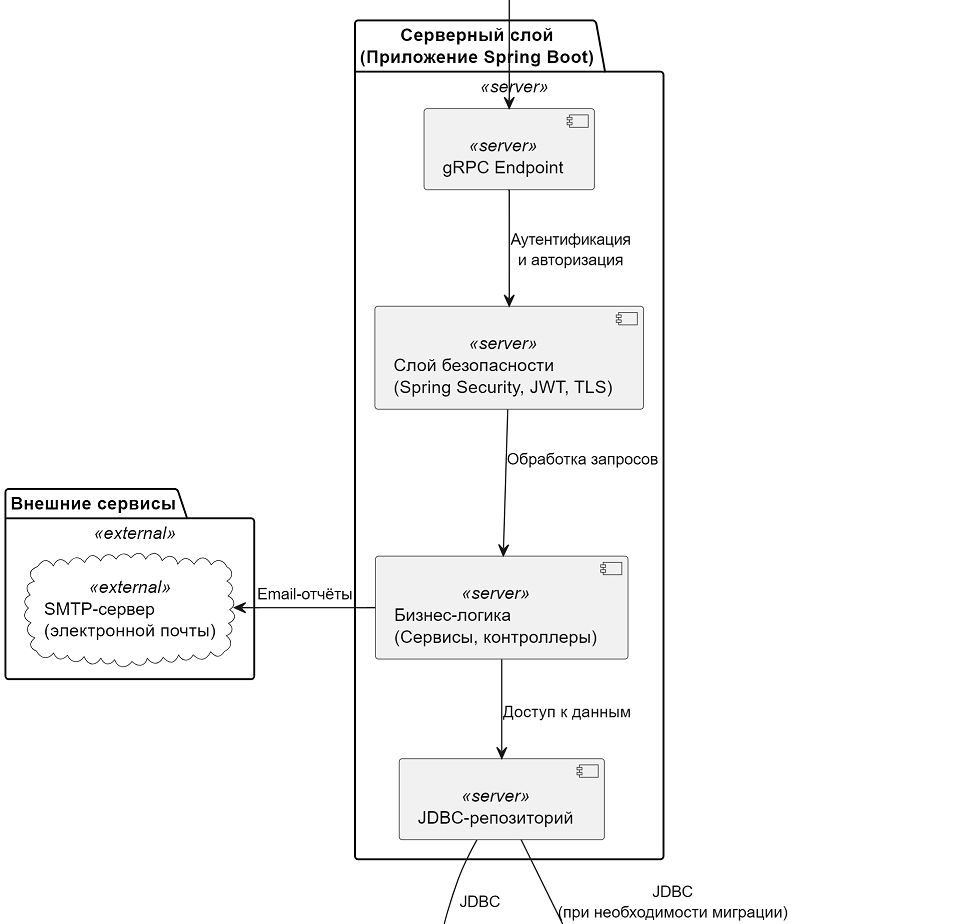


Рисунок 5.2 – Диаграмма компонентов серверной части системы

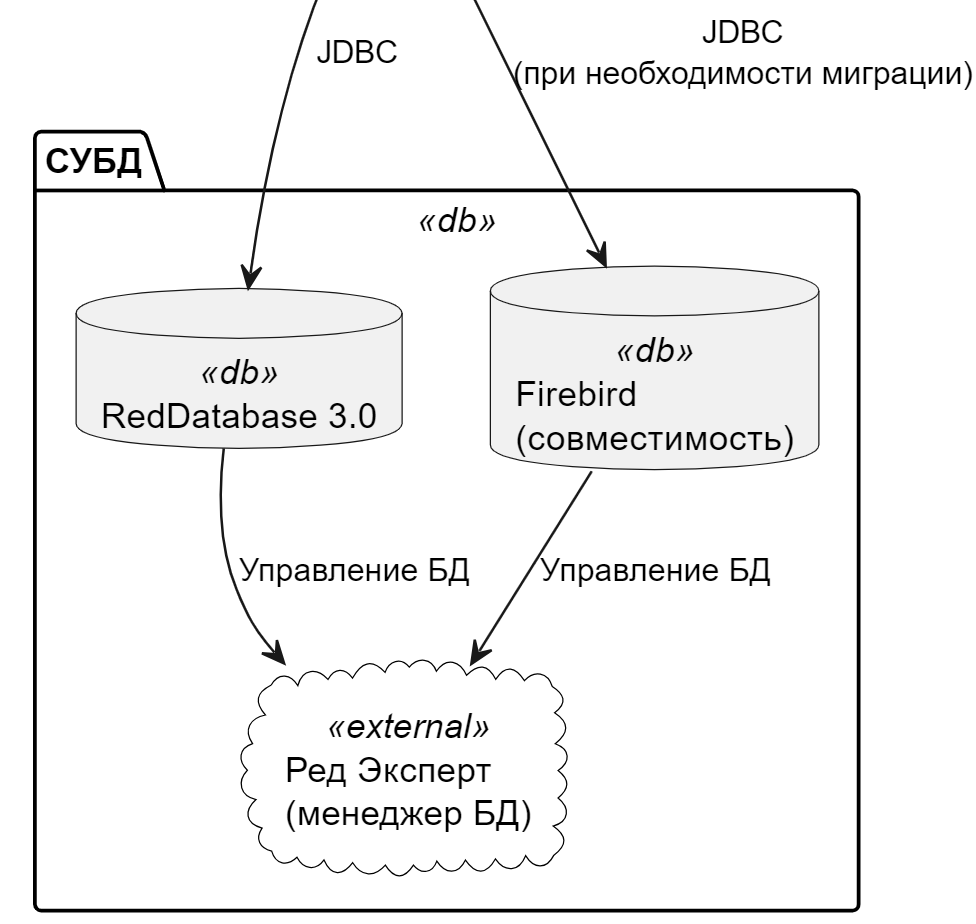


Рисунок 5.3 – Диаграмма компонентов базы данных

Диаграмма начинается с клиентского слоя на рисунке 5.1, где размещены JavaFX-клиент и его пользовательский интерфейс. JavaFXClient выступает в роли основного приложения на стороне пользователя, которое взаимодействует с UI-компонентами для отображения информации и сбора вводимых данных. Между JavaFXClient и JavaFXUI обозначена прямая связь, показывающая тесное взаимодействие. Кроме того, JavaFXClient осуществляет вызовы к серверу через gRPC Stub — специализированный клиент для удалённого вызова процедур, использующий протокол HTTP/2 и формат сериализации Protocol Buffers. Это гарантирует эффективную и безопасную передачу данных.

Переходя к серверному слою, который продемонстрирован на рисунке 5.2, диаграмма выделяет несколько ключевых компонентов. Основной точкой входа является gRPC Endpoint (GRPCServer), который принимает и обрабатывает запросы от клиента. Передача запросов защищена слоем безопасности, который реализует механизмы аутентификации и авторизации, включая Spring Security, JWT и TLS, что обеспечивает надёжную защиту системы и контроль доступа.

Далее запросы передаются в бизнес-логику, представленную сервисами и контроллерами Spring Boot, где реализуется основная функциональность системы: обработка данных, валидация, взаимодействие с базой. Взаимодействие бизнес-логики с базой данных организовано через JDBC-репозиторий, который обеспечивает доступ к данным с помощью стандартизированного интерфейса JDBC.

Далее следует слой баз данных на рисунке 5.3, в котором показаны две системы управления базами данных: RedDatabase 3.0 и Firebird. RedDatabase выступает основной СУБД, тогда как Firebird обозначена для совместимости и возможной миграции данных. Управление этими базами осуществляется через внешнее приложение «Ред Эксперт», выступающее в роли менеджера баз данных, обеспечивающего администрирование, мониторинг и обслуживание.

JDBC-репозиторий напрямую взаимодействует с обеими СУБД, что позволяет гибко переключаться или использовать обе системы, обеспечивая устойчивость и масштабируемость решения.

5 Диаграмма развертывания (Deployment diagram)

Показывает физическое распределение компонентов системы по серверам, узлам и другим аппаратным ресурсам, что позволяет планировать инфраструктуру и обеспечивает масштабируемость

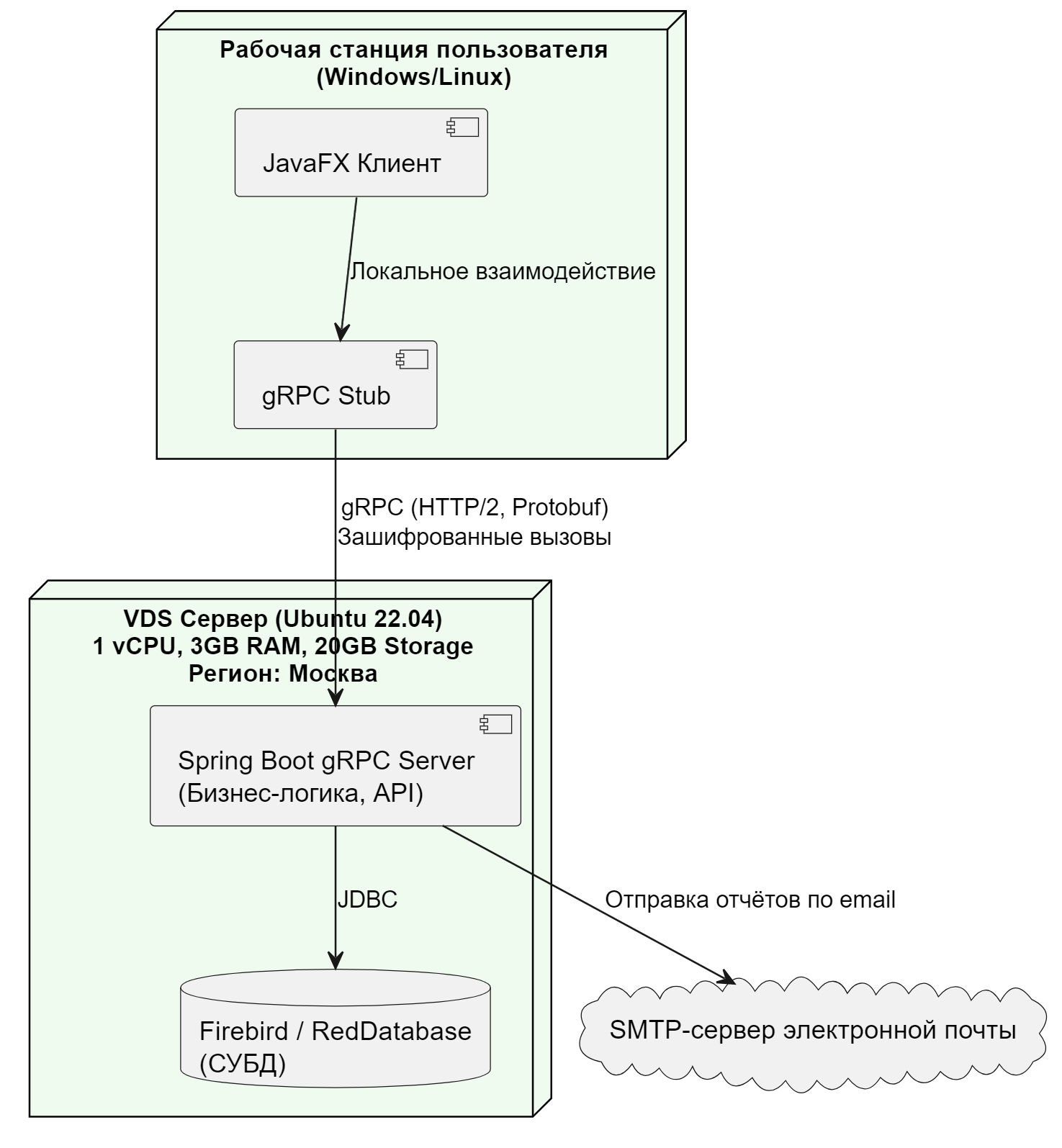


Рисунок 6 – Диаграмма развертывания

Диаграмма развертывания на рисунке 6 наглядно отражает физическую архитектуру системы оценки знаний студентов, демонстрируя распределение компонентов по инфраструктуре и их взаимодействие. В центре внимания находится выделенный виртуальный сервер (VDS), расположенный в регионе Москва, работающий под управлением Ubuntu 22.04. Этот сервер оснащён одним виртуальным ядром процессора, 3 ГБ оперативной памяти и 20 ГБ дискового пространства, что соответствует минимальным требованиям для размещения серверной части системы.

На этом сервере сосредоточены ключевые компоненты — Spring Boot gRPC сервер, который реализует бизнес-логику, API взаимодействия и обработку входящих запросов, а также система управления базами данных (СУБД), представленная Firebird или RedDatabase. Между сервером приложений и базой данных установлено соединение через JDBC — стандартный и надёжный интерфейс доступа к реляционным базам, обеспечивающий стабильность и совместимость при работе с данными пользователей, практик, заданий и ответов.

Пользовательская часть системы представлена рабочими станциями с операционными системами Windows и Linux, на которых запущено десктопное приложение JavaFX. Взаимодействие клиента с сервером организовано через локальный gRPC Stub, который отвечает за формирование и отправку запросов, а также получение ответов посредством протокола gRPC с использованием HTTP/2 и эффективной сериализации Protocol Buffers. Такая архитектура гарантирует высокую производительность, минимальные задержки и безопасность передачи данных.

Для автоматизации рассылки отчётов и уведомлений система интегрирована с внешним SMTP-сервером. Серверная часть напрямую обращается к этому облачному сервису, обеспечивая своевременную и надёжную доставку электронных писем студентам и преподавателям с результатами выполненных практических заданий.

Таким образом, диаграмма развертывания демонстрирует сбалансированную распределённую архитектуру, которая обеспечивает централизованное управление бизнес-логикой и данными, при этом снижая нагрузку на клиентские устройства и расширяя функциональность за счёт внешних сервисов.

# 3 Реализация информационной системы

# 3.1 Структура интерфейса системы

Клиентская часть информационной системы разработана с использованием фреймворка JavaFX, что обеспечивает современный, отзывчивый и расширяемый пользовательский интерфейс. Основное архитектурное решение ориентировано на модульность и переиспользование компонентов, а также на удобство взаимодействия пользователя с системой.

1 Общий обзор интерфейса

В приложении реализовано 10 основных окон, отображенных в таблице 2, каждое из которых выполняет отдельную роль в системе.

Таблица 2 – Основные окна информационной системы

|  |  |  |
| --- | --- | --- |
| № | Название файла | Назначение |
| 1 | auth.fxml | Окно авторизации пользователя |
| 2 | main.fxml | Главное рабочее окно после входа |
| 3 | menu.fxml | Навигационное меню, обеспечивающее переходы |
| 4 | profile.fxml | Страница профиля пользователя |
| 5 | groups.fxml | Управление группами студентов |
| 6 | students.fxml | Управление списком студентов |
| 7 | practices.fxml | Список доступных практических работ |
| 8 | single\_practice.fxml | Работа с конкретной практической работой |
| 9 | info\_about\_practice.fxml | Дополнительная информация о практике |
| 10 | custom\_alert.fxml | Кастомные диалоговые окна и предупреждения |

Начинается взаимодействие с системой с окна авторизации — auth.fxml. Это простое, но функциональное окно, где пользователь вводит логин и пароль. Поля для ввода расположены удобно, а кнопка входа активируется после проверки заполнения данных. В случае ошибки авторизации отображается понятное сообщение. Таким образом, окно обеспечивает проверку подлинности пользователя и является точкой входа в систему. Это можно увидеть на рисунке 7.

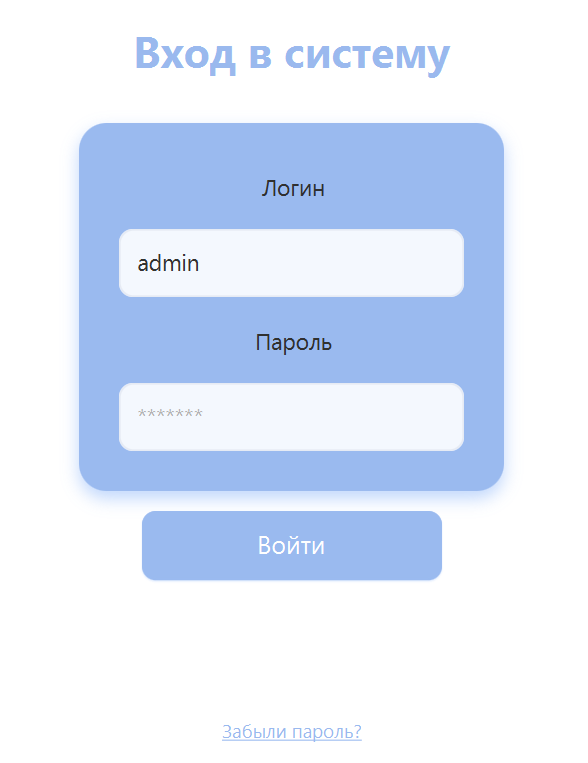


Рисунок 7 – Окно авторизации

После успешного входа пользователь попадает в главное рабочее окно — main.fxml. Это базовый каркас, который содержит слева меню навигации и основную область для динамического отображения содержимого. Такое архитектурное решение позволяет сохранять меню на месте при переходе между разделами, что значительно улучшает восприятие интерфейса и скорость работы. Основная область является контейнером, в который подгружаются остальные окна и страницы, обеспечивая плавный переход и предотвращая мерцание.

Навигационное меню, реализованное в файле menu.fxml (рисунок 8), расположено слева и предоставляет удобные ссылки на все ключевые разделы: профиль, группы студентов, список студентов, практические работы и дополнительную информацию. Меню визуально выделяет активный пункт, что помогает пользователю ориентироваться. При клике на элемент меню содержимое основной области меняется без перезагрузки всего окна.

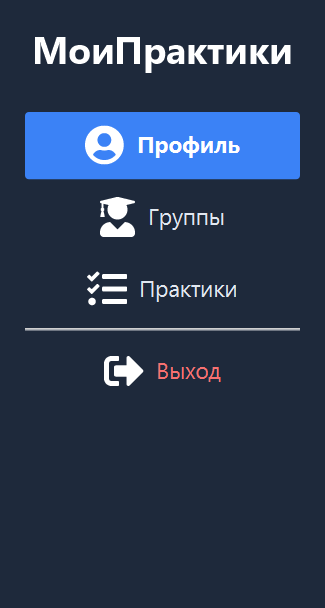


Рисунок 8 – Основное навигационное меню

Страница профиля — profile.fxml, изображенная на рисунке 9, служит для отображения и редактирования личной информации пользователя. Здесь можно увидеть аватар, который при необходимости можно изменить, а также поля с логином, email и возможностью смены пароля. Все изменения сохраняются асинхронно: при отправке данных на сервер пользователь видит индикатор загрузки, а в случае ошибок или успешного сохранения появляются информативные сообщения. Такой подход обеспечивает удобство работы и прозрачность взаимодействия.

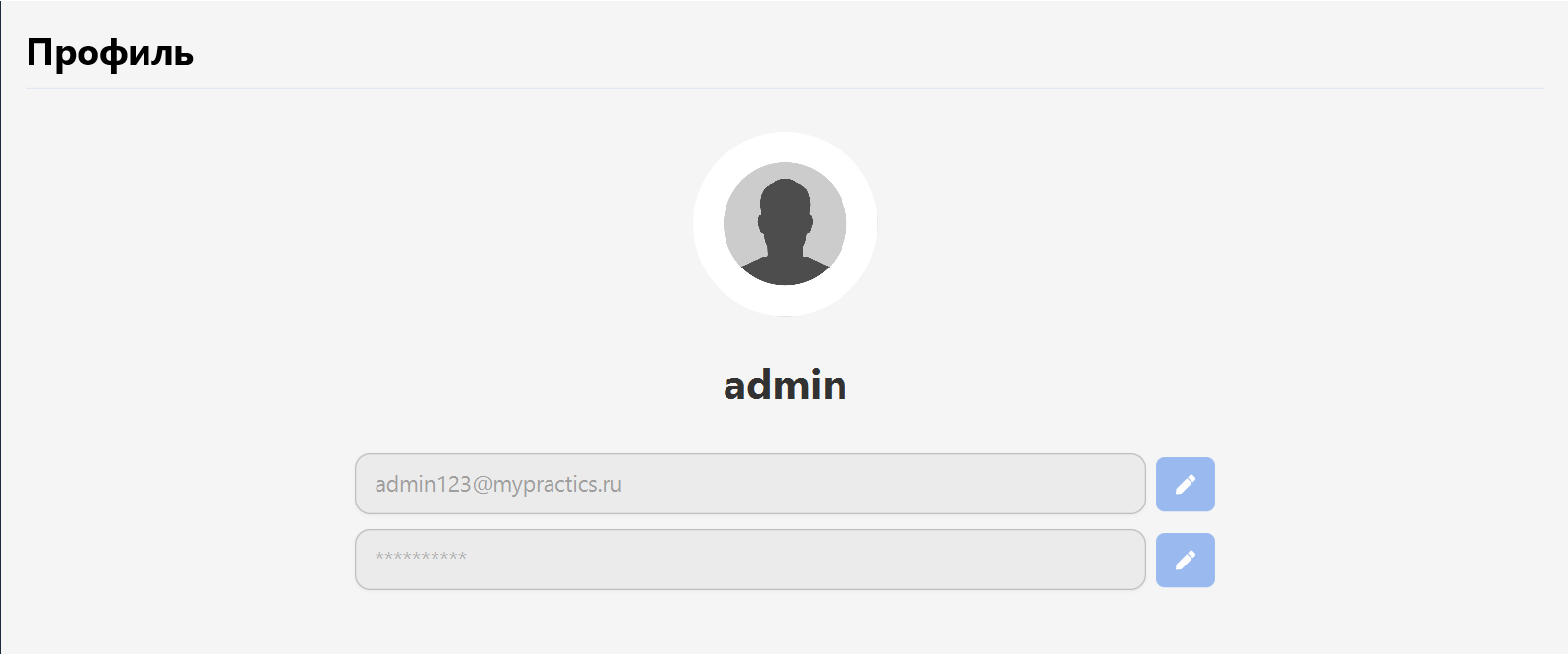


Рисунок 9 – Страница профиля

Управление учебными группами реализовано в окне groups.fxml. В этом окне, представленном на рисунке 10, преподавателю предоставлен список групп с ключевой информацией, такой как название группы и количество студентов. Преподаватель может добавлять новые группы, редактировать или удалять существующие через карточку группы. Для удобства предусмотрена фильтрация и поиск по группам. Все операции обновляют данные синхронно с сервером.

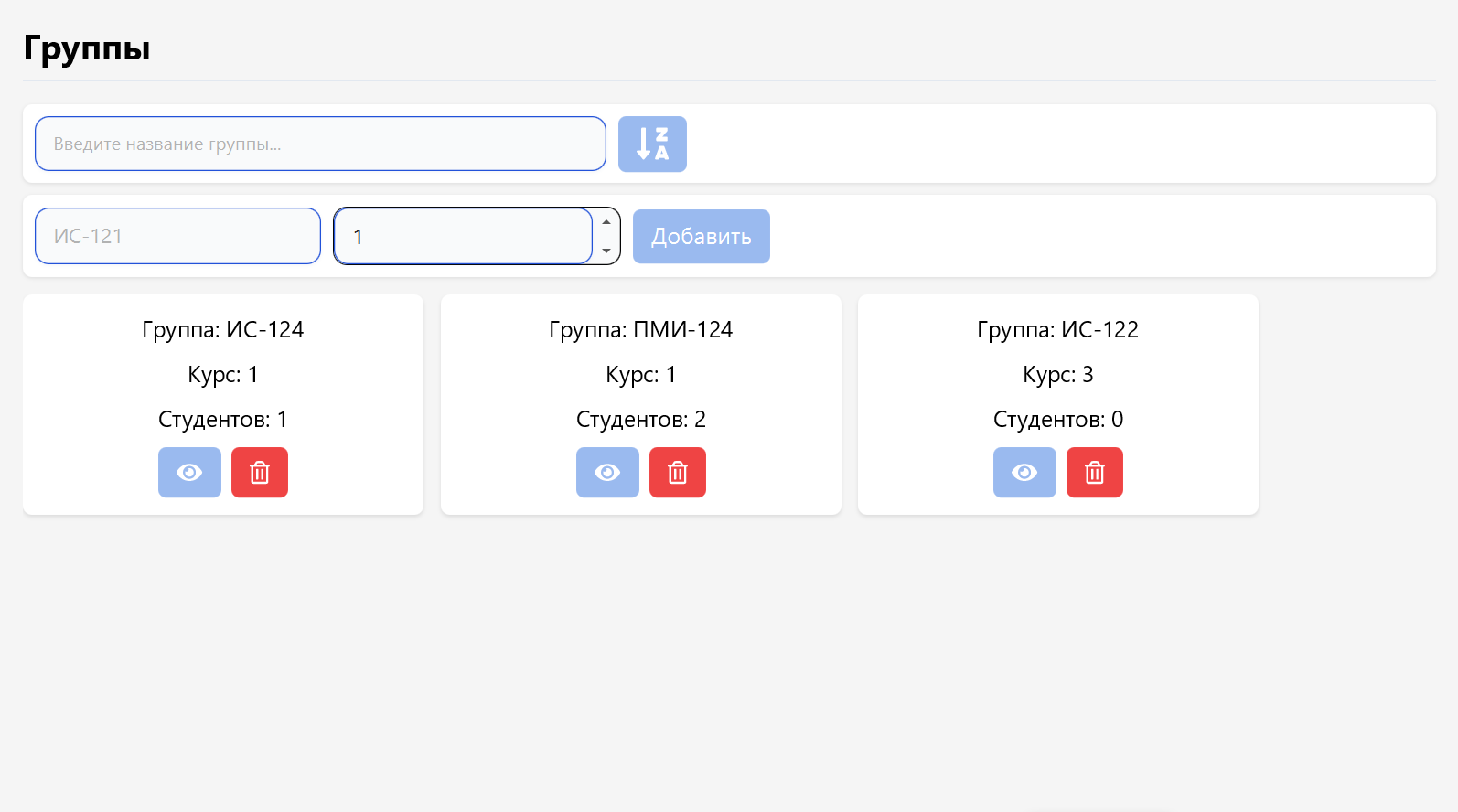


Рисунок 10 – Страница с группами студентов

Окно students.fxml позволяет работать со списком студентов. Внешний вид окна изображён на рисунке 11. Оно напоминает окно управления группами, но дополнительно включает сортировку и фильтрацию по различным параметрам, например по фамилии или группе. Здесь можно просматривать данные студента, редактировать их и добавлять новых участников учебного процесса. Все изменения строго валидируются и при необходимости отображаются сообщения с ошибками или подтверждениями.

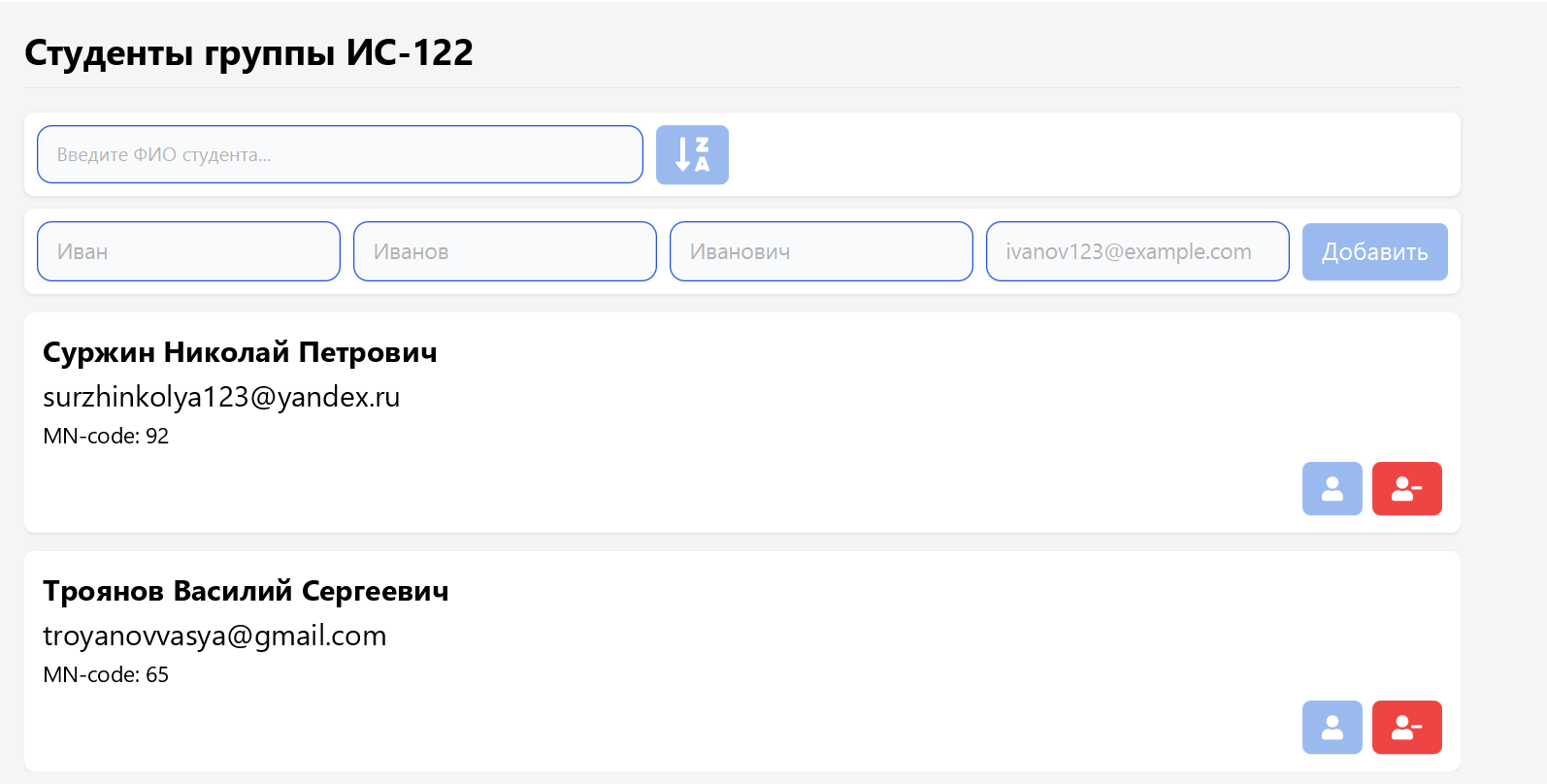


Рисунок 11 – Окно списка студентов

Список доступных практических работ выводится в окне practices.fxml, как показано на рисунке 12. Оно отображает актуальные практики с дополнительной информацией о них. Пользователь может найти нужную практику с помощью поиска или фильтров и перейти к её подробной работе. Состояние выполнения отражается визуально, что облегчает контроль прогресса.

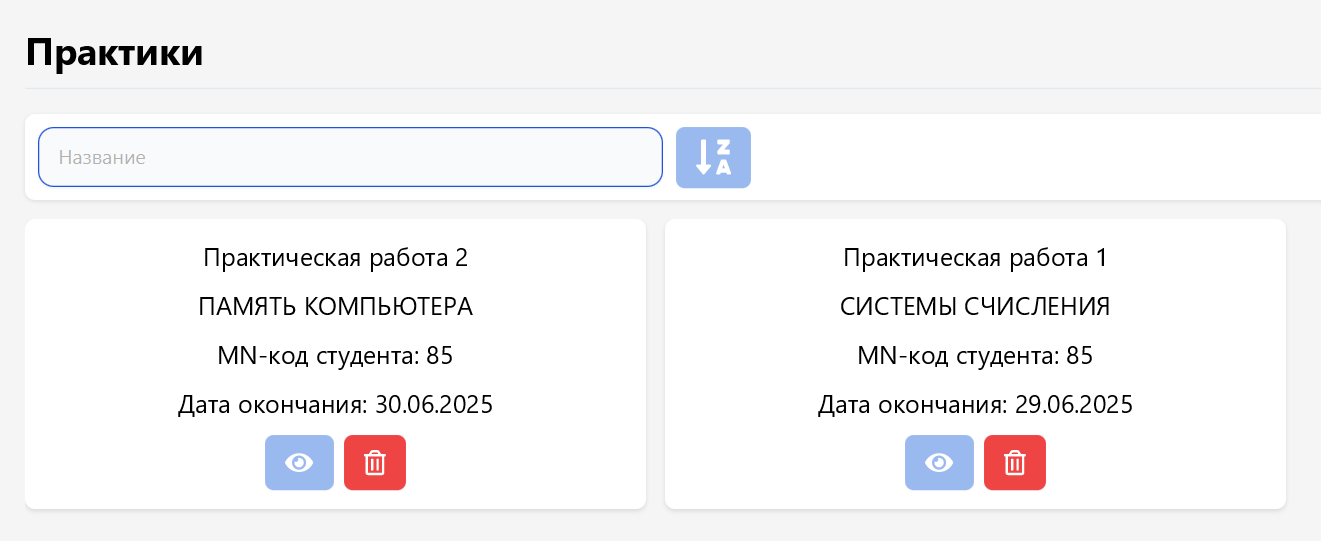


Рисунок 12 – Окно списка практических работ

Дополнительная информация о практике доступна в окне info\_about\_practice.fxml (рисунок 13). Оно содержит методические указания, цели практики и статистику выполнения как отдельного пользователя, так и всей группы. Также здесь могут быть размещены дополнительные материалы для скачивания или просмотра. Все данные загружаются динамически, что обеспечивает актуальность информации.

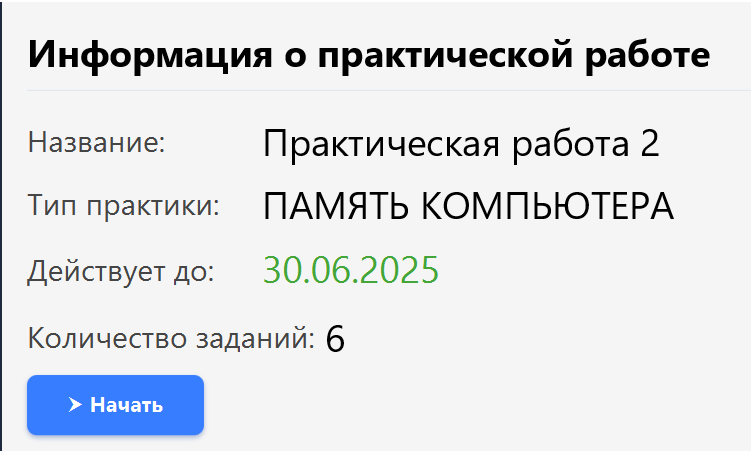


Рисунок 13 – Окно с информацией о практической работе

Работа с конкретной практической работой происходит в окне single\_practice.fxml на рисунке 14. Здесь пользователю предлагается набор заданий с интерактивными элементами, такими как текстовые поля. Все данные подгружаются асинхронно, а введённые ответы можно сохранить, отправить на проверку или изменить. Интерфейс снабжён кнопками для управления процессом и индикаторами статуса, чтобы пользователь чётко понимал свой прогресс.

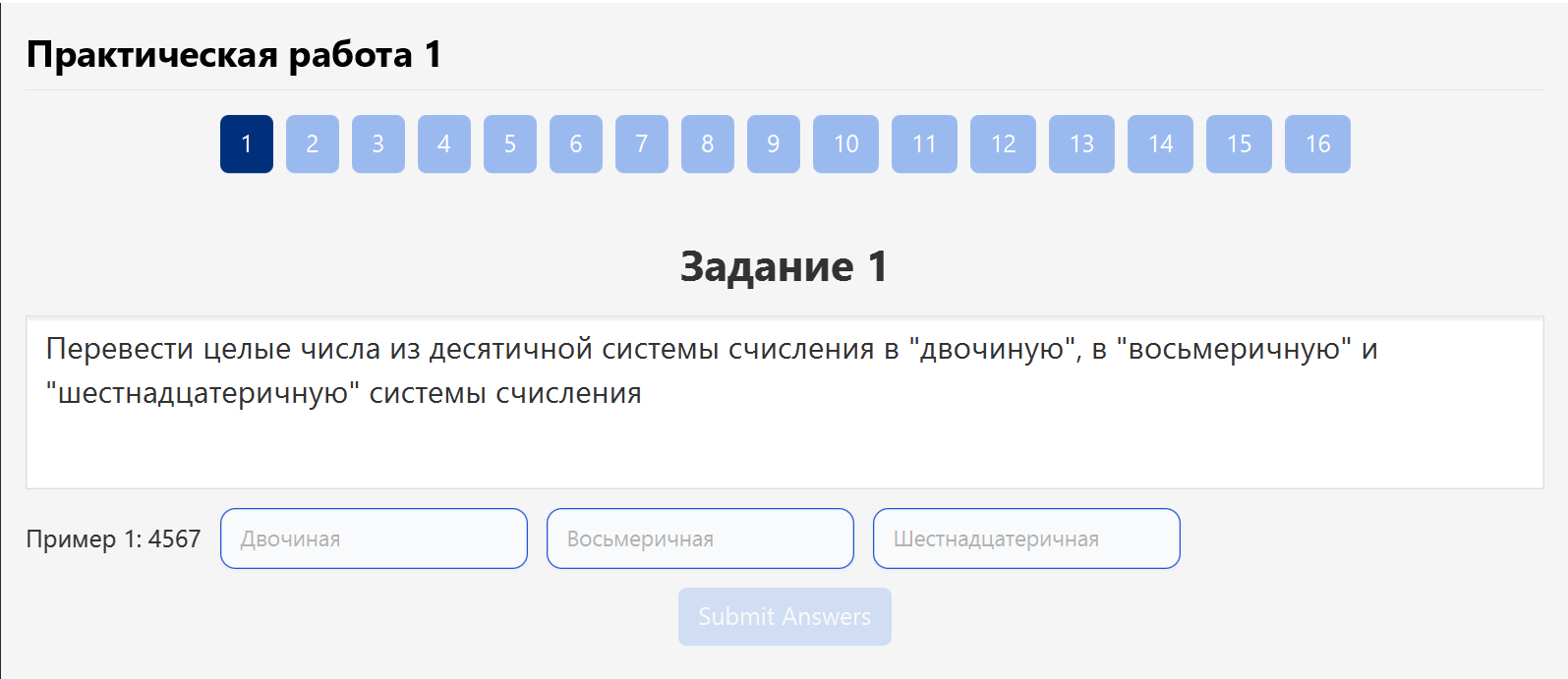


Рисунок 14 – Окно выполнения практической работы

Наконец, custom\_alert.fxml — это набор кастомных диалоговых окон, одно из которых показано на рисунке 15. Используются для вывода сообщений пользователю в системе. Это могут быть предупреждения, информационные сообщения или ошибки. Диалоговые окна имеют унифицированный стиль, содержат заголовок, описание и кнопки подтверждения или отмены. Такой подход обеспечивает единый пользовательский опыт при взаимодействии с системой.

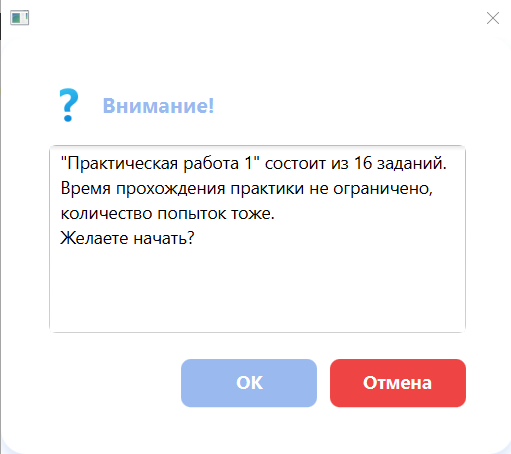


Рисунок 15 – Диалоговое окно

Таким образом, структура клиентского интерфейса объединяет функциональные и удобные окна, обеспечивая плавную навигацию и эффективное выполнение задач, что способствует комфортной работе преподавателей и студентов.

# 3.2 Программная архитектура системы

Информационная система построена на классической клиент-серверной архитектуре с чётким разделением ролей: сервер реализован на базе Spring Boot, а клиент — на JavaFX. Между ними организовано эффективное взаимодействие с помощью современного протокола gRPC, обеспечивающего высокую производительность и безопасность.

gRPC в проекте используется для передачи данных и вызова удалённых процедур через HTTP/2 с применением Protocol Buffers (protobuf). Этот подход позволяет достичь сжатия и строго типизированной сериализации сообщений, что снижает нагрузку на сеть и улучшает отзывчивость системы. Описание API и структур данных сосредоточено в proto-файлах, которые едины для клиента и сервера. На их основе генерируются DTO и сервисные классы, обеспечивая согласованность и надёжность обмена информацией. Каждый gRPC-запрос сопровождается заголовком с JWT access-токеном, который добавляется в метаданные клиента следующим образом:

Metadata.Key<String> AUTHORIZATION\_KEY = Metadata.Key.of("authorization", Metadata.ASCII\_STRING\_MARSHALLER);

headers.put(AUTHORIZATION\_KEY, "Bearer " + token);

Этот токен используется сервером для аутентификации и авторизации, реализуемых через Spring Security и специализированные компоненты, включая классы JwtConfig и JwtProvider. В случае истечения срока действия access-токена клиент автоматически обновляет сессию, используя refresh-токен, что обеспечивает безопасность и непрерывность работы без необходимости повторного ввода учетных данных.

Особое значение в системе занимает подсистема автоматической генерации и проверки заданий. Архитектурно она построена по паттерну «Фабрика», что позволяет удобно масштабировать и расширять функциональность без изменения базового кода. Фабрика генераторов заданий (TaskGeneratorFactory) создаёт конкретные генераторы, реализующие интерфейс TaskGenerator. Каждый генератор инкапсулирует логику создания заданий определённого типа, учитывая индивидуальные параметры студентов, включая уникальный mn-код, формируемый на уровне базы данных. Например, генераторы AlgebraLogicGenerator и IntegerRepresentationGenerator формируют задания по алгебраической логике и представлению чисел соответственно.

Параллельно фабрика валидаторов (TaskValidatorFactory) предоставляет валидаторы, реализующие интерфейс TaskValidator, которые проверяют корректность решений, поступающих от студентов. Такой подход создаёт замкнутый цикл «генерация — проверка» и обеспечивает объективность оценки. Реализации валидаторов, например FloatRepresentationValidator и NumeralCustomValidator, отвечают за специфику проверки ответов конкретных типов заданий.

Для поддержки безопасности паролей используется класс CryptoHash, который реализует алгоритм PBKDF2 с HMAC SHA-256. Пароли хранятся в виде защищённых хэшей с солью, что значительно повышает защиту учётных данных:

public static String hashPassword(String password, String salt) throws Exception {

PBEKeySpec spec = new PBEKeySpec(

password.toCharArray(),

Base64.getDecoder().decode(salt),

ITERATIONS,

KEY\_LENGTH

);

SecretKeyFactory skf = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");

byte[] hash = skf.generateSecret(spec).getEncoded();

return Base64.getEncoder().encodeToString(hash) + ':' + salt;

}

Утилита CredentialGenerator автоматизирует процесс создания уникальных логинов и паролей для студентов, снижая нагрузку на преподавателей. Логины формируются по шаблону с указанием имени группы и порядкового номера, например, is-121-student12.

Функционал формирования отчётов реализован через ReportPdfUtil, который генерирует PDF-документы с результатами выполненных практических работ. Эти отчёты могут автоматически отправляться на электронную почту пользователей посредством интеграции с SMTP-сервером, настройки которого содержатся в конфигурационном файле application.properties.

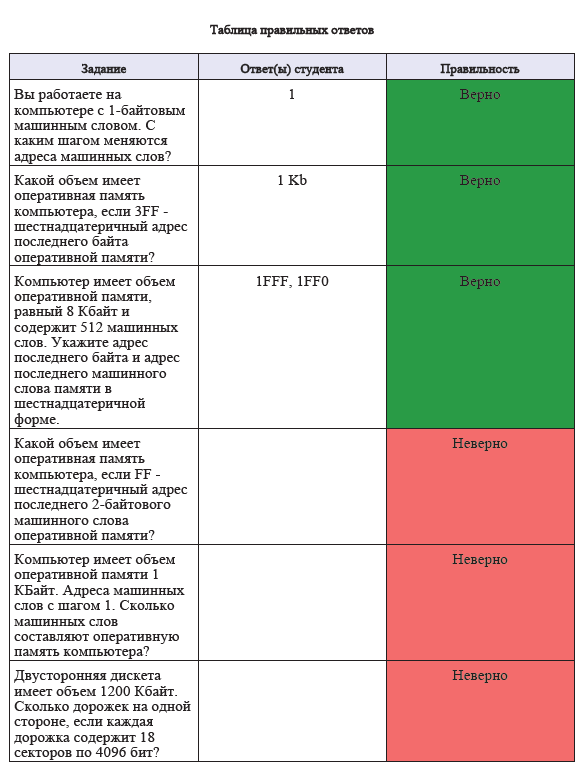


Рисунок 16 – Пример табличной части сформированного отчета

Таким образом, программная архитектура сочетает в себе современные технологии и паттерны проектирования, обеспечивая модульность, расширяемость и высокую безопасность. Использование gRPC гарантирует эффективный обмен данными между клиентом и сервером, а фабричный механизм генерации и валидации заданий служит ядром бизнес-логики, обеспечивая индивидуальный и объективный подход к оценке знаний студентов.

# 3.3 Реализация доступа к базе данных

В информационной системе для хранения и управления данными используется реляционная база данных RedDatabase 3.0. Для взаимодействия с базой данных применена технология Spring Data JPA, которая значительно упрощает работу с СУБД за счёт использования объектно-реляционного отображения (ORM). Spring Data JPA позволяет абстрагироваться от написания сложных SQL-запросов, работая с базой на уровне объектов Java, что повышает читаемость и сопровождение кода.

Spring Data JPA автоматически генерирует SQL-запросы на основе определённых в репозиториях методов, обеспечивая при этом поддержку операций создания, чтения, обновления и удаления (CRUD). Кроме того, данный фреймворк эффективно управляет жизненным циклом сущностей, транзакциями и кэшированием, что способствует сохранению целостности данных и повышению производительности системы.

Для каждой ключевой сущности проекта создан отдельный репозиторий, расширяющий интерфейс JpaRepository. Например, интерфейс GroupRepository предоставляет методы для работы с сущностью групп:

public interface GroupRepository extends JpaRepository<Group, Long> {

@EntityGraph(attributePaths = {"students", "students.user", "students.user.role"})

@Query("select g from Group g")

List<Group> findAllWithStudents();

}

Аналогично реализован PracticeRepository для практических заданий и другие репозитории, каждый из которых отвечает за взаимодействие с соответствующими моделями. Через репозитории сервисный слой обращается к базе данных, передавая объекты сущностей, которые JPA автоматически сопоставляет с таблицами и столбцами базы данных.

Подключение к базе данных настроено через файл application.properties, где указаны параметры подключения, драйвер, настройки пула соединений и форматирование запросов. Пример конфигурации выглядит следующим образом:

spring.datasource.url=jdbc:firebirdsql://localhost:3050/<path\_to\_db>?lc\_ctype=WIN1251

spring.datasource.username=sysdba

spring.datasource.password=pass

spring.datasource.driver-class-name=org.firebirdsql.jdbc.FBDriver

spring.jpa.database-platform=org.hibernate.community.dialect.FirebirdDialect

spring.jpa.hibernate.ddl-auto=update

spring.datasource.hikari.maximum-pool-size=10

spring.datasource.hikari.connection-timeout=30000

spring.datasource.hikari.idle-timeout=600000

spring.datasource.hikari.max-lifetime=1800000

spring.jpa.show-sql=false

spring.jpa.properties.hibernate.format\_sql=true

URL подключения содержит адрес сервера, порт, путь к файлу базы данных и параметр кодировки WIN1251, необходимый для корректного отображения русскоязычных данных. Логин и пароль указываются для доступа к базе. Важной частью является указание диалекта Hibernate для Firebird, который позволяет корректно формировать SQL-запросы. Параметры пула соединений HikariCP регулируют количество одновременных подключений и тайм-ауты, что позволяет оптимально использовать ресурсы. Форматирование и вывод SQL-запросов в лог отключено для повышения производительности.

Связи между сущностями отражены в JPA-моделях через аннотации, что обеспечивает отображение отношений таблиц в объектную модель. Например, связь между группами и студентами реализована через @OneToMany и @ManyToOne:

@Entity

public class Group {

@OneToMany(mappedBy = "group", cascade = CascadeType.ALL, fetch = FetchType.LAZY)

private List<Student> students;

}

@Entity

public class Student {

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "group\_id")

private Group group;

}

Подобные связи позволяют легко получать связанные объекты, а настройка каскадных операций и ленивой загрузки оптимизирует работу с данными. Кроме того, в моделях применяются ограничения, например, @Column(nullable = false), что гарантирует обязательность полей, а также уникальные ограничения для сохранения целостности данных. На уровне базы данных используются триггеры — например, для автоматической генерации уникального mn-кода студента при регистрации, что обеспечивает индивидуальность заданий и предотвращает дублирование.

Таким образом, использование Spring Data JPA в сочетании с RedDatabase 3.0 обеспечивает удобное, надёжное и производительное управление данными в системе, сокращая объём низкоуровневого кода и повышая качество архитектуры.

# 3.4 Реализация моделей данных

В информационной системе реализована двухуровневая модель данных, которая включает в себя JPA-модели для долгосрочного хранения информации в базе данных и DTO (Data Transfer Objects), описанные с помощью Protocol Buffers в .proto файлах, предназначенные для передачи данных между сервером и клиентом через gRPC. Такое разделение позволяет оптимизировать каждый этап работы с данными, обеспечивая гибкость, производительность и удобство поддержки.

JPA-модели: основа для хранения и обработки данных.

JPA-модели — это Java-классы, аннотированные с использованием спецификации Java Persistence API, которые представляют сущности предметной области и отражают структуру таблиц базы данных. Они являются ключевым компонентом серверной части, обеспечивая объектно-реляционное отображение (ORM) и управление данными на уровне бизнес-логики.

Например, модель студента включает обязательные поля, уникальные идентификаторы, а также связи с другими сущностями — в частности с группой:

@Entity

@Table(name = "students")

public class Student {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(nullable = false, unique = true)

private String uuid;

@Column(nullable = false)

private String name;

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "group\_id")

private Group group;

@Column(name = "mn\_code", nullable = false)

private int mnCode;

}

Такая модель позволяет работать с объектами в привычном для Java виде, а ORM автоматически переводит эти операции в SQL-запросы, что снижает количество шаблонного кода и ошибок.

Использование JPA даёт следующие преимущества: управление транзакциями на высоком уровне, поддержка ленивой загрузки данных (fetch = LAZY), каскадные операции, а также автоматическое создание и обновление схемы базы (в режиме ddl-auto=update).

Однако JPA-модели не предназначены для прямой передачи данных клиенту. Их избыточность и сложная структура неэффективны для сетевого обмена, что требует применения отдельного слоя DTO.

DTO из Protocol Buffers: эффективный транспорт данных.

Для коммуникации между клиентом и сервером используются DTO, описанные в файлах .proto с помощью языка описания Protocol Buffers. Protocol Buffers — это бинарный формат сериализации, разработанный Google, который обеспечивает сжатие и строго типизированную структуру сообщений.

DTO имеют лаконичную структуру, содержащую только те поля, которые нужны для передачи данных конкретного запроса или ответа. Например, описание DTO для студента:

message StudentDTO {  
 int64 id = 1;  
 string login = 2;  
 int32 mnCode = 3;  
 string fullName = 4;  
 string groupName = 5;  
 int64 groupId = 6;  
 string email = 7;  
}

Использование protobuf даёт ряд преимуществ. Во-первых, бинарный формат сообщений существенно меньше по объёму, чем аналогичные данные в JSON или XML, что снижает сетевую нагрузку и ускоряет передачу. Во-вторых, строгая типизация и порядок полей позволяют избежать ошибок в десериализации, а также легко поддерживать версии API с обратной совместимостью.

Вместе с тем, protobuf требует определённых усилий: для генерации классов DTO необходима интеграция с системой сборки и изучение синтаксиса описания сообщений. При отладке протокол сложнее визуализировать без специализированных инструментов. Кроме того, поддержка сложных бизнес-правил в DTO не предусмотрена — это задача серверной логики.

Разделение ролей и взаимосвязь моделей.

В системе строго разделены роли JPA-моделей и DTO. JPA-модели служат основой для работы с базой данных, обеспечивая управление данными и бизнес-правилами на сервере. DTO, напротив, — облегчённые объекты для эффективной передачи данных по сети.

В процессе обработки запроса данные из JPA-моделей конвертируются в DTO, которые отправляются клиенту. Аналогично, входящие с клиента DTO трансформируются в JPA-сущности для сохранения или обработки. Это разграничение позволяет избежать избыточности, уменьшить сетевой трафик и повысить безопасность (например, не передавать клиенту чувствительные данные, хранящиеся в моделях).

Ниже приведён пример конвертации сущности студента в DTO:

private StudentDTO toDto(Student student) {  
 StudentDTO.Builder builder = StudentDTO.*newBuilder*()  
 .setId(student.getId())  
 .setFullName(student.getFullName())  
 .setMnCode(student.getMnCode())  
 .setEmail(student.getUser().getEmail() == null ? "" : student.getUser().getEmail())  
 .setGroupId(student.getGroupId());  
 if (student.getGroup() != null) {  
 builder.setGroupName(student.getGroup().getName());  
 }  
 if (student.getUser() != null) {  
 builder.setLogin(student.getUser().getLogin());  
 }  
 return builder.build();  
}

Этот пример подчёркивает явное маппирование необходимых данных и отсутствие излишних деталей, не требуемых клиенту.

Использование моделей в сервисах и gRPC

Серверный слой содержит сервисы, которые работают с репозиториями JPA и бизнес-логикой. Они получают или модифицируют JPA-сущности и затем формируют соответствующие DTO для ответа по gRPC. gRPC-сервисы принимают запросы от клиента в виде DTO, передают их на сервисный слой для обработки и возвращают ответы в том же формате.

Такое разделение поддерживает архитектурную чистоту и удобство тестирования: бизнес-логика сосредоточена в сервисах и моделях, а транспортные объекты выполняют только функцию обмена.

Анализ и сравнительный обзор

Выбор такого подхода, когда JPA-модели и DTO реализуются отдельно, позволяет достичь баланса между удобством работы с данными и эффективностью передачи.

Если бы использовалась одна модель и для базы, и для обмена, возникли бы следующие проблемы: избыточность передаваемых данных, риск раскрытия внутренней информации, сложности с версионированием API и тестированием.

С другой стороны, поддержка двух видов моделей увеличивает сложность разработки из-за необходимости маппинга, но современные инструменты и практика показывают, что плюсы значительно перевешивают минусы.

Использование Protocol Buffers вместе с gRPC даёт возможность строго контролировать API и легко масштабировать систему, а Spring Data JPA обеспечивает надёжное хранение и управление данными.

В итоге, именно благодаря чёткому разделению ответственности между JPA-моделями и DTO, а также эффективному взаимодействию через gRPC, реализуется надёжная и масштабируемая архитектура моделей данных, которая соответствует требованиям современного информационного приложения для оценки знаний студентов.

# 4 Руководство по программному продукту

# 4.1 Руководство администратора

Раздел предназначен для системных администраторов, которые занимаются развертыванием, настройкой и поддержкой серверной части информационной системы. Здесь подробно описываются базовые действия по подготовке и конфигурации виртуального выделенного сервера (VDS), базы данных RedDatabase или Firebird, а также ключевые параметры конфигурации, мониторинг и обеспечение безопасности.

1 Настройка виртуального выделенного сервера (VDS)

Для стабильной работы системы рекомендуется использовать VDS с параметрами, обеспечивающими производительность и безопасность. Оптимально:

* ОС Linux (Ubuntu 20.04 LTS, CentOS 8 и др.)
* Минимум 2 ядра CPU с частотой не ниже 2.5 ГГц
* ОЗУ не менее 3 ГБ
* Дисковое пространство от 20 ГБ, предпочтительно SSD
* Открытые необходимые порты: 8099 (Spring Boot), 3050 (Firebird), 9090 (gRPC)
* Установленные JDK 17+ и Firebird 3.0

После получения сервера нужно выполнить базовое обновление и установить утилиты для администрирования:

sudo apt update && sudo apt upgrade -y

sudo apt install ufw htop wget curl nano –y

Утилиты обеспечивают удобное управление firewall, мониторинг ресурсов, загрузку файлов и редактирование конфигов.

2 Установка и настройка Firebird 3.0 или RedDatabase 3.0

Установка на Ubuntu Firebird:

sudo apt install firebird3.0-server firebird3.0-utils -y

sudo systemctl status firebird

Пароль пользователя SYSDBA рекомендуется сменить сразу после установки:

sudo dpkg-reconfigure firebird3.0-server

Файлы «Ред База Данных» 3.0 поставляются в виде бинарного пакета. При запуске его из любой графической системы (например, KDE) будет вызван мастер установки, который произведет сбор всей необходимой информации и установит СУБД «Ред База Данных» 3.0 на Ваш компьютер. Для установки СУБД «Ред База Данных» необходимо скопировать дистрибутивный файл RedDatabase-3.0.Х.Х-Х-linux-X.bin на жесткий диск, а в операционной системе назначить в правах этого файла разрешение на исполнение:

chmod +x RedDatabase-3.0.Х.Х-Х-linux-X.bin

После этого запустить установку СУБД «Ред База Данных»:

./RedDatabase-3.0.Х.Х-Х-linux-X.bin

Для установки сервера «Ред База Данных» 3.0 необходимы права суперпользователя (root). Существуют различные виды пакетов для 32-х и 64-х разрядных Unix-систем. Инсталляция СУБД «Ред База Данных» осуществляется с помощью стандартного мастера установки программ.

Если программа инсталляции запущена в текстовом режиме (с ключом --mode text), то она последовательно будет выводить запросы на подтверждение тех или иных параметров установки, таких как выбор компонентов или пароль пользователя SYSDBA. В случае, если на тот или иной запрос мастера установки предусмотрен ответ по умолчанию, то такой вариант обозначен заглавной буквой, и в этом случае этот вариант можно подтвердить нажатием клавиши «Ввод», в случае же, если выбор по умолчанию не предусмотрен, то необходимо обязательно ответить на вопрос «Да» (Y) или «Нет» (N) [18].

Основные конфигурационные файлы:

* /etc/firebird/3.0/firebird.conf — основные настройки сервера
* /etc/firebird/3.0/databases.conf — конфигурация баз

Firebird и RedDatabase по умолчанию слушают порт 3050 TCP, проверить можно командой:

sudo netstat -tuln | grep 3050

3 Настройка firewall (ufw)

Для безопасности необходимо включить ufw и ограничить доступ к сервисам:

sudo ufw enable

sudo ufw allow 9090/tcp # порт gRPC сервера

sudo ufw allow 3050/tcp # порт Firebird

sudo ufw allow 22/tcp # SSH

sudo ufw status

Для повышения безопасности можно ограничить доступ к Firebird только с доверенных IP:

sudo ufw allow from <trusted\_ip> to any port 3050

4 Настройка Firebird/RedDatabase для внешнего доступа

В файле /etc/firebird/3.0/firebird.conf (для firebird) или /opt/RedDatabase/firebird.conf параметр RemoteBindAddress должен быть установлен на 0.0.0.0, чтобы база слушала на всех интерфейсах:

RemoteBindAddress = 0.0.0.0

После изменений перезапустите сервис:

sudo systemctl restart firebird

5 Конфигурация серверного приложения (Spring Boot)

Основные параметры находятся в application.properties в ресурсах проекта. Важные настройки:

Подключение к базе данных через JDBC:

spring.datasource.url=jdbc:firebirdsql://ip:3050//path/to/database.fdb

spring.datasource.username=your\_username

spring.datasource.password=your\_password

spring.datasource.driver-class-name=org.firebirdsql.jdbc.FBDriver

Настройки JPA и Hibernate:

spring.jpa.database-platform=org.hibernate.community.dialect.FirebirdDialect

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=false

Параметры пула соединений Hikari:

spring.datasource.hikari.maximum-pool-size=10

spring.datasource.hikari.connection-timeout=30000

Кэширование (Caffeine):

spring.cache.type=caffeine

spring.cache.caffeine.spec=maximumSize=500,expireAfterAccess=10m

JWT-параметры:

jwt.secret.access=your\_access\_secret

jwt.secret.refresh=your\_refresh\_secret

jwt.access-token-expiration=3600

jwt.refresh-token-expiration=2592000

SMTP-конфигурация для рассылок:

spring.mail.host=smtp.gmail.com

spring.mail.port=587

spring.mail.username=your\_email@gmail.com

spring.mail.password=your\_email\_password

spring.mail.properties.mail.smtp.auth=true

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

app.mail.from=your\_email@gmail.com

Для секретных данных рекомендуется использовать защищённые переменные окружения или менеджеры секретов.

6 Запуск и управление серверным приложением

Для запуска Spring Boot gRPC сервера удобно использовать systemd.

Тестовый запуск в фоне:

cd /opt/myapp

nohup java -jar myserver.jar > server.log 2>&1 &

Проверка, что порт 9090 слушается:

sudo netstat -tuln | grep 9090

Создание systemd-сервиса /etc/systemd/system/myapp.service:

[Unit]

Description=My Spring Boot gRPC Service

After=network.target

[Service]

User=ubuntu

WorkingDirectory=/opt/myapp

ExecStart=/usr/bin/java -jar /opt/myapp/myserver.jar

SuccessExitStatus=143

Restart=on-failure

[Install]

WantedBy=multi-user.target

Запуск и включение автозапуска:

sudo systemctl daemon-reload

sudo systemctl start myapp

sudo systemctl enable myapp

7 Проверка соединений и работоспособности

Для Firebird/RedDatabase:

telnet <server\_ip> 3050

Для gRPC сервера:

grpcurl -plaintext <server\_ip>:9090 list

Также важно проверить корректное подключение к базе из Java-приложения и ответ сервера на gRPC-запросы.

8 Мониторинг и логирование

Для контроля состояния и отладки используются:

* логирование через Spring Boot (logback или log4j2) с информацией о gRPC вызовах, ошибках и безопасности. Настроена ротация логов;
* интеграция Spring Actuator для мониторинга метрик, состояния здоровья сервисов и статистики запросов;
* встроенные средства Firebird для мониторинга базы, анализ долгих запросов.

9 Настройка SMTP для рассылок

Система автоматически отправляет студентам регистрационные данные и отчёты.

Для корректной работы SMTP Gmail требует:

* разрешить SMTP приложениям (например, создать пароль приложения в настройках аккаунта);
* указать в конфигурации корректные логин и пароль;
* тестировать отправку через встроенные механизмы.

10 Обеспечение безопасности

Ключевые меры безопасности:

* регулярная ротация JWT секретов, хранение в безопасном хранилище;
* обновление Spring Boot, gRPC, библиотек и СУБД для устранения уязвимостей;
* управление правами пользователей через административный интерфейс и ограничение доступа на уровне сети (firewall);
* мониторинг попыток несанкционированного доступа с анализом логов и оповещениями.

11 Чек-лист перед вводом в эксплуатацию:

* ОС обновлена, установлены Firebird 3.0 и утилиты ufw, htop;
* Firebird слушает порт 3050, порт открыт в firewall;
* gRPC сервер слушает порт 9090, порт открыт;
* доступ к Firebird проверен локально и удалённо (isql, Java);
* gRPC сервер отвечает на запросы (grpcurl);
* Spring Boot сервис запущен через systemd и работает стабильно.

Данное руководство содержит все необходимые инструкции для подготовки, установки, конфигурации и сопровождения серверной части информационной системы с учётом современных требований по безопасности и производительности. Следуя ему, администратор сможет обеспечить надёжную работу приложения и своевременный мониторинг.

4.2 Руководство программиста

В данном разделе подробно описывается архитектура и ключевые компоненты проекта как на серверной, так и на клиентской части, а также особенности их взаимодействия.

1 Структура проекта на сервере и клиенте.

Проект разделен на две основные части: серверную и клиентскую. Серверная часть реализована с использованием Spring Boot и предоставляет API через gRPC. Клиентская часть — это JavaFX-приложение, которое взаимодействует с сервером посредством gRPC, обеспечивая удобный пользовательский интерфейс.

На сервере структура проекта отражает слои ответственности: модель данных, сервисы бизнес-логики, gRPC-интерфейсы и вспомогательные утилиты. Каждая из частей организована в отдельные пакеты для удобства поддержки и масштабирования.

Клиентская часть организована вокруг контроллеров для различных окон JavaFX, сервисов для общения с сервером, и утилит для обработки данных и управления асинхронностью.

2 Описание слоев архитектуры: модели, сервисы, gRPC, утилиты

Модели в серверной части представляют собой классы-сущности, соответствующие таблицам базы данных. Они аннотированы с помощью JPA и Hibernate, что позволяет автоматически управлять сохранением и загрузкой данных.

Сервисы содержат бизнес-логику, выполняя операции с моделями, включая валидацию, обработку и координацию данных. Они вызываются из gRPC-слоя и обеспечивают разделение ответственности.

gRPC-слой реализует протокол взаимодействия с клиентом. Здесь описаны сервисы, реализующие методы, определённые в proto-файлах. Эти сервисы обрабатывают запросы, вызывают бизнес-логику и формируют ответы.

Утилиты — вспомогательные классы, обеспечивающие работу с JWT-токенами, логированием, конфигурациями, а также общие функции, такие как парсинг и форматирование данных.

3 Использование JPA и Spring Data

Для работы с базой данных применяется Spring Data JPA, которая упрощает реализацию слоя доступа к данным. Основные сущности аннотируются с использованием JPA, что позволяет использовать возможности ORM — автоматическую генерацию SQL, кэширование и управление транзакциями.

Репозитории определяются как интерфейсы, расширяющие JpaRepository или CrudRepository, что даёт готовые методы для стандартных операций: сохранение, обновление, удаление и поиск. Дополнительные запросы создаются через метод-имена или с использованием аннотаций @Query.

4 Принципы работы с gRPC и proto-файлами

gRPC обеспечивает строго типизированный и эффективный обмен сообщениями между клиентом и сервером. Все API описаны в proto-файлах, где задаются сервисы, методы, а также структуры сообщений.

На основе proto-файлов автоматически генерируются Java-стабы и серверные интерфейсы. На сервере реализуются методы, обрабатывающие входящие запросы, валидацию и вызовы бизнес-логики. На клиенте — вызывается соответствующий stub, инкапсулирующий удалённое взаимодействие.

При расширении функционала достаточно добавить новые методы в proto-файл, сгенерировать код и реализовать логику на сервере и клиенте.

5 Организация и расширение фабрик генераторов и валидаторов заданий

Для обеспечения расширяемости системы генерации практических заданий реализованы фабрики генераторов (TaskGenerator) и валидаторов (TaskValidator). Каждая новая категория задания создаётся наследованием базового класса генератора или валидатора с переопределением методов генерации и проверки.

Для интеграции нового типа необходимо добавить соответствующую запись в таблицу PRACTICE\_TYPE\_TASK\_DEF в базе данных, указав в поле GENERATOR\_KEY уникальное имя класса генератора, а в params\_json — необходимые параметры. Это позволяет динамически расширять список типов заданий без изменения кода ядра.

6 Реализация JWT аутентификации и интеграция с Spring Security

Безопасность API обеспечивается с помощью JWT (JSON Web Token). При успешной аутентификации сервер генерирует пару токенов — access и refresh. Access-токен используется для доступа к защищённым методам gRPC.

Spring Security интегрирован в серверную часть и реализует фильтр для проверки токенов в каждом gRPC-запросе. При получении запроса токен валидируется, и на основе его содержимого формируется объект аутентификации с ролями пользователя.

Обновление токенов реализуется через отдельный сервис, позволяющий продлить сессию без повторного ввода учетных данных.

7 Асинхронная обработка и работа с CompletableFuture на клиенте

Для поддержания отзывчивости интерфейса JavaFX-клиента используется асинхронное выполнение операций, связанных с сетевыми запросами к серверу. Для этого применяется CompletableFuture, позволяющий выполнять gRPC-вызовы в фоновом потоке.

Результаты асинхронных операций возвращаются обратно в UI-поток с помощью методов thenAccept и Platform.runLater(), что исключает блокировку интерфейса. Такая архитектура обеспечивает плавный пользовательский опыт и позволяет масштабировать обработку данных.

8 Обработка ошибок и рекомендации по логированию

На сервере ошибки контролируются с помощью централизованной обработки исключений. Необработанные исключения логируются с использованием SLF4J и Logback, с указанием контекста запроса и стека вызовов.

Для gRPC предусмотрены специальные обработчики, которые преобразуют исключения в корректные статусы ответа, например UNAUTHENTICATED или INTERNAL.

На клиенте ошибки отображаются пользователю в удобной форме через диалоги и логируются локально при необходимости. Рекомендуется добавлять информативные сообщения и коды ошибок, чтобы облегчить диагностику.

Инструкции по добавлению новых типов заданий и расширению функционала

Для добавления нового типа задания необходимо:

* создать класс генератора, унаследованный от TaskGenerator, реализовав методы генерации контента;
* создать класс валидатора, унаследованный от TaskValidator, реализовав проверку корректности ответов;
* добавить запись в справочник PRACTICE\_TYPE\_TASK\_DEF, указав уникальный ключ генератора и параметры в формате JSON;
* обновить proto-файлы, если требуется новый тип сообщений или сервисных методов;
* при необходимости расширить клиентскую часть для корректного отображения и обработки нового типа заданий.

Таким образом архитектура остаётся гибкой и масштабируемой, позволяя безболезненно внедрять новые категории практик с минимальными изменениями в ядре системы.

4.3 Руководство пользователя

Данный раздел содержит подробное описание основных функций и пользовательского интерфейса системы, а также рекомендации по эффективной работе с приложением.

1 Вход в систему и работа с интерфейсом

Для доступа к системе необходимо авторизоваться, используя свои учетные данные — логин и пароль. Как выглядит экран авторизации показано на рисунке 7.

После успешного входа открывается главное рабочее окно, где отображается навигационное меню и список доступных функций.

Интерфейс выполнен в современном стиле, с удобной навигацией и логичным расположением элементов. Меню позволяет быстро перемещаться между разделами: профиль, группы, студенты, практические задания и отчёты.

2 Навигация и описание основных окон

Главное окно служит отправной точкой для пользователя. В нём расположено навигационное меню с основными разделами (см. рисунок 8).

Окно профиля отображает персональные данные пользователя — имя, email, аватар. Здесь же реализованы функции редактирования профиля и изменения пароля (см. рисунок 9).

Раздел управления группами (см. рисунок 10) и студентами (см. рисунок 11) предназначен для преподавателей, обеспечивая создание, редактирование и просмотр учебных групп и списка студентов.

В разделе практик (см. рисунок 12) пользователь может выбрать доступные задания, выполнить их, а также получить обратную связь по результатам.

Отчёты позволяют просматривать результаты и статистику выполнения практических заданий, включая детальный разбор ошибок.

3 Редактирование профиля, загрузка аватара, изменение пароля и email

В окне профиля (см. рисунок 9) доступен функционал обновления персональных данных. Пользователь может загрузить новый аватар, выбрав изображение с локального устройства. Для изменения пароля необходимо ввести 4-х значный код, отправленного на email, что обеспечивает безопасность учётной записи.

Изменение email требует подтверждения, чтобы избежать ошибок. Все изменения сохраняются на сервере и синхронизируются с профилем пользователя.

4 Выполнение практических заданий и получение обратной связи

Пользователь выбирает практическую работу из списка доступных (см. рисунок 12). После запуска практической работы открывается интерфейс с описанием, вопросами и полями для ответов (см. рисунок 14).

Ответы сохраняются автоматически и могут быть отредактированы до отправки на проверку. После завершения задания система выполняет валидацию и отображает результаты, отмечая правильные и ошибочные ответы.

Преподаватели имеют расширенный доступ к статистике по группам и отдельным студентам.

5 Обновление сессии и работа с безопасностью (авторизация с токенами)

Система использует JWT-токены для аутентификации и защиты данных. При входе выдается access-токен с ограниченным сроком действия.

Если токен истекает, клиент автоматически обращается к серверу за обновлением, используя refresh-токен, что позволяет поддерживать сессию без повторного ввода данных.

Для безопасной работы не рекомендуется передавать учетные данные третьим лицам и использовать публичные сети без VPN.

# 5 Тестирование

Тестирование разработанной информационной системы проводилось с целью проверки корректности функционирования всех ключевых компонентов, обеспечения устойчивости работы при различных сценариях использования и подтверждения соответствия системы требованиям технического задания.

В рамках тестирования были выполнены модульные, интеграционные и функциональные проверки. Модульное тестирование охватило отдельные компоненты серверной и клиентской частей, включая сервисы бизнес-логики, репозитории, gRPC-сервисы и контроллеры интерфейса. Особое внимание уделялось проверке алгоритмов генерации и валидации заданий, а также корректности работы механизма аутентификации с использованием JWT.

Интеграционные тесты проводились для оценки взаимодействия между слоями системы, в частности — корректности обмена данными между клиентом и сервером по протоколу gRPC, правильности преобразования JPA-моделей в DTO и обратно, а также стабильности работы с базой данных Firebird.

Функциональное тестирование включало проверку пользовательских сценариев: авторизации, навигации по интерфейсу, выполнения практических заданий, редактирования профиля и получения отчетов. Тестировалась асинхронная обработка запросов и отображение статусов загрузки, что обеспечило плавный пользовательский опыт.

В процессе тестирования выявленные ошибки и несоответствия оперативно фиксировались и устранялись, что позволило повысить качество и надёжность системы.  
Тест для проверки одного из генератора заданий, результат которого можно увидеть на рисунке 17.

public class NumberGeneratorTest {  
  
 @Test  
 void randomBinaryNumberHasValidDigits() {  
 String value = NumberGenerator.*generateRandom*(SystemBase.*BINARY*, 4, 2);  
 *assertTrue*(value.matches("[01]{4}(\\.[01]{2})?"));  
 }  
  
 @Test  
 void customBinaryGenerationRepeatsPattern() {  
 String result = NumberGenerator.*generateCustom*(SystemBase.*BINARY*, 2, 1, "10", "1");  
 *assertEquals*("1010.1", result);  
 }  
}

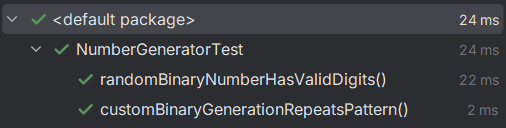


Рисунок 17 – Результат теста генерации чисел для двоичной системы счисления

Тест для проверки авторизация по JWT токену, результат которого можно увидеть на рисунке 18.

public class JwtProviderTest {  
  
 @Test  
 void accessTokenContainsSubjectAndRole() {  
 String secret = "verysecretkey12345678901234567890";  
 SecretKey key = new SecretKeySpec(secret.getBytes(), "HmacSHA256");  
 JwtEncoder encoder = new NimbusJwtEncoder(new ImmutableSecret<>(key));  
 JwtProvider provider = new JwtProvider(encoder, encoder, 3600, 7200);  
 Role role = new Role();  
 role.setRole("student");  
 User user = new User();  
 user.setLogin("bob");  
 user.setRole(role);  
 String token = provider.generateAccessToken(user);  
 JwtDecoder decoder = NimbusJwtDecoder.*withSecretKey*(key).build();  
 Jwt decoded = decoder.decode(token);  
 *assertEquals*("bob", decoded.getSubject());  
 *assertEquals*("STUDENT", decoded.getClaimAsString("role"));  
 *assertTrue*(decoded.getExpiresAt().isAfter(Instant.*now*()));  
 }  
}

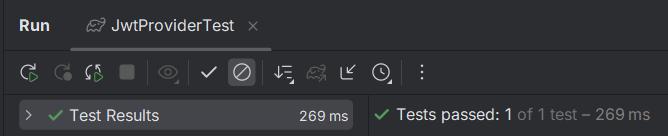


Рисунок 18 – Результат теста авторизации пользователя по access-токену

Тест для проверки извлечения данных через gRPC метод, результат которого можно увидеть на рисунке 19.

public class StudentGrpcServiceTest {

private Server server;

private ManagedChannel channel;

private StudentService studentService;

@BeforeEach

void setUp() throws IOException {

studentService = mock(StudentService.class);

StudentGrpcService service = new StudentGrpcService();

ReflectionTestUtils.setField(service, "studentService", studentService);

String serverName = InProcessServerBuilder.generateName();

server = InProcessServerBuilder.forName(serverName).directExecutor().addService(service).build().start();

channel = InProcessChannelBuilder.forName(serverName).directExecutor().build();

}

@AfterEach

void tearDown() {

if (channel != null) channel.shutdownNow();

if (server != null) server.shutdownNow();

}

@Test

void getStudentByMnCodeReturnsDto() {

Student st = new Student();

st.setId(1L);

st.setMnCode(10);

st.setFullName("Ivan Ivanov");

st.setGroupId(2L);

User user = new User();

user.setLogin("ivan");

st.setUser(user);

when(studentService.getByMnCode(10)).thenReturn(Optional.of(st));

StudentServiceGrpc.StudentServiceBlockingStub stub = StudentServiceGrpc.newBlockingStub(channel);

StudentDTO dto = stub.getStudentByMnCode(MnCodeRequest.newBuilder().setMnCode(10).build());

assertEquals(1L, dto.getId());

assertEquals("ivan", dto.getLogin());

verify(studentService).getByMnCode(10);

}

}

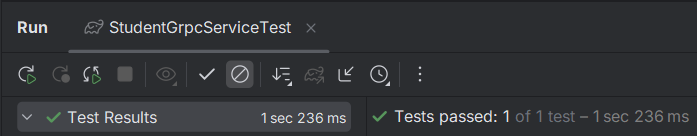


Рисунок 19 – Результат теста извлечения студента по mn-коду через gRPC

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы была разработана информационная система для автоматизированной оценки знаний студентов при выполнении практических заданий по информатике. Созданный программный продукт основан на современной клиент-серверной архитектуре с использованием технологий Spring Boot на серверной стороне и JavaFX на клиентской, что обеспечило гибкость, масштабируемость и удобство взаимодействия пользователей с системой.

Особое внимание было уделено безопасности и надёжности системы, реализованной через JWT-аутентификацию и интеграцию с Spring Security, что позволило обеспечить защиту данных и контроль доступа на уровне каждого запроса. Для эффективного взаимодействия между клиентом и сервером использован протокол gRPC с компактной и строго типизированной передачей данных посредством Protocol Buffers, что существенно повысило производительность и надёжность обмена информацией.

Ключевым элементом системы стала разработка подсистемы автоматической генерации и проверки заданий, построенной на паттерне «Фабрика». Эта архитектура позволила создавать индивидуальные задания с учётом уникальных параметров студентов, таких как mn-код, и обеспечивать объективную проверку ответов. Гибкая система фабрик генераторов и валидаторов гарантирует лёгкое расширение функционала и адаптацию под новые типы заданий без значительных изменений в кодовой базе.

Кроме того, была реализована комплексная работа с базой данных RedDataBase 3.0 через Spring Data JPA, что обеспечило надёжное хранение, управление и целостность данных. Использование асинхронной обработки запросов на клиенте повысило отзывчивость пользовательского интерфейса.

В завершение стоит отметить, что созданная система отвечает современным требованиям цифровизации образовательного процесса, обеспечивая удобные инструменты для преподавателей и студентов, повышая качество контроля знаний и автоматизируя рутинные операции. Реализация всех ключевых компонентов в рамках одного интегрированного решения демонстрирует успешное применение передовых технологий и паттернов проектирования в области образовательного программного обеспечения.

Таким образом, поставленные цели и задачи проекта были полностью достигнуты, а полученный программный продукт готов к внедрению и дальнейшему развитию.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Григорьев, А. В. Java. Полное руководство / А. В. Григорьев. — Москва: Диалектика, 2019. — 800 с.
2. Шилдт, Г. Java. Библиотека профессионала / Г. Шилдт; пер. с англ. — Москва: Вильямс, 2020. — 960 с.
3. Филиппов, В. А. JavaFX 9. Создание графических приложений / В. А. Филиппов. — Санкт-Петербург: Питер, 2018. — 384 с.
4. Блох, Дж. Эффективное программирование на Java / Дж. Блох; пер. с англ. — Москва: ДМК Пресс, 2019. — 384 с.
5. Ляшенко, А. В. Spring Framework 5 для профессионалов / А. В. Ляшенко. — Москва: БХВ-Петербург, 2020. — 560 с.
6. Смирнов, И. В. Клиент-серверные приложения на Java / И. В. Смирнов. — Москва: Наука и техника, 2017. — 320 с.
7. Ковалёв, М. Ю. Разработка корпоративных приложений с Spring Boot / М. Ю. Ковалёв. — Санкт-Петербург: Питер, 2021. — 400 с.
8. Петров, С. В. gRPC в Java: эффективное взаимодействие между сервисами / С. В. Петров. — Москва: Лань, 2022. — 256 с.
9. Иванов, А. А. Проектирование баз данных / А. А. Иванов. — Москва: Юрайт, 2018. — 320 с.
10. Соколов, В. В. Firebird. Полное руководство / В. В. Соколов. — Москва: ДМК Пресс, 2016. — 400 с.
11. Сидоров, П. Н. Информационные системы: основы проектирования / П. Н. Сидоров. — Москва: КНОРУС, 2019. — 288 с.
12. Кузнецов, Д. В. Архитектура клиент-серверных приложений / Д. В. Кузнецов. — Санкт-Петербург: Питер, 2017. — 256 с.
13. Захаров, Е. А. Основы программирования на Java / Е. А. Захаров. — Москва: БХВ-Петербург, 2018. — 350 с.
14. Баранов, В. А. Объектно-ориентированное программирование на Java / В. А. Баранов. — Москва: Инфра-М, 2020. — 480 с.
15. Логинов, С. И. Методы и средства разработки информационных систем / С. И. Логинов. — Москва: Юрайт, 2021. — 312 с.
16. Freeman, S. Pro JavaFX 9: A Definitive Guide to Building Desktop, Mobile, and Embedded Java Clients / S. Freeman. — Berkeley: Apress, 2017. — 650 p.
17. Ричардсон К. Паттерны микросервисов: с примерами на Java / пер. с англ. — Санкт-Петербург: Питер, 2019. — 544 с.
18. RedDatabase : руководство администратора : электрон. дан. / RedDatabase Team. – 3.0.19. – [Сайт]. – URL: https://reddatabase.ru/ru/pdfjs/?file=/media/documentation/RedDatabase-3.0.19-Admin\_Guide-ru.pdf (дата обращения: 09.06.2025). – Текст : электронный