МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Вятский государственный университет»**

**Колледж ВятГУ**

**КУРСОВОЙ ПРОЕКТ**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| По междисциплинарному курсу | | | | | МДК 05.01 Проектирование и дизайн | | | |
| информационных систем | | | |
| Тема | «Информационная система для управления тестами и результатами» | | | | | | | |
|  | | | | | | | |
| Студента (ки) | | | | Юдинцева Богдана Сергеевича | | | | |
|  | | | *ФИО (полностью в родительном падеже)* | | | | | |
| Курс | | 3 | | | Форма обучения | очная | |
|  | | *(арабской цифрой)* | | |  | | *(очная, заочная)* | |
| Основная профессиональная образовательная программа по специальности | | | | | | | | |
| 09.02.07 Информационные системы и программирование | | | | | | | | |
| (код и наименование специальности без кавычек) | | | | | | | | |

Руководитель курсового проекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (фамилия, инициалы)

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(прописью, без сокращений)*

Киров, 2023

**Реферат**

Пояснительная записка к курсовому проекту содержит: 30 страницы, 9 таблиц, 10 использованных источников, 21 рисунков, 2 приложения.

Объектом и предметом исследования является система автоматизированного тестирования.

Цель работы – ознакомление с процессом создания технического задания на разработку информационной системы.

Поставлена задача разработать техническое задание для информационной системы выбранного объекта исследования.

В процессе работы были проведены следующие исследования: 1) обзор предметной области, обзор аналогов и сравнительный анализ; 2) написание самого технического задания, состоящего из: назначения разработки, функциональных характеристик, требований к надёжности, условий эксплуатации и требований к составу и параметрам технических средств; 3) описание решения и концепции; 4) архитектура решения; 5) разработка схем бизнес-процессов с их описанием; 6) разработка схем алгоритмов и кода на естественном языке; 7) проектирование прототипа пользовательского интерфейса с описанием.

Элементами научного новшества полученных результатов является автоматизация тестирования.

Областью возможного практического применения являются: образовательные организации, тестирование обыденных пользователей.

Технико-экономическая и социальная значимость: внедрение систем адаптивного тестирования.

**СОДЕРЖАНИЕ**

**Изм.**

**Лист**

**№ докум.**

**Подпись**

**Дата**

**Лист**

**2**

**ТПЖА 09.02.07 443767 ПЗ**

**Разраб.**

**Провер.**

**Реценз.**

**Н. Контр.**

**Утверд.**

**Тема курсовой работы**

**Лит.**

**Листов**

**29**

**Колледж ВятГУ**

**группа**

[ВВЕДЕНИЕ 4](#_Toc154381881)

[1. ПРЕДМПЕТНАЯ ОБЛАСТЬ 5](#_Toc154381882)

[1.1. Анализ предметной области 5](#_Toc154381883)

[1.2. Обзор аналогов 5](#_Toc154381884)

[2. ТЕХНИЧЕСКОЕ ЗАДАНИЕ 8](#_Toc154381885)

[3. РЕАЛИЗАЦИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ 9](#_Toc154381886)

[3.1. Моделирование ИС 9](#_Toc154381887)

[3.1.1. Функциональное моделирование IDEF0 9](#_Toc154381888)

[3.1.2. Моделирование потоков данных DFD 11](#_Toc154381889)

[3.1.3. Моделирование в нотации UML 12](#_Toc154381890)

[3.2. Разработка интерфейса 14](#_Toc154381891)

[3.3. Разработка базы данных 21](#_Toc154381892)

[3.3.1. Описание сущностей и атрибутов 21](#_Toc154381893)

[3.3.2. Логическая модель данных в нотации IDEF1X 24](#_Toc154381894)

[3.3.3. Физическая модель данных 26](#_Toc154381895)

[ЗАКЛЮЧЕНИЕ 27](#_Toc154381896)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 28](#_Toc154381897)

[ПРИЛОЖЕНИЕ А (ТЕХНИЧЕСКОЕ ЗАДАНИЕ) 29](#_Toc154381898)

[ПРИЛОЖЕНИЕ Б (ИСХОДНЫЙ КОД) 30](#_Toc154381899)

# ВВЕДЕНИЕ

**Изм.**

**Лист**

**№ докум.**

**Подпись**

**Дата**

**Лист**

**3**

**ТПЖА 09.02.07 443767 ПЗ**

Цель данной курсовой работы - ознакомление с процессом создания технического задания на разработку информационной системы для автоматизированного тестирования. Основная задача заключается в разработке технического задания для информационной системы, специализирующейся на управлении тестами и результатами. В рамках курсовой работы осуществляется несколько ключевых этапов:

* Обзор предметной области и анализ существующих аналогов информационных систем для автоматизированного тестирования, включая их функциональные возможности и области применения.
* Разработка технического задания, включающего назначение разработки, функциональные характеристики, требования к надежности, условиям эксплуатации, а также к составу и параметрам технических средств.
* Описание предлагаемого решения и концепции разрабатываемой системы.
* Проектирование архитектуры системы и разработка схем бизнес-процессов.
* Разработка схем алгоритмов и кода на естественном языке.
* Проектирование прототипа пользовательского интерфейса, учитывающего потребности пользователей и требования к удобству использования.

Основное научное новшество данной работы заключается в разработке автоматизированной системы тестирования, которая может быть применена в образовательных организациях и для тестирования обыденных пользователей. Такая система может значительно повысить эффективность и качество процессов оценки знаний и навыков, что имеет важное технико-экономическое и социальное значение. Внедрение систем адаптивного тестирования, разрабатываемой в рамках данной курсовой работы, позволит образовательным учреждениям и другим организациям оптимизировать процесс оценки и повысить его точность и объективность.

# ПРЕДМЕТНАЯ ОБЛАСТЬ

**Изм.**

**Лист**

**№ докум.**

**Подпись**

**Дата**

**Лист**

**4**

**ТПЖА 09.02.07 443767 ПЗ**

## **Анализ предметной области**

Здесь описана предметная область «Информационной системы для управления тестами и результатами».

1. Создание и редактирование тестов;

2. Управление пользователями и группами;

3. Распределение тестов;

4. Сбор и анализ результатов.

## **Обзор аналогов**

Moodle — это бесплатная образовательная платформа, предоставляющая возможность создания персонализированных учебных курсов.

Функционал: Создание курсов, тестов, управление обучением, отслеживание успеваемости, форумы, чаты.

Применение: широко используется в учебных заведениях и корпоративном обучении.

Достоинства: Большое сообщество, множество плагинов, гибкость настройки.

Недостатки: Высокий порог входа для новых пользователей, необходимость хостинга и технического обслуживания.

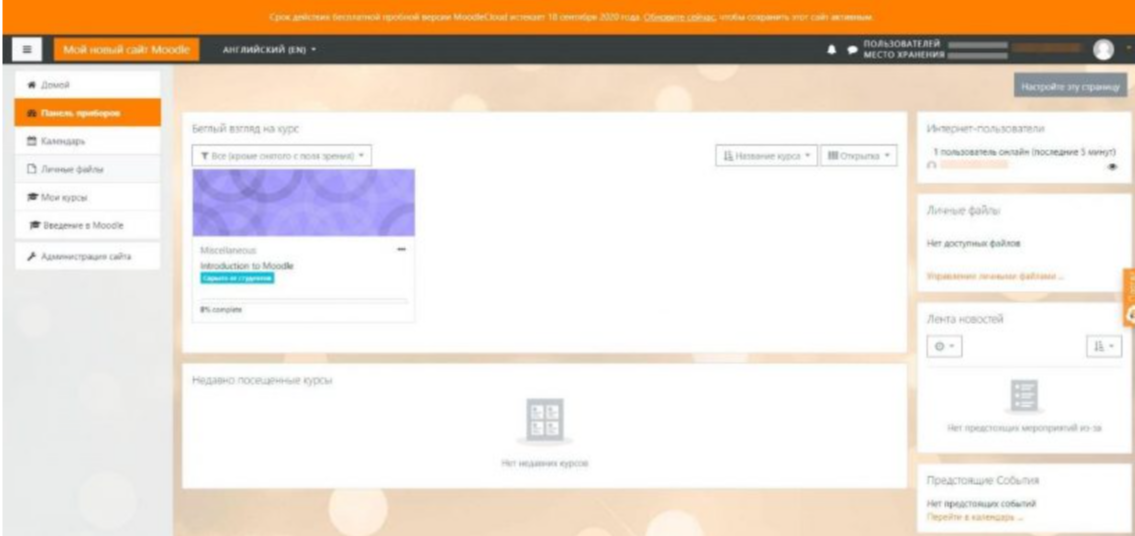


Рисунок 1. – онлайн сервис Moodle

Blackboard — коммерческая система для дистанционного обучения, используемая во многих университетах мира.

Функционал: Управление курсами, организация виртуальных классов, интеграция с различными инструментами и сервисами.

Применение: Образовательные учреждения, крупные организации для внутреннего обучения.

Достоинства: Мощные аналитические инструменты, высокая безопасность данных, поддержка множества образовательных стандартов.

Недостатки: Высокая стоимость лицензии, сложность в освоении и настройке.

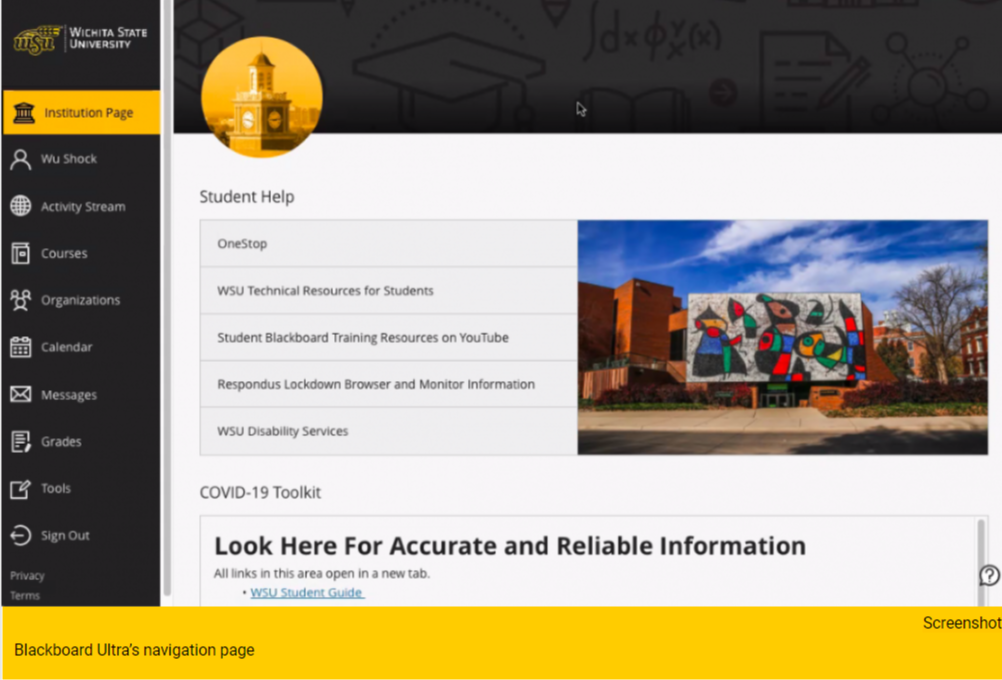


Рисунок 2. – онлайн сервис Blackboard

**Изм.**

**Лист**

**№ докум.**

**Подпись**

**Дата**

**Лист**

**5**

**ТПЖА 09.02.07 443767 ПЗ**

Google Forms — инструмент от Google, позволяющий создавать формы для опросов и тестов.

Функционал: Простое создание форм, автоматическая организация данных, интеграция с Google Таблицами.

Применение: Школы, бизнес, исследования, быстрые опросы и тесты.

Достоинства: Простота использования, бесплатность, хорошая интеграция с другими продуктами Google.

Недостатки: Ограниченные возможности кастомизации, отсутствие сложной логики вопросов.

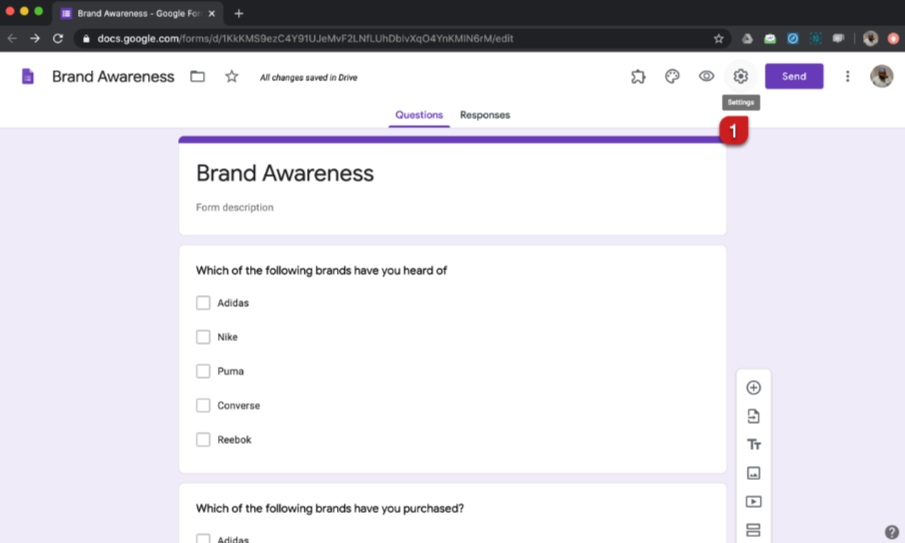


Рисунок 3. – онлайн сервис Google Forms

**Изм.**

**Лист**

**№ докум.**

**Подпись**

**Дата**

**Лист**

**6**

**ТПЖА 09.02.07 443767 ПЗ**

# ТЕХНИЧЕСКОЕ ЗАДАНИЕ

**Изм.**

**Лист**

**№ докум.**

**Подпись**

**Дата**

**Лист**

**7**

**ТПЖА 09.02.07 443767 ПЗ**

Настоящий документ представляет собой Техническое задание для курсовой работы, целью которой является разработка информационной системы для управления тестами и результатами. Данный документ является основополагающим и определяет ключевые требования к разрабатываемому программному продукту, включая его основные функции и характеристики. Техническое задание включает в себя детальное описание целей и задач разработки, предметной области, а также проведённый анализ аналогов. Оно также содержит подробные требования к системе, её интерфейсу и техническому обеспечению.

В документе уделено особое внимание разработке удобного и функционального приложения, предназначенного для применения в образовательных учреждениях и HR-отделах компаний. Этот аспект подробно освещен в разделах, посвященных описанию программы, её назначению и анализу предметной области. Включение информации о согласовании проекта с руководителями и преподавателями подчеркивает официальный статус и надежность документа.

Документ структурирован таким образом, что четко определяет стадии и этапы разработки, указывая конкретные шаги и сроки их выполнения, что способствует эффективной организации процесса разработки. Техническое задание базируется на требованиях стандартов ГОСТ, включая ГОСТ 19.201–78, ГОСТ 34.602–2020 и iso-iec-ieee-29148-2011, что гарантирует соответствие всех нормативных требований.

Раздел Технического задания и соответствующие приложения представлены в Приложении А к данной курсовой работе.

# РЕАЛИЗАЦИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ

**Изм.**

**Лист**

**№ докум.**

**Подпись**

**Дата**

**Лист**

**8**

**ТПЖА 09.02.07 443767 ПЗ**

## **Моделирование ИС**

* + 1. **Функциональное моделирование IDEF0**

Модель IDEF0 состоит из нескольких ключевых компонентов, включая входы, выходы, механизмы и управление. Давайте подробнее рассмотрим каждый из них.

Входы (Inputs):

* Запросы от пользователей: Этот важный входной элемент включает в себя запросы, поступающие от разных категорий пользователей, таких как администраторы, учителя и студенты.
* Тестовые данные: В данной системе неотъемлемой частью являются тестовые данные, которые содержат вопросы и необходимый материал для проведения тестов.
* Пользовательские данные: Этот вход включает информацию о пользователях системы, таких как студенты, учителя и администраторы.

Выходы (Outputs):

* Результаты тестов: Один из ключевых выходов системы — это получение результатов тестов, включая оценки и анализ успеваемости студентов.
* Управленческие отчеты: для администраторов и учителей генерируются управленческие отчеты, которые содержат информацию о прогрессе и успеваемости группы студентов.
* Учетные записи и группы: Система также предоставляет информацию о пользователях, списке групп и назначенных тестах.

Механизмы (Mechanisms):

* Система управления базой данных: Важным механизмом этого блока является система управления базой данных.
* Интерфейс пользователя: для взаимодействия пользователей с системой предоставляется веб-интерфейс или приложение. Управление (Controls):
* Правила и политики системы: для эффективного функционирования системы разрабатываются и применяются набор правил и политик.
* Безопасность и доступ: Механизмы аутентификации и авторизации используются для обеспечения безопасности системы и разграничения доступа пользователей к разным функциональным элементам.

Все эти компоненты взаимодействуют внутри основного блока "Управление Тестами и Результатами" (рис. 4), обеспечивая эффективное управление образовательными процессами и обработку результатов тестирования в данной системе.



**Изм.**

**Лист**

**№ докум.**

**Подпись**

**Дата**

**Лист**

**9**

**ТПЖА 09.02.07 443767 ПЗ**

Рисунок 4. – Функциональная модель IDEF0

* + 1. **Моделирование потоков данных DFD**

**Изм.**

**Лист**

**№ докум.**

**Подпись**

**Дата**

**Лист**

**10**

**ТПЖА 09.02.07 443767 ПЗ**

На диаграмме DFD (рис. 5), показаны ключевые процессы информационной системы, внешние акторы и потоки данных между ними.

Основные элементы диаграммы включают:

Внешние акторы: Администратор, Учитель, Студент, которые взаимодействуют с системой.

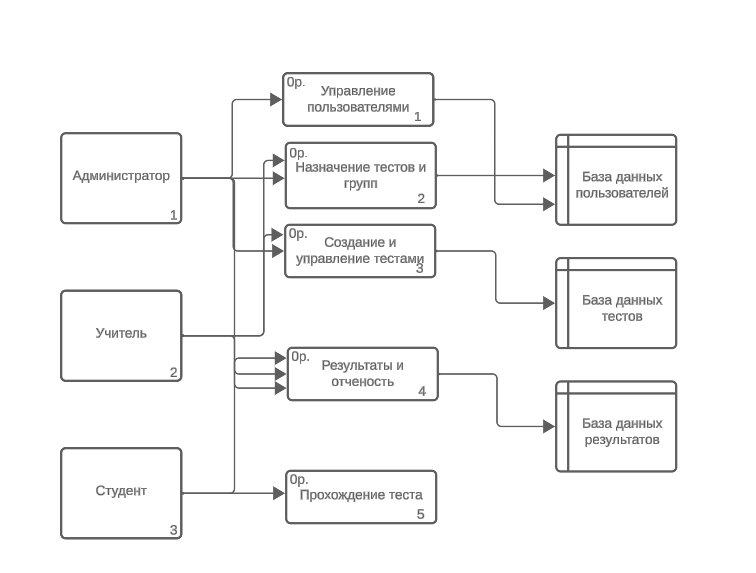
Процессы:

* Управление пользователями — включает в себя создание и управление учетными записями.
* Назначение тестов и групп — отвечает за распределение тестов между группами и отдельными студентами.
* Создание и управление тестами — позволяет разрабатывать тесты и управлять ими.
* Результаты и отчетность — процесс анализа результатов тестов и генерация отчетов.
* Прохождение теста — процесс, в ходе которого студенты выполняют тесты.

Хранилища данных:

* База данных пользователей — содержит информацию обо всех пользователях системы.
* База данных тестов — хранит тесты и связанные с ними данные.
* База данных результатов — архивирует результаты прохождения тестов.

Потоки данных на диаграмме иллюстрируют, как информация перемещается от внешних акторов к процессам и хранилищам данных, а также между самими процессами.

****

**Изм.**

**Лист**

**№ докум.**

**Подпись**

**Дата**

**Лист**

**11**

**ТПЖА 09.02.07 443767 ПЗ**

Рисунок 5. – Модель потоков данных DFD

* + 1. **Моделирование в нотации UML**

Диаграмма UML (рис. 6) демонстрирует различные аспекты системы через взаимодействие акторов и случаи использования. Элементы диаграммы включают:

Акторы: Администратор, Учитель, Студент — пользователи системы, взаимодействующие с различными функциями.

Случаи использования:

* Управление пользователями — администратор может создавать и управлять учетными записями.
* Назначение тестов и групп — учитель распределяет тесты между студентами и группами.
* Создание и управление тестами — учитель разрабатывает и модифицирует тесты.
* Просмотр результатов и отчетности — учитель и администратор анализируют данные о прохождении тестов.
* Прохождение тестов — студенты выполняют тесты и получают результаты.

Эти случаи использования показывают функциональные возможности системы и то, как акторы взаимодействуют с системой для выполнения своих задач.

**Изм.**

**Лист**

**№ докум.**

**Подпись**

**Дата**

**Лист**

**12**

**ТПЖА 09.02.07 443767 ПЗ**

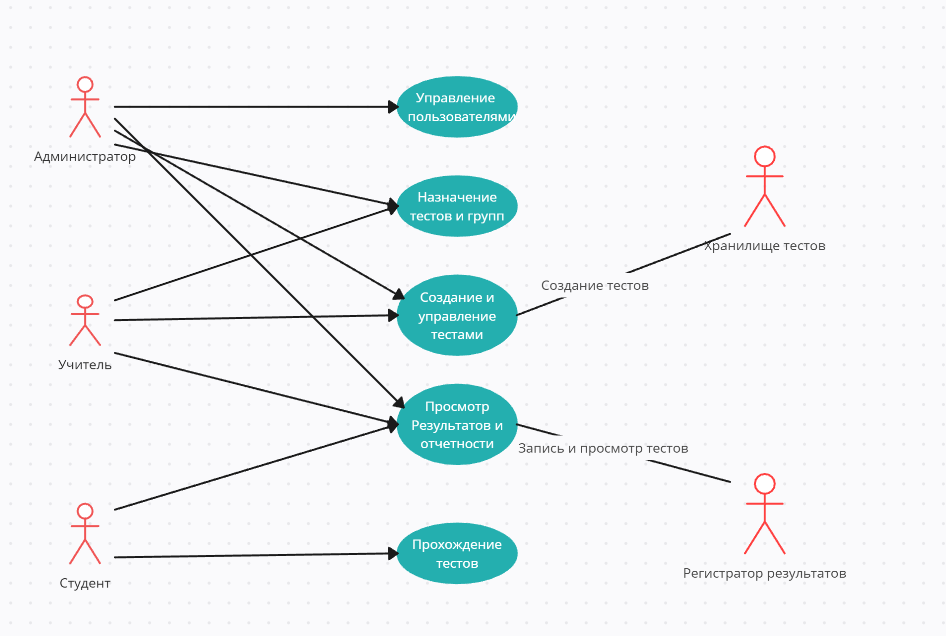
****

Рисунок 6. – Модель в нотации UML

## **Разработка интерфейса**

В этом разделе описываются структура интерфейса приложения “Информационная система для управления тестами и результатами”

**Меню авторизации**

Меню Авторизации: при запуске программы отображается экран авторизации, предлагающий пользователю войти или зарегистрироваться. (рис. 7 и 8)

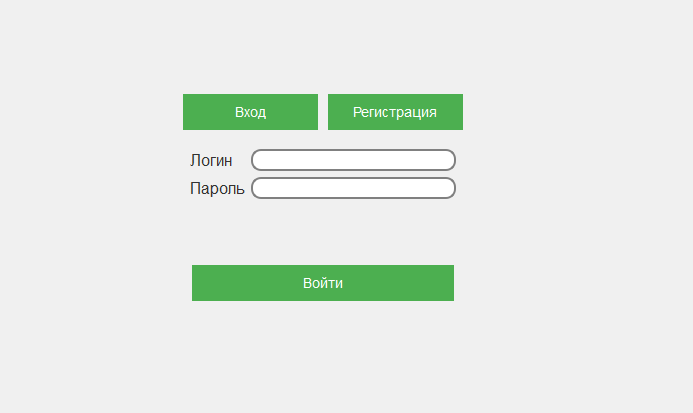


Рисунок 7. – окно «Входа».

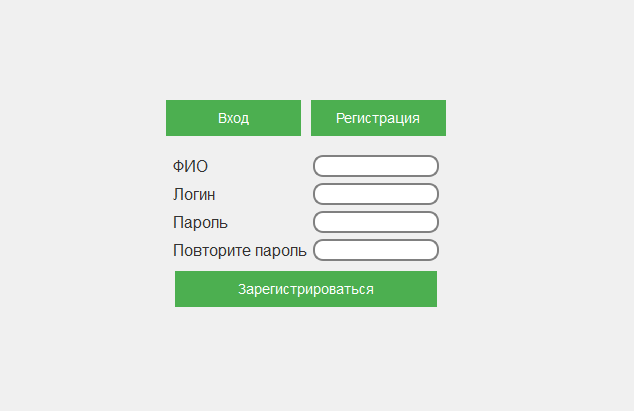


Рисунок 8. – окно «Регистрации».

**Изм.**

**Лист**

**№ докум.**

**Подпись**

**Дата**

**Лист**

**13**

**ТПЖА 09.02.07 443767 ПЗ**

Главное меню после успешной аутентификации включает в себя следующие основные функции: просмотр и редактирование профиля, настройки и выход из аккаунта. Эти опции могут быть представлены на главном экране (рисунок 9 и 10).

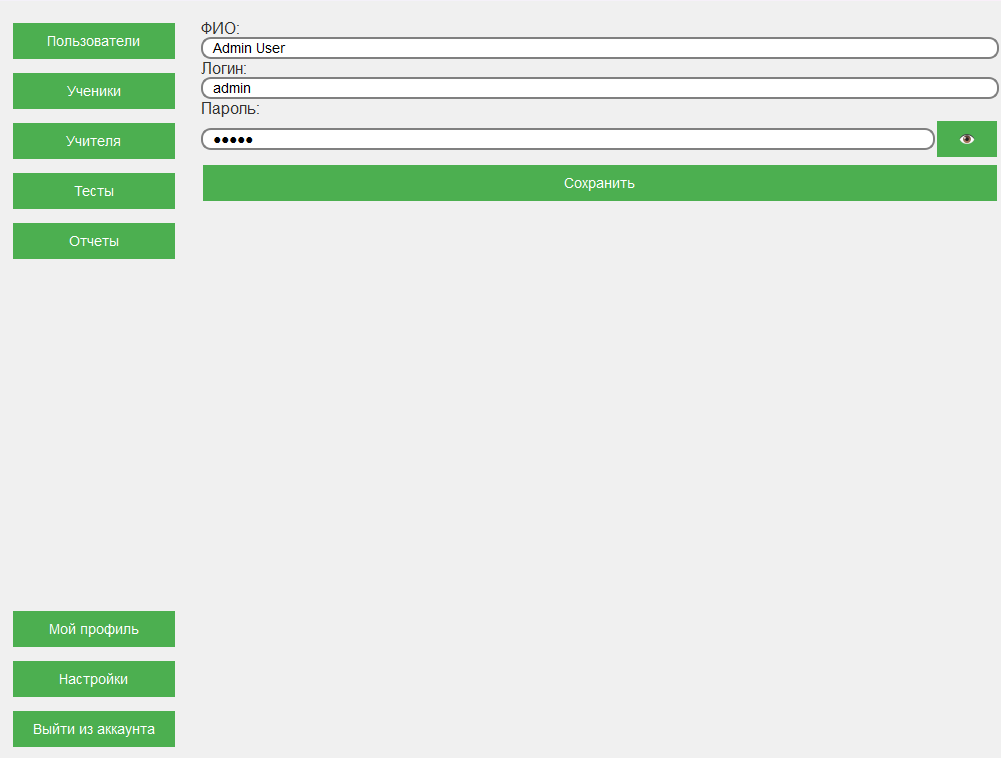


Рисунок 9. – окно «Профиля пользователя».

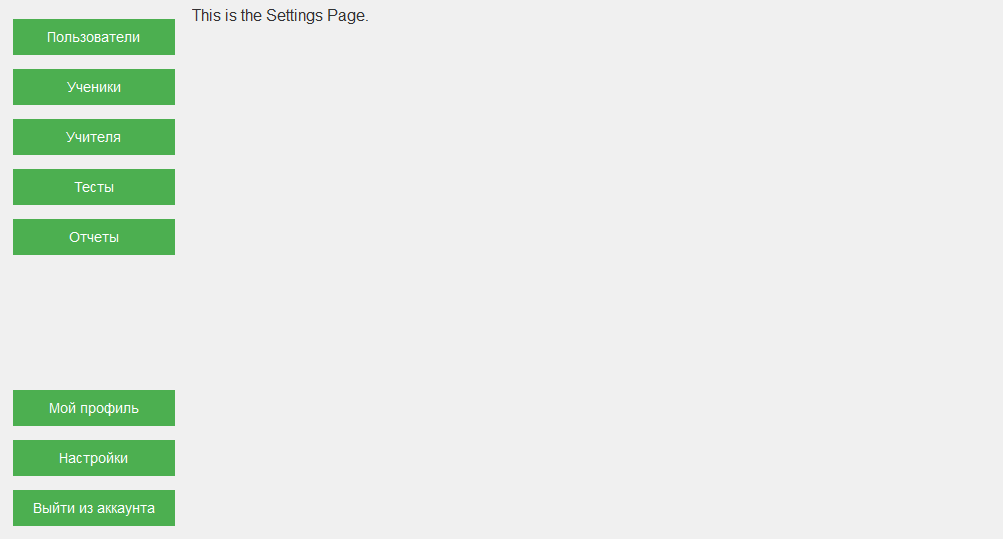


Рисунок 10. – окно «Настройки»

**Изм.**

**Лист**

**№ докум.**

**Подпись**

**Дата**

**Лист**

**14**

**ТПЖА 09.02.07 443767 ПЗ**

После успешной аутентификации, программа автоматически определяет роль пользователя и отображает соответствующее меню. Всего доступны три роли: администратор, учитель и студент.

**Меню Администратора**

Управление Пользователями: Функционал для добавления (рис. 11), удаления, редактирования пользователей и просмотра их информации.

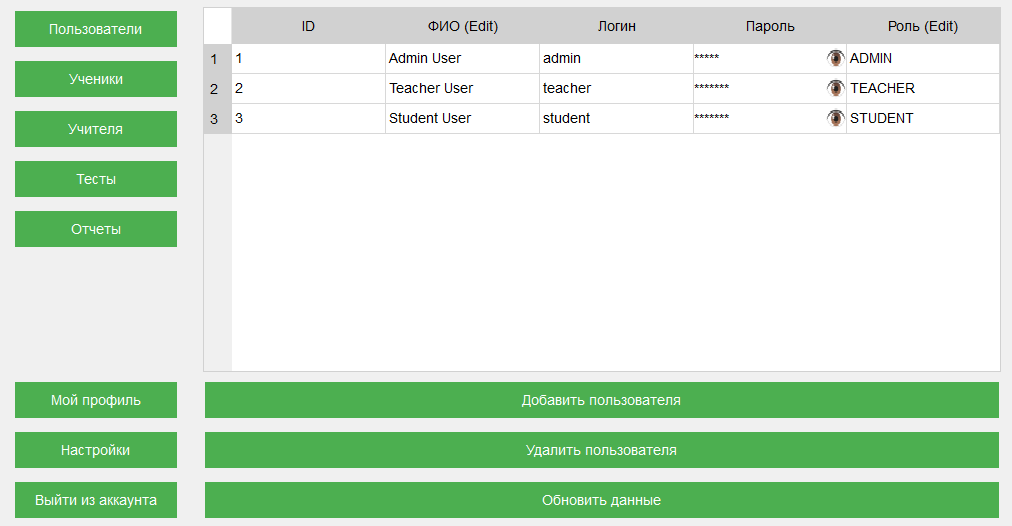


Рисунок 11. – окно «Управление пользователями».

Управление Студентами: Возможности по управлению группами студентов (рис. 12)., назначению тестов и изменению групп.

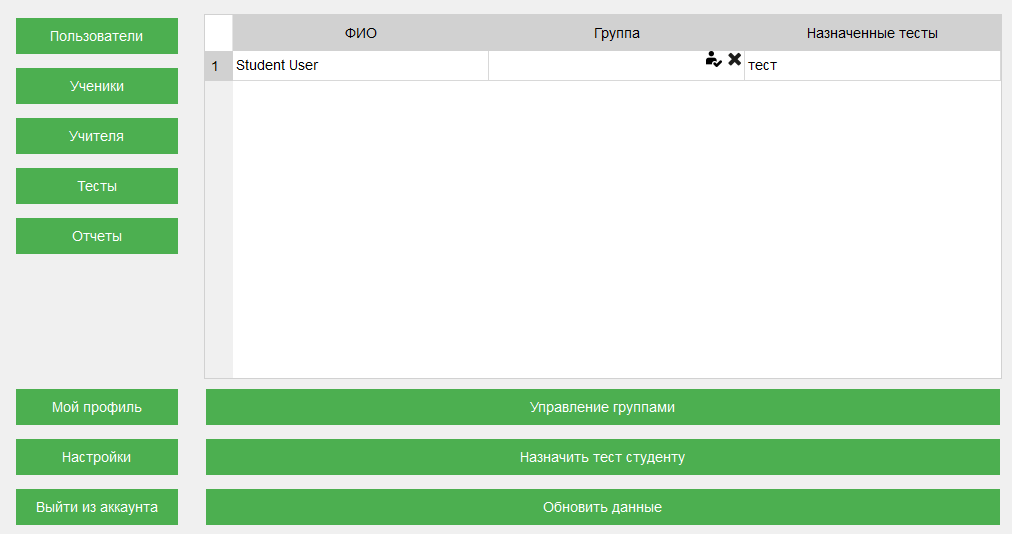


Рисунок 12. – окно «Управление студентами».

**Изм.**

**Лист**

**№ докум.**

**Подпись**

**Дата**

**Лист**

**15**

**ТПЖА 09.02.07 443767 ПЗ**

Управление Учителями: Просмотр учителей и их тестов (рис. 13).

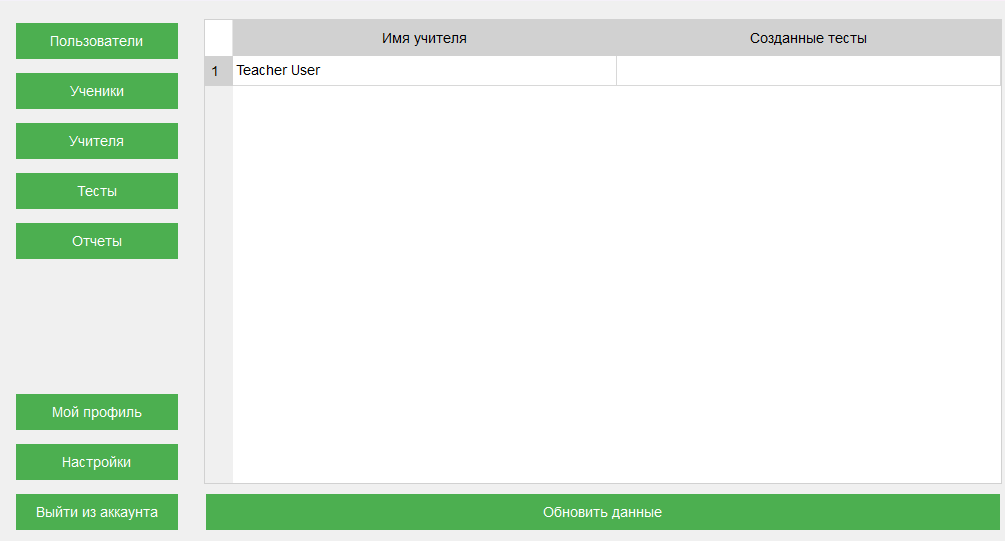


Рисунок 13. – окно «Учителя».

Управление Тестами: Просмотр, создание (рис. 14) и удаление тестов.

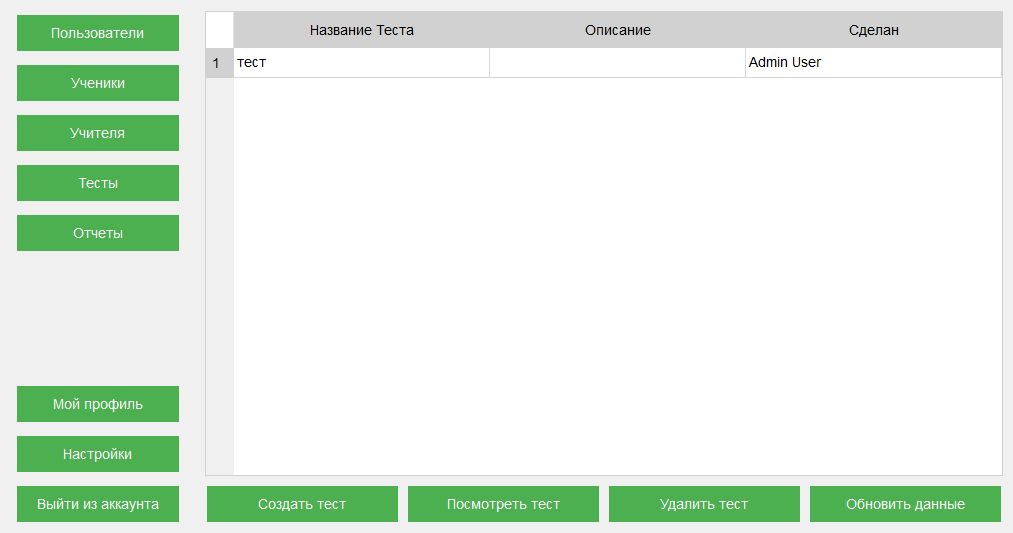


Рисунок 14. – окно «Тесты».

Отчеты: Функционал для выбора, генерации и сохранения отчетов. (рис. 19)

**Изм.**

**Лист**

**№ докум.**

**Подпись**

**Дата**

**Лист**

**16**

**ТПЖА 09.02.07 443767 ПЗ**

**Меню Учителя**

Управление Студентами: Функции для управления группами (рис. 15) и назначения тестов студентам.

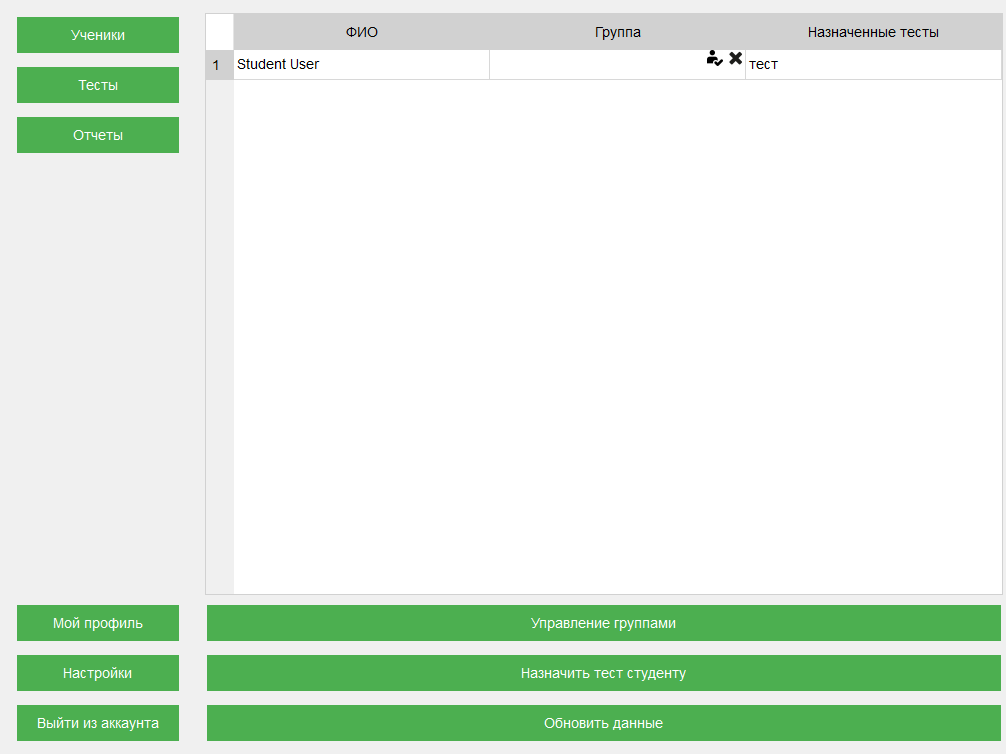


Рисунок 15. – окно «Студенты».

Управление Тестами: Возможности для просмотра, создания (рис. 16) и удаления своих тестов.

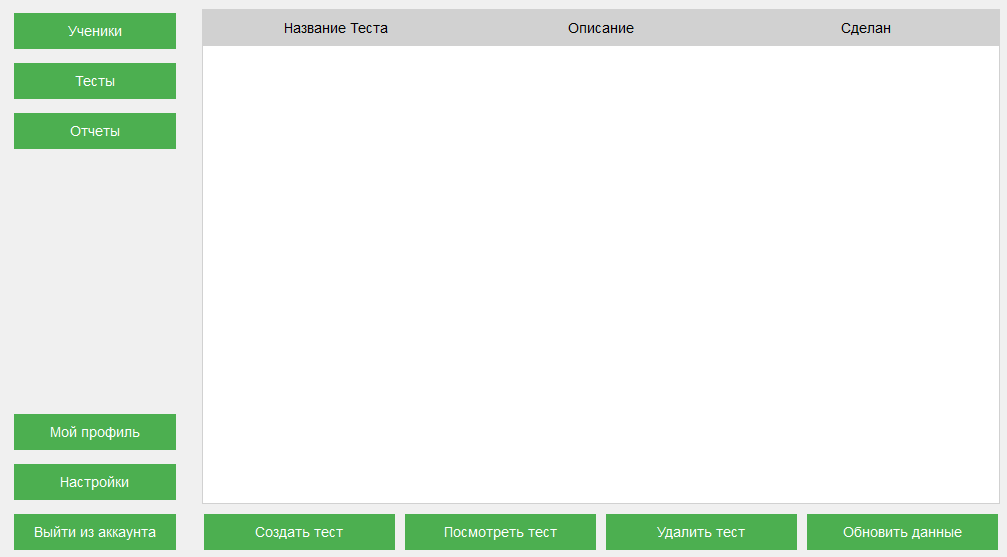


Рисунок 16. – окно «Тесты».

Отчеты: Выбор, генерация и сохранение отчетов учителями. (рис. 19)

**Изм.**

**Лист**

**№ докум.**

**Подпись**

**Дата**

**Лист**

**17**

**ТПЖА 09.02.07 443767 ПЗ**

**Меню Студента**

Мои Тесты: Просмотр доступных тестов и возможность их прохождения (рис. 17).

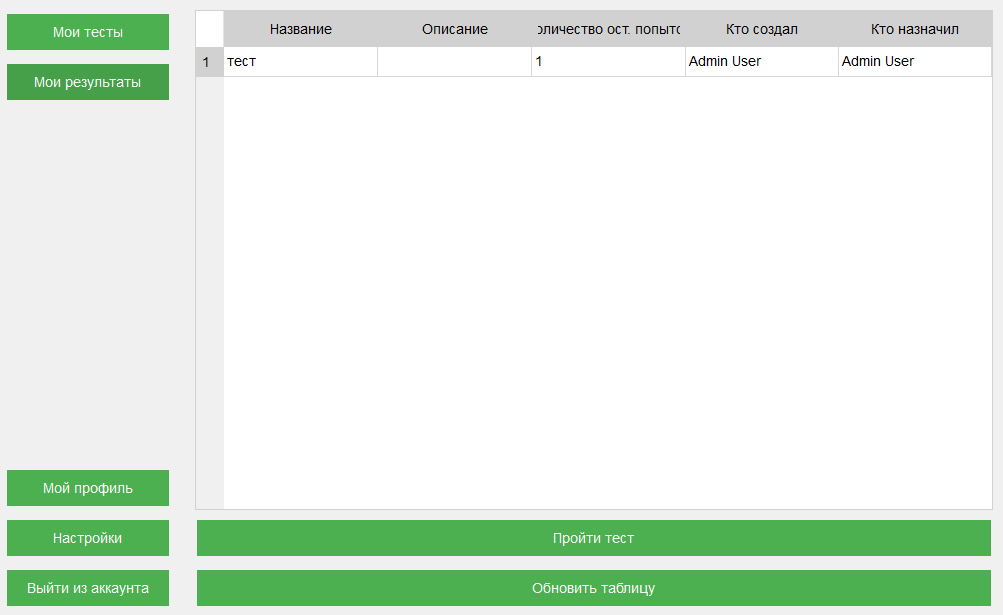


Рисунок 17. – окно «Мои тесты».

Отчеты: Функционал для выбора, генерации и сохранения студенческих отчетов. (рис. 19)

**Общие элементы Интерфейса**

Создание теста: Создание теста и запись в БД (рис. 18).

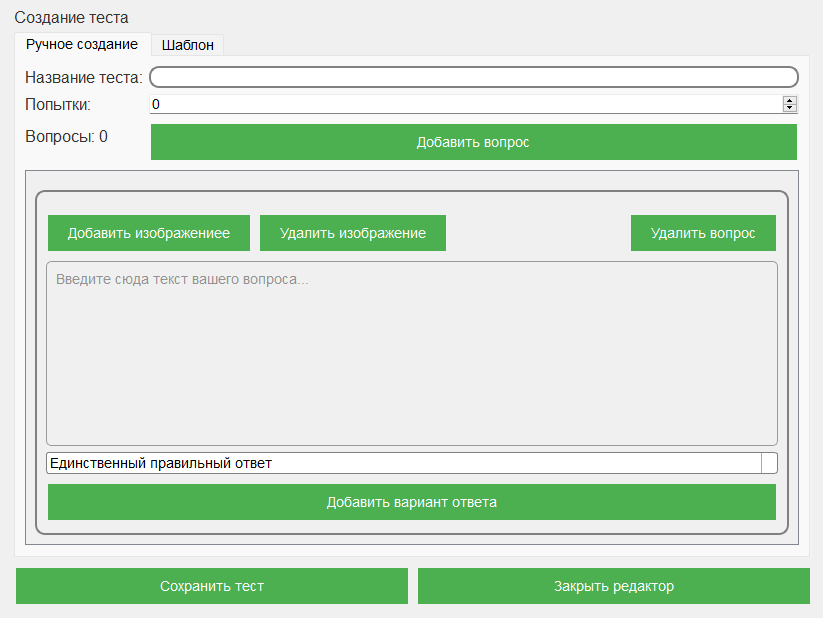
****

Рисунок 18. – окно «Создание теста».

**Изм.**

**Лист**

**№ докум.**

**Подпись**

**Дата**

**Лист**

**18**

**ТПЖА 09.02.07 443767 ПЗ**

Окно "Отчеты" предоставляет пользователю возможность использовать шаблон отчета, который отображает результаты тестов, а также позволяет сохранить их локально.

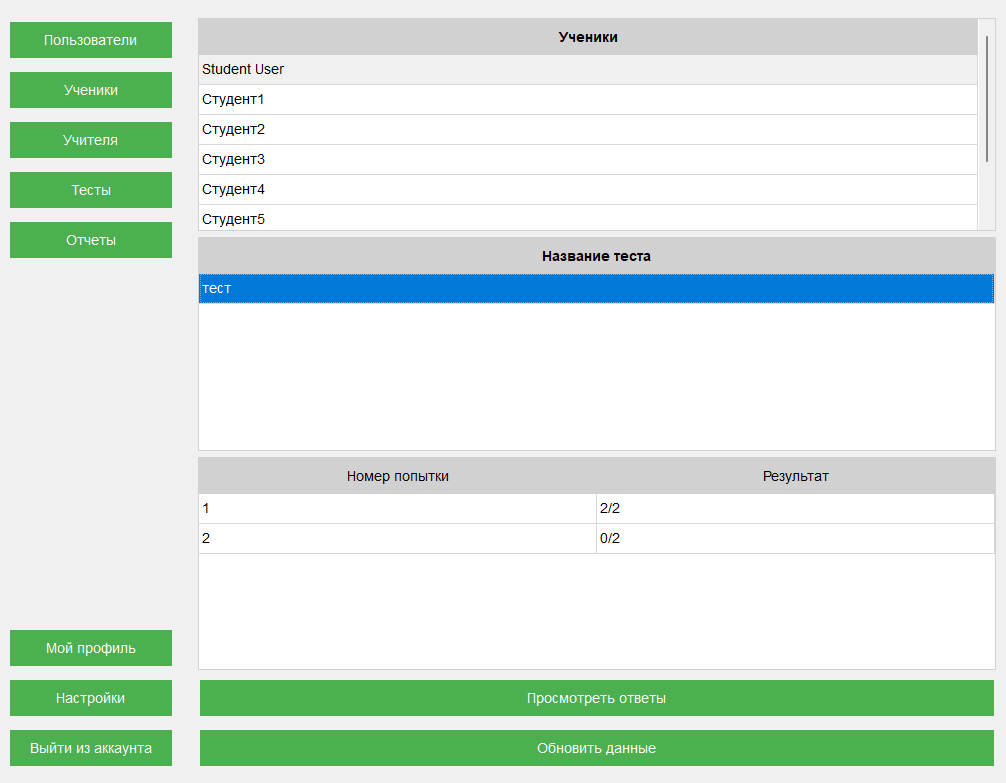


Рисунок 19. – окно «Отчеты».

**Изм.**

**Лист**

**№ докум.**

**Подпись**

**Дата**

**Лист**

**19**

**ТПЖА 09.02.07 443767 ПЗ**

## **Разработка базы данных**

**Изм.**

**Лист**

**№ докум.**

**Подпись**

**Дата**

**Лист**

**20**

**ТПЖА 09.02.07 443767 ПЗ**

### **Описание сущностей и атрибутов**

Ниже представлено описание сущностей и их атрибутов в табличном формате для созданных таблиц в базе данных:

Таблица "users":

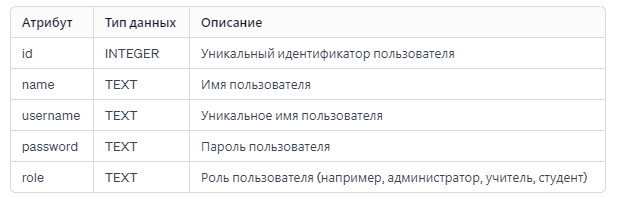


Таблица "groups":

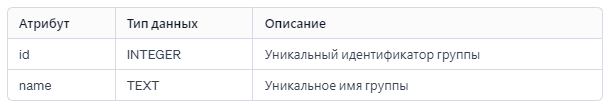


Таблица "tests":



Таблица "user\_groups":

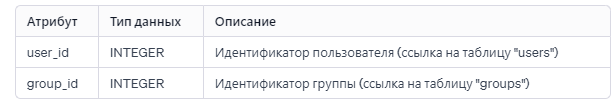


Таблица "student\_tests":

**Изм.**

**Лист**

**№ докум.**

**Подпись**

**Дата**

**Лист**

**21**

**ТПЖА 09.02.07 443767 ПЗ**

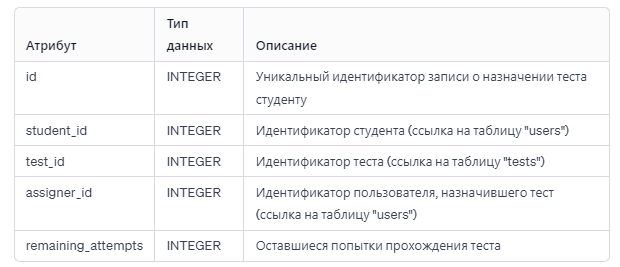


Таблица "questions":

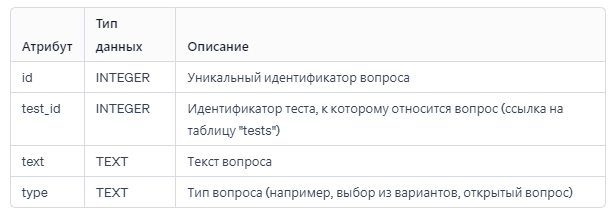


Таблица "answers":

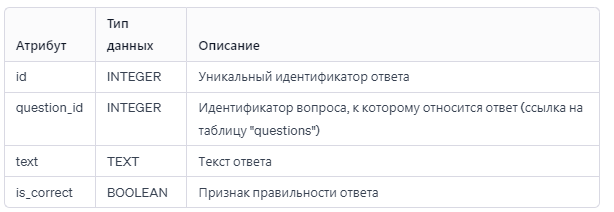


Таблица "student\_answers":

**Изм.**

**Лист**

**№ докум.**

**Подпись**

**Дата**

**Лист**

**22**

**ТПЖА 09.02.07 443767 ПЗ**

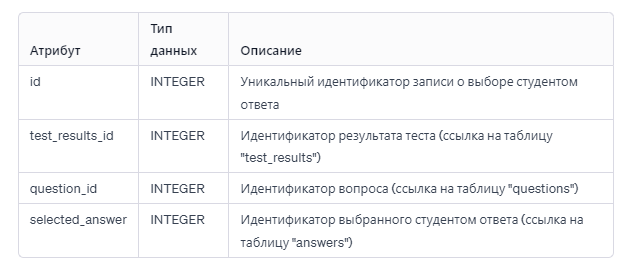
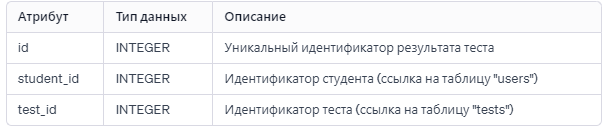


Таблица "test\_results":



### **Логическая модель данных в нотации IDEF1X**

**Изм.**

**Лист**

**№ докум.**

**Подпись**

**Дата**

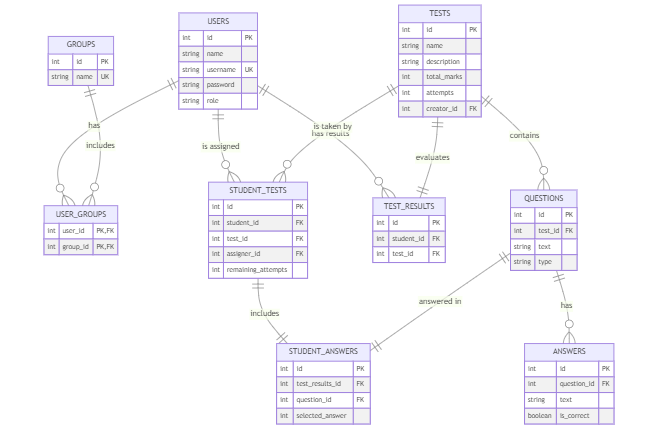
**Лист**

**23**

**ТПЖА 09.02.07 443767 ПЗ**

Логическая модель данных (рис. 20), которая описывает структуру базы данных в терминах сущностей и их связей. Это абстрактное представление, которое не зависит от конкретной технологии реализации базы данных:

* USERS: содержит информацию о пользователях системы, включая их имя, имя пользователя, пароль и роль.
* GROUPS: определяет различные группы в системе.
* USER\_GROUPS: Таблица многие-ко-многим, связывающая пользователей с группами.
* TESTS: содержит информацию о тестах, включая описание, общее количество баллов и количество попыток.
* STUDENT\_TESTS: связывает студента (пользователя) с тестом, который он проходит, и отслеживает оставшиеся попытки.
* QUESTIONS: хранит вопросы для каждого теста.
* ANSWERS: хранит возможные ответы на каждый вопрос и указывает, правильный ли каждый из них.
* TEST\_RESULTS: хранит результаты тестов, пройденных студентами.
* STUDENT\_ANSWERS: записывает выбранные студентами ответы на каждый вопрос в тесте.



**Изм.**

**Лист**

**№ докум.**

**Подпись**

**Дата**

**Лист**

**24**

**ТПЖА 09.02.07 443767 ПЗ**

Рисунок 20. – Логическая модель данных

### **Физическая модель данных**

На втором изображении представлена физическая модель данных, которая является более конкретным представлением и включает в себя определение типов данных и ограничений для системы управления базами данных:

* users, groups, tests, user\_groups, student\_tests, questions, answers, student\_answers, test\_results: Эти таблицы в физической модели соответствуют сущностям в логической модели, но включают конкретные типы данных, такие как INTEGER, TEXT, BOOLEAN, и ограничения, такие как PRIMARY KEY, FOREIGN KEY, NOT NULL и UNIQUE.

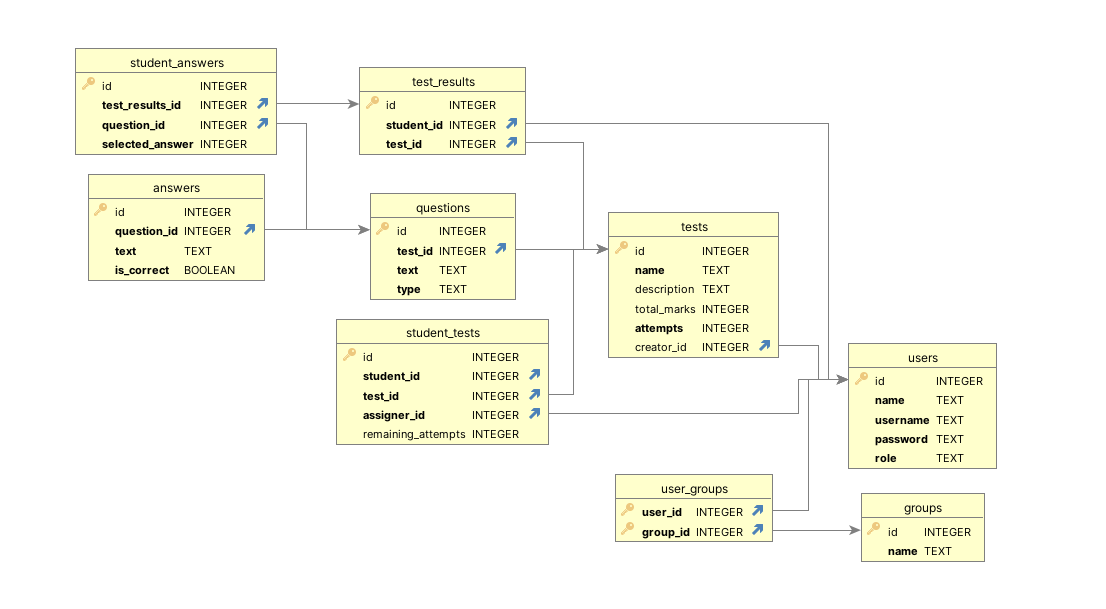


Рисунок 21. – Физическая модель данных

**Изм.**

**Лист**

**№ докум.**

**Подпись**

**Дата**

**Лист**

**25**

**ТПЖА 09.02.07 443767 ПЗ**

**Изм.**

**Лист**

**№ докум.**

**Подпись**

**Дата**

**Лист**

**3**

**ТПЖА 09.02.07 443767 ПЗ**

# ЗАКЛЮЧЕНИЕ

**Изм.**

**Лист**

**№ докум.**

**Подпись**

**Дата**

**Лист**

**26**

**ТПЖА 09.02.07 443767 ПЗ**

В рамках данной курсовой работы была осуществлена разработка и детальное изучение технического задания для информационной системы управления тестами и результатами. Работа над проектом включала в себя анализ предметной области, изучение существующих аналогов и проведение сравнительного анализа, что позволило более полно осознать специфику и требования к разрабатываемой системе.

Основное внимание было уделено формированию четких и осмысленных требований к функциональным характеристикам системы, надежности, условиям эксплуатации и техническим параметрам используемых средств. Важной составляющей работы стало моделирование информационной системы, включая функциональное моделирование IDEF0, моделирование потоков данных DFD, а также моделирование в нотации UML. Разработка интерфейса и базы данных проводилась с учетом текущих трендов и лучших практик в данной области.

В ходе курсовой работы был разработан прототип пользовательского интерфейса, предоставляющий наглядное представление будущей системы, а также выполнено проектирование схем бизнес-процессов и алгоритмов на естественном языке.

В результате проделанной работы можно сделать вывод о том, что разработанная информационная система для управления тестами и результатами способна обеспечить эффективное и удобное тестирование в образовательных учреждениях и других организациях. Система имеет высокий потенциал для адаптации и масштабирования в соответствии с изменяющимися требованиями и условиями эксплуатации. Технико-экономическая и социальная значимость проекта подтверждается его способностью вносить значительный вклад в процессы обучения и оценки квалификации, повышая эффективность и качество тестирования.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

**Изм.**

**Лист**

**№ докум.**

**Подпись**

**Дата**

**Лист**

**27**

**ТПЖА 09.02.07 443767 ПЗ**

1. Кормен, Т., Лейзерсон, Ч., Ривест, Р., и Штайн, К. "Введение в алгоритмы". - ISBN: 978-5-8459-1794-1.
2. Седжвик, Р., и Уэйн, К. "Алгоритмы: построение и анализ". - ISBN: 978-5-496-00484-6.
3. Кнут, Д. "Искусство программирования, том 1-3". - ISBN: 978-5-496-00056-5.
4. Гудрич, М., и Томас, Т. "Алгоритмы и структуры данных в Java". - ISBN: 978-5-7502-0060-8.
5. Кормен, Т., Лейзерсон, Ч., Ривест, Р., и Штайн, К. "Введение в алгоритмы: Учебное пособие". - ISBN: 978-5-907114-43-8.
6. Стивенс, Р., Бирл, Б., и Льюис, Д. "Алгоритмы: пособие для программистов и разработчиков". - ISBN: 978-5-6041104-7-8.
7. Кнут, Д. "Сортировка и поиск". - ISBN: 978-5-94588-163-7.
8. Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. "Алгоритмы. Построение и анализ". - ISBN: 978-5-8459-1794-1.
9. Стивен Скиена. "Алгоритмы: реализация и применение". - ISBN: 978-5-496-00484-6.
10. Роберт Лафоре. "Алгоритмы и структуры данных". - ISBN: 978-5-94774-174-9.

# ПРИЛОЖЕНИЕ А (ТЕХНИЧЕСКОЕ ЗАДАНИЕ)

**Изм.**

**Лист**

**№ докум.**

**Подпись**

**Дата**

**Лист**

**28**

**ТПЖА 09.02.07 443767 ПЗ**

# ПРИЛОЖЕНИЕ Б (ИСХОДНЫЙ КОД)

**Изм.**

**Лист**

**№ докум.**

**Подпись**

**Дата**

**Лист**

**29**

**ТПЖА 09.02.07 443767 ПЗ**

# interfaces\admin\admin\_interface.py

**from** PyQt5.QtWidgets **import** QWidget, QHBoxLayout, QStackedWidget

**from** interfaces.sidebar **import** SidebarMenu

**from** .users\_page **import** UsersPage

**from** ..students\_page **import** StudentsPage

**from** .teachers\_page **import** TeachersPage

**from** ..tests\_page **import** TestsPage

**from** ..reports\_page **import** ReportsWindow

**class** AdminWindow(QWidget):

**def** \_\_init\_\_(self, main\_window):

super().\_\_init\_\_()

self.main\_window = main\_window

self.initUI()

**def** initUI(self):

upper\_buttons = ["Пользователи", "Ученики",

"Учителя", "Тесты", "Отчеты"]

self.stack = QStackedWidget()

self.stack.addWidget(UsersPage())

self.stack.addWidget(StudentsPage(self.main\_window))

self.stack.addWidget(TeachersPage())

self.test\_page = TestsPage(self)

self.stack.addWidget(self.test\_page)

self.stack.addWidget(ReportsWindow(self.main\_window))

self.sidebar = SidebarMenu(upper\_buttons, self.stack, self.main\_window)

layout = QHBoxLayout()

layout.addWidget(self.sidebar)

layout.addWidget(self.stack)

self.setLayout(layout)

self.connectButtons()

**def** connectButtons(self):

self.sidebar.connectStack()

# interfaces\admin\teachers\_page.py

**from** PyQt5.QtWidgets **import** (QWidget, QVBoxLayout, QTableView, QPushButton,

QHeaderView, QDialog, QMessageBox)

**from** PyQt5.QtGui **import** QStandardItemModel, QStandardItem

**import** database

**class** TeachersPage(QWidget):

**def** \_\_init\_\_(self):

super().\_\_init\_\_()

self.initUI()

**def** initUI(self):

self.layout = QVBoxLayout(self)

self.teachersTestsModel = QStandardItemModel()

self.teachersTestsModel.setHorizontalHeaderLabels(

["Имя учителя", "Название теста"])

self.teachersTestsTable = QTableView(self)

self.teachersTestsTable.setModel(self.teachersTestsModel)

self.teachersTestsTable.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)

self.teachersTestsTable.setSelectionBehavior(QTableView.SelectRows)

self.teachersTestsTable.setSelectionMode(QTableView.SingleSelection)

self.teachersTestsTable.setEditTriggers(QTableView.NoEditTriggers)

self.layout.addWidget(self.teachersTestsTable)

self.refreshButton = QPushButton("Обновить данные", self)

self.refreshButton.clicked.connect(self.refresh\_data)

self.layout.addWidget(self.refreshButton)

self.refresh\_data()

**def** refresh\_data(self):

teachers\_tests\_data = database.get\_teachers\_tests()

self.teachersTestsModel.clear()

self.teachersTestsModel.setHorizontalHeaderLabels(

["Имя учителя", "Созданные тесты"])

**for** data **in** teachers\_tests\_data:

teacher\_name = data["teacher\_name"]

tests = data["tests"]

teacher\_item = QStandardItem(teacher\_name)

tests\_item = QStandardItem(tests)

self.teachersTestsModel.appendRow([teacher\_item, tests\_item])

# interfaces\admin\users\_page.py

**from** PyQt5.QtWidgets **import** (QWidget, QPushButton, QTableView, QVBoxLayout, QMessageBox, QDialog,

QLineEdit, QHBoxLayout, QFormLayout, QComboBox, QStyledItemDelegate, QTextEdit, QHeaderView)

**from** PyQt5.QtCore **import** Qt, QTimer, QEvent

**from** PyQt5.QtGui **import** QIcon, QStandardItem, QStandardItemModel

**import** database

**class** CustomStandardItemModel(QStandardItemModel):

**def** \_\_init\_\_(self, rows, columns):

super().\_\_init\_\_(rows, columns)

**def** flags(self, index):

flags = super().flags(index)

**if** index.column() **not** **in** [1, 4]:

flags &= ~Qt.ItemIsEditable

**return** flags

**class** AddUserDialog(QDialog):

**def** \_\_init\_\_(self, parent=None):

super().\_\_init\_\_(parent)

self.setWindowTitle("�"обавить нового пользователя"***)***

self.layout = QFormLayout(self***)***

self.name = QLineEdit(self***)***

self.username = QLineEdit(self***)***

self.password = QLineEdit(self***)***

self.role = QComboBox(self***)***

self.role.addItems(["ADMIN", "TEACHER", "STUDENT"]***)***

self.layout.addRow("ФИО:", self.name***)***

self.layout.addRow("Логин:", self.username***)***

self.layout.addRow("Пароль:", self.password***)***

self.layout.addRow("Роль:", self.role***)***

self.buttons = QHBoxLayout(***)***

self.addButton = QPushButton("�"обавить", self)

self.addButton.clicked.connect(self.accept)

self.cancelButton = QPushButton("Отмена", self)

self.cancelButton.clicked.connect(self.reject)

self.buttons.addWidget(self.addButton)

self.buttons.addWidget(self.cancelButton)

self.layout.addRow(self.buttons)

**def** get\_inputs(self):

**return** {

"name": self.name.text(),

"username": self.username.text(),

"password": self.password.text(),

"role": self.role.currentText(),

}

**class** CustomDelegate(QStyledItemDelegate):

**def** \_\_init\_\_(self, parent=None):

super().\_\_init\_\_(parent)

self.button\_icon = QIcon("resources/icons/eye-icon.png")

self.show\_password = {}

self.timer = QTimer()

self.timer.timeout.connect(self.hide\_password)

self.roles = ["ADMIN", "STUDENT", "TEACHER"]

**def** paint(self, painter, option, index):

super().paint(painter, option, index)

**if** index.column() == 3:

password = index.data(Qt.DisplayRole)

show = self.show\_password.get(index, False)

painter.eraseRect(option.rect)

**if** password **is** **not** None:

**if** show:

painter.drawText(option.rect, Qt.AlignVCenter, password)

**else**:

masked\_password = "\*" \* len(password)

painter.drawText(

option.rect, Qt.AlignVCenter, masked\_password)

button\_rect = option.rect.adjusted(

option.rect.width() - 20, 0, 0, 0)

painter.drawPixmap(button\_rect, self.button\_icon.pixmap(20, 20))

**def** editorEvent(self, event, model, option, index):

**if** index.column() == 3 **and** event.type() == QEvent.MouseButtonRelease:

button\_rect = option.rect.adjusted(

option.rect.width() - 20, 0, 0, 0)

**if** button\_rect.contains(event.pos()):

self.show\_password[index] = **not** self.show\_password.get(

index, False)

model.dataChanged.emit(index, index)

**if** self.show\_password[index]:

self.timer.start(3000)

**else**:

self.timer.stop()

**return** True

**return** super().editorEvent(event, model, option, index)

**def** hide\_password(self):

**for** index **in** self.show\_password.keys():

self.show\_password[index] = False

index.model().dataChanged.emit(index, index)

**def** createEditor(self, parent, option, index):

**if** index.column() == 1:

editor = QTextEdit(parent)

**return** editor

**elif** index.column() == 4:

editor = QComboBox(parent)

editor.addItems(self.roles)

**return** editor

**else**:

**return** super().createEditor(parent, option, index)

**def** setEditorData(self, editor, index):

**if** index.column() == 1:

editor.setPlainText(index.data())

**elif** index.column() == 4:

editor.setCurrentText(index.data())

**else**:

super().setEditorData(editor, index)

**def** setModelData(self, editor, model, index):

**if** index.column() == 1:

new\_name = editor.toPlainText()

old\_name = index.data()

**if** new\_name != old\_name:

model.setData(index, new\_name)

username\_index = model.index(index.row(), 2)

username = model.data(username\_index)

database.update\_name(username, new\_name)

**elif** index.column() == 4:

new\_role = editor.currentText()

old\_role = index.data()

**if** new\_role != old\_role:

model.setData(index, new\_role)

username\_index = model.index(index.row(), 2)

username = model.data(username\_index)

database.update\_role(username, new\_role)

model.dataChanged.emit(index, index)

**class** UsersPage(QWidget):

**def** \_\_init\_\_(self):

super().\_\_init\_\_()

self.initUI()

**def** initUI(self):

self.layout = QVBoxLayout(self)

self.tableView = QTableView(self)

self.tableView.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)

self.layout.addWidget(self.tableView)

self.header\_map = {

"ID": "id",

"ФИО (Edit)": "name",

"Логин": "username",

"Пароль": "password",

"Роль (Edit)": "role",

}

self.users = database.get\_all\_users\_as\_dicts()

self.tableModel = CustomStandardItemModel(

len(self.users), len(self.header\_map))

self.tableModel.setHorizontalHeaderLabels(self.header\_map.keys())

self.tableView.setModel(self.tableModel)

self.tableView.horizontalHeader().setStretchLastSection(True)

self.custom\_delegate = CustomDelegate()

self.tableView.setItemDelegate(self.custom\_delegate)

**for** row, user **in** enumerate(self.users):

**for** col, field **in** enumerate(self.header\_map.values()):

item = QStandardItem(str(user.get(field, "")))

**if** field **in** ["name", "role"]:

item.setFlags(item.flags() | Qt.ItemIsEditable)

self.tableModel.setItem(row, col, item)

self.addButton = QPushButton("�"обавить пользователя", self***)***

self.addButton.clicked.connect(self.add\_user***)***

self.layout.addWidget(self.addButton***)***

self.deleteButton = QPushButton("Удалить пользователя", self***)***

self.deleteButton.clicked.connect(self.delete\_user***)***

self.layout.addWidget(self.deleteButton***)***

self.refreshButton = QPushButton("Обновить данные", self***)***

self.refreshButton.clicked.connect(self.refresh\_table***)***

self.layout.addWidget(self.refreshButton***)***

def add\_user(self)***:***

dialog = AddUserDialog(self***)***

if dialog.exec\_() == QDialog.Accepted***:***

user\_data = dialog.get\_inputs(***)***

if user\_data["username"] in [user["username"] for user in self.users]***:***

QMessageBox.warning***(***

self***,***

"Имя пользователя существует"***,***

"Пользователь с таким именем пользователя уже существует."***,***

***)***

else***:***

database.add\_user(\*\*user\_data***)***

self.refresh\_table(***)***

def delete\_user(self)***:***

selected\_indexes = self.tableView.selectionModel().selectedRows(***)***

if selected\_indexes***:***

row = selected\_indexes[0].row(***)***

username\_index = self.tableModel.index(row, 2***)***

username = self.tableModel.data(username\_index, Qt.DisplayRole***)***

reply = QMessageBox.question***(***

self***,***

"Подтвердить удаление"***,***

f'�'ы хотите удалить пользователя с именем пользователя "{username}"?'***,***

QMessageBox.Yes | QMessageBox.No***,***

***)***

if reply == QMessageBox.Yes***:***

database.delete\_user(username***)***

self.refresh\_table(***)***

else***:***

QMessageBox.warning***(***

self***,***

"Предупреждение"***,***

"Пожалуйста, выберите пользователя для удаления."***,***

***)***

def refresh\_table(self)***:***

self.users = database.get\_all\_users\_as\_dicts(***)***

self.tableModel.setRowCount(len(self.users)***)***

for row, user in enumerate(self.users)***:***

for col, field in enumerate(self.header\_map.values())***:***

item = QStandardItem(str(user.get(field, ""))***)***

self.tableModel.setItem(row, col, item***)***

# interfaces\sidebar\profile\_page.py

**from** PyQt5.QtWidgets **import** (QWidget, QLabel, QLineEdit, QPushButton,

QHBoxLayout, QVBoxLayout, QMessageBox, QSpacerItem, QSizePolicy)

**from** PyQt5.QtGui **import** QIcon

**from** PyQt5.QtCore **import** QTimer

**import** database

**class** ProfilePage(QWidget):

**def** \_\_init\_\_(self, main\_window):

super().\_\_init\_\_()

self.main\_window = main\_window

self.initUI()

**def** initUI(self):

main\_layout = QVBoxLayout(self)

form\_layout = QVBoxLayout()

form\_layout.setSpacing(0)

self.name\_label = QLabel("ФИО:", self)

self.name\_edit = QLineEdit(self)

form\_layout.addWidget(self.name\_label)

form\_layout.addWidget(self.name\_edit)

self.username\_label = QLabel("Логин:", self)

self.username\_edit = QLineEdit(self)

form\_layout.addWidget(self.username\_label)

form\_layout.addWidget(self.username\_edit)

self.password\_label = QLabel("Пароль:", self)

self.password\_edit = QLineEdit(self)

self.password\_edit.setEchoMode(QLineEdit.Password)

self.show\_password\_button = QPushButton(

QIcon("resources/icons/eye-icon.png"), "", self)

self.show\_password\_button.clicked.connect(

self.toggle\_password\_visibility)

password\_layout = QHBoxLayout()

password\_layout.addWidget(self.password\_edit)

password\_layout.addWidget(self.show\_password\_button)

form\_layout.addWidget(self.password\_label)

form\_layout.addLayout(password\_layout)

self.save\_button = QPushButton("Сохранить", self)

self.save\_button.clicked.connect(self.update\_user\_info)

form\_layout.addWidget(self.save\_button)

main\_layout.addLayout(form\_layout)

spacer = QSpacerItem(20, 40, QSizePolicy.Minimum,

QSizePolicy.Expanding)

main\_layout.addItem(spacer)

self.load\_user\_info()

**def** load\_user\_info(self):

user\_info = database.get\_user\_info(self.main\_window.user\_id)

**if** user\_info:

self.name\_edit.setText(user\_info["name"])

self.username\_edit.setText(user\_info["username"])

self.password\_edit.setText(user\_info["password"])

**def** update\_user\_info(self):

name = self.name\_edit.text()

username = self.username\_edit.text()

password = self.password\_edit.text()

**if** **not** name **or** **not** username **or** **not** password:

QMessageBox.warning(

self, "Предупреждение", "�'се поля обязательны для заполнения.")

**return**

database.update\_user\_info(

self.main\_window.user\_id, name, username, password)

QMessageBox.information(

self, "Успех", "Информация о пользователе успешно обновленаy.")

**def** toggle\_password\_visibility(self):

**if** self.password\_edit.echoMode() == QLineEdit.Password:

self.password\_edit.setEchoMode(QLineEdit.Normal)

QTimer.singleShot(

3000, **lambda**: self.password\_edit.setEchoMode(

QLineEdit.Password))

**else**:

self.password\_edit.setEchoMode(QLineEdit.Password)

# interfaces/sidebar/settings\_page.py

**from** PyQt5.QtWidgets **import** QWidget, QLabel

**class** SettingsPage(QWidget):

**def** \_\_init\_\_(self):

super().\_\_init\_\_()

self.initUI()

**def** initUI(self):

label = QLabel("This is the Settings Page.", self)

# interfaces\sidebar\sidebar.py

**from** PyQt5.QtWidgets **import** (

QWidget, QVBoxLayout, QPushButton, QSpacerItem, QSizePolicy)

**from** .profile\_page **import** ProfilePage

**from** .settings\_page **import** SettingsPage

**class** SidebarMenu(QWidget):

**def** \_\_init\_\_(self, upper\_buttons, stack, main\_window, parent=None):

super().\_\_init\_\_(parent)

self.stack = stack

self.main\_window = main\_window

self.initUI(upper\_buttons)

**def** initUI(self, upper\_buttons):

layout = QVBoxLayout()

self.buttons = []

**for** button\_name **in** upper\_buttons:

button = QPushButton(button\_name)

layout.addWidget(button)

self.buttons.append(button)

spacer = QSpacerItem(20, 40, QSizePolicy.Minimum,

QSizePolicy.Expanding)

layout.addItem(spacer)

self.profile\_button = QPushButton("Мой профиль")

self.settings\_button = QPushButton("Настройки")

self.logout\_button = QPushButton("�'ыйти из аккаунта")

layout.addWidget(self.profile\_button)

layout.addWidget(self.settings\_button)

layout.addWidget(self.logout\_button)

self.logout\_button.clicked.connect(self.logout)

self.setLayout(layout)

self.profile\_page = ProfilePage(self.main\_window)

self.settings\_page = SettingsPage()

self.stack.addWidget(self.profile\_page)

self.stack.addWidget(self.settings\_page)

**def** connectStack(self):

**for** i, button **in** enumerate(self.buttons):

button.clicked.connect(

**lambda** \_, b=i: self.stack.setCurrentIndex(b))

self.profile\_button.clicked.connect(

**lambda**: self.stack.setCurrentIndex(len(self.buttons)))

self.settings\_button.clicked.connect(

**lambda**: self.stack.setCurrentIndex(len(self.buttons) + 1))

**def** logout(self):

self.main\_window.login\_window.username\_input.clear()

self.main\_window.login\_window.password\_input.clear()

self.main\_window.login\_window.reg\_username\_input.clear()

self.main\_window.login\_window.reg\_password\_input.clear()

self.main\_window.login\_window.reg\_name\_input.clear()

self.main\_window.login\_window.reg\_password\_retry\_input.clear()

self.main\_window.central\_widget.setCurrentWidget(

self.main\_window.login\_window)

# interfaces\student\my\_tests\_page.py

**from** PyQt5.QtWidgets **import** (

QWidget, QVBoxLayout, QTableView, QPushButton, QHeaderView, QMessageBox)

**from** PyQt5.QtGui **import** QStandardItem, QStandardItemModel

**import** database

**from** ..test\_page **import** TakeTestPage

**class** MyTestsPage(QWidget):

**def** \_\_init\_\_(self, student\_window, main\_window):

super().\_\_init\_\_()

self.student\_window = student\_window

self.main\_window = main\_window

self.test\_ids = {}

self.initUI()

**def** initUI(self):

layout = QVBoxLayout(self)

self.tableView = QTableView()

self.model = QStandardItemModel(self.tableView)

self.tableView.setModel(self.model)

self.tableView.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)

self.tableView.setSelectionBehavior(QTableView.SelectRows)

self.tableView.setSelectionMode(QTableView.SingleSelection)

self.tableView.setEditTriggers(QTableView.NoEditTriggers)

layout.addWidget(self.tableView)

take\_test\_button = QPushButton("Пройти тест")

take\_test\_button.clicked.connect(self.takeSelectedTest)

layout.addWidget(take\_test\_button)

refresh\_button = QPushButton("Обновить таблицу")

refresh\_button.clicked.connect(self.loadTestData)

layout.addWidget(refresh\_button)

self.loadTestData()

**def** loadTestData(self):

self.model.clear()

self.model.setColumnCount(5)

self.model.setHorizontalHeaderLabels(

["Название", "Описание", "Количество ост. попыток",

"Кто создал", "Кто назначил"]

)

tests = database.get\_assigned\_tests\_for\_student(

self.main\_window.user\_id)

self.model.setRowCount(len(tests))

**for** row, test **in** enumerate(tests):

self.model.setItem(row, 0, QStandardItem(test["name"]))

self.model.setItem(row, 1, QStandardItem(test["description"]))

self.model.setItem(

row, 2, QStandardItem(str(test["remaining\_attempts"]))

)

self.model.setItem(row, 3, QStandardItem(test["creator\_name"]))

self.model.setItem(row, 4, QStandardItem(test["assigner\_name"]))

self.test\_ids[row] = test["id"]

self.adjustColumnWidths()

**def** takeSelectedTest(self):

selected\_rows = self.tableView.selectionModel().selectedRows()

**if** selected\_rows:

selected\_row = selected\_rows[0].row()

**if** selected\_row **in** self.test\_ids:

test\_id = self.test\_ids[selected\_row]

test\_details\_list = database.get\_assigned\_tests\_for\_student(

self.main\_window.user\_id)

test\_details = next(

(item **for** item **in** test\_details\_list **if** item["id"] == test\_id), None)

**if** test\_details **is** None:

QMessageBox.warning(

self, "Ошибка", "Ошибка при получении информации о тесте.")

**return**

**if** test\_details["remaining\_attempts"] <= 0:

QMessageBox.warning(

self, "Ошибка", "У вас не осталось попыток для этого теста.")

**return**

response = QMessageBox.warning(

self,

"Предупреждение",

"�'ы собираетесь начать тест. �'ы не сможете вернуться к другим разделам, пока не завершите тест. �'ы уверены, что хотите продолжить?",

QMessageBox.Yes | QMessageBox.No,

QMessageBox.No,)

**if** response == QMessageBox.Yes:

self.startTest(test\_id)

**else**:

QMessageBox.warning(

self, "Ошибка", "Пожалуйста, выберите тест для прохождения.")

**def** adjustColumnWidths(self):

column\_count = self.model.columnCount()

total\_width = self.tableView.viewport().width()

column\_width = total\_width // column\_count

**for** column **in** range(column\_count):

self.tableView.setColumnWidth(column, column\_width)

**def** startTest(self, test\_id):

self.student\_window.sidebar.setEnabled(False)

self.take\_test\_page = TakeTestPage(

test\_id, self.main\_window, self.student\_window)

self.student\_window.stack.addWidget(self.take\_test\_page)

self.student\_window.stack.setCurrentWidget(self.take\_test\_page)

# interfaces\student\student\_interface.py

**from** PyQt5.QtWidgets **import** QWidget, QHBoxLayout, QStackedWidget

**from** interfaces.sidebar **import** SidebarMenu

**from** .my\_tests\_page **import** MyTestsPage

**from** ..reports\_page **import** ReportsWindow

**class** StudentWindow(QWidget):

**def** \_\_init\_\_(self, main\_window):

super().\_\_init\_\_()

self.main\_window = main\_window

self.initUI()

**def** initUI(self):

upper\_buttons = ["Мои тесты", "Мои результаты"]

self.stack = QStackedWidget()

self.tests\_page = MyTestsPage(self, self.main\_window)

self.stack.addWidget(self.tests\_page)

self.stack.addWidget(ReportsWindow(self.main\_window))

self.sidebar = SidebarMenu(upper\_buttons, self.stack, self.main\_window)

layout = QHBoxLayout()

layout.addWidget(self.sidebar)

layout.addWidget(self.stack)

self.setLayout(layout)

self.connectButtons()

**def** connectButtons(self):

self.sidebar.connectStack()

# interfaces\teacher\teacher\_interface.py

**from** PyQt5.QtWidgets **import** QWidget, QHBoxLayout, QStackedWidget

**from** interfaces.sidebar **import** SidebarMenu

**from** ..students\_page **import** StudentsPage

**from** ..tests\_page **import** TestsPage

**from** ..reports\_page **import** ReportsWindow

**class** TeacherWindow(QWidget):

**def** \_\_init\_\_(self, main\_window):

super().\_\_init\_\_()

self.main\_window = main\_window

self.initUI()

**def** initUI(self):

upper\_buttons = ['Ученики', 'Тесты', 'Отчеты']

self.stack = QStackedWidget()

self.stack.addWidget(StudentsPage(self.main\_window))

self.test\_page = TestsPage(self)

self.stack.addWidget(self.test\_page)

self.stack.addWidget(ReportsWindow(self.main\_window))

self.sidebar = SidebarMenu(upper\_buttons, self.stack, self.main\_window)

layout = QHBoxLayout()

layout.addWidget(self.sidebar)

layout.addWidget(self.stack)

self.setLayout(layout)

self.connectButtons()

**def** connectButtons(self):

self.sidebar.connectStack()

# interfaces\login.py

**from** PyQt5.QtWidgets **import** (QWidget, QVBoxLayout, QHBoxLayout, QLabel, QLineEdit, QPushButton,

QFormLayout, QFrame, QMessageBox, QStackedWidget, QSpacerItem, QSizePolicy)

**from** PyQt5.QtCore **import** Qt

**import** database

**class** LoginWindow(QWidget):

**def** \_\_init\_\_(self, main\_window):

super().\_\_init\_\_()

self.main\_window = main\_window

self.initUI()

**def** initUI(self):

self.stack = QStackedWidget()

self.login\_form\_widget = self.createLoginForm()

self.register\_form\_widget = self.createRegisterForm()

self.stack.addWidget(self.login\_form\_widget)

self.stack.addWidget(self.register\_form\_widget)

self.login\_button\_top = QPushButton("�'ход")

self.register\_button\_top = QPushButton("Регистрация")

self.login\_button\_top.clicked.connect(

**lambda**: self.stack.setCurrentIndex(0))

self.register\_button\_top.clicked.connect(

**lambda**: self.stack.setCurrentIndex(1))

top\_layout = QHBoxLayout()

top\_layout.addWidget(self.login\_button\_top)

top\_layout.addWidget(self.register\_button\_top)

frame = QFrame()

frame\_layout = QVBoxLayout()

frame\_layout.addLayout(top\_layout)

frame\_layout.addWidget(self.stack)

frame.setLayout(frame\_layout)

main\_layout = QVBoxLayout()

spacer\_top = QSpacerItem(

20, 40, QSizePolicy.Minimum, QSizePolicy.Expanding)

spacer\_bottom = QSpacerItem(

20, 40, QSizePolicy.Minimum, QSizePolicy.Expanding)

main\_layout.addItem(spacer\_top)

main\_layout.addWidget(frame, alignment=Qt.AlignCenter)

main\_layout.addItem(spacer\_bottom)

self.setLayout(main\_layout)

**def** createLoginForm(self):

form\_layout = QFormLayout()

self.username\_input = QLineEdit()

self.password\_input = QLineEdit()

self.password\_input.setEchoMode(QLineEdit.Password)

form\_layout.addRow("Логин", self.username\_input)

form\_layout.addRow("Пароль", self.password\_input)

login\_button = QPushButton("�'ойти")

login\_button.clicked.connect(self.login)

layout = QVBoxLayout()

layout.addLayout(form\_layout)

layout.addWidget(login\_button)

frame = QFrame()

frame.setLayout(layout)

**return** frame

**def** createRegisterForm(self):

form\_layout = QFormLayout()

self.reg\_name\_input = QLineEdit()

self.reg\_username\_input = QLineEdit()

self.reg\_password\_input = QLineEdit()

self.reg\_password\_input.setEchoMode(QLineEdit.Password)

self.reg\_password\_retry\_input = QLineEdit()

self.reg\_password\_retry\_input.setEchoMode(QLineEdit.Password)

form\_layout.addRow("ФИО", self.reg\_name\_input)

form\_layout.addRow("Логин", self.reg\_username\_input)

form\_layout.addRow("Пароль", self.reg\_password\_input)

form\_layout.addRow("Повторите пароль", self.reg\_password\_retry\_input)

register\_button = QPushButton("�-арегистрироваться")

register\_button.clicked.connect(self.register)

layout = QVBoxLayout()

layout.addLayout(form\_layout)

layout.addWidget(register\_button)

frame = QFrame()

frame.setLayout(layout)

**return** frame

**def** switchToRegister(self):

self.stack.setCurrentIndex(1)

**def** switchToLogin(self):

self.stack.setCurrentIndex(0)

**def** login(self):

username = self.username\_input.text()

password = self.password\_input.text()

user\_info = database.authenticate\_user(username, password)

**if** user\_info:

user\_id, role = user\_info

self.main\_window.logged\_in\_user\_id = user\_id

self.main\_window.user\_role = role

self.main\_window.user\_id = user\_id

**if** role == "ADMIN":

self.main\_window.initAdminWindow()

self.main\_window.central\_widget.setCurrentWidget(

self.main\_window.admin\_window)

**elif** role == "TEACHER":

self.main\_window.initTeacherWindow()

self.main\_window.central\_widget.setCurrentWidget(

self.main\_window.teacher\_window)

**elif** role == "STUDENT":

self.main\_window.initStudentWindow()

self.main\_window.central\_widget.setCurrentWidget(

self.main\_window.student\_window)

**else**:

QMessageBox.warning(self, "Ошибка", "Неверный логин или пароль")

**def** register(self):

name = self.reg\_name\_input.text().strip()

username = self.reg\_username\_input.text().strip()

password = self.reg\_password\_input.text()

retry\_password = self.reg\_password\_retry\_input.text()

**if** **not** all([name, username, password, retry\_password]):

QMessageBox.warning(

self, "Ошибка", "�'се поля должны быть заполнены")

**return**

**if** password != retry\_password:

QMessageBox.warning(self, "Ошибка", "Пароли не совпадают")

**return**

**if** database.check\_existing\_user(username):

QMessageBox.warning(

self, "Ошибка", "Пользователь с таким логином уже существует")

**return**

database.add\_user(name, username, password, "STUDENT")

QMessageBox.information(self, "Успех", "Регистрация прошла успешно")

self.switchToLogin()

# interfaces\reports\_page.py

**from** PyQt5.QtWidgets **import** QWidget, QVBoxLayout, QTableView, QHeaderView, QAbstractItemView, QPushButton, QTableWidget, QTableWidgetItem

**from** PyQt5.QtCore **import** Qt, QModelIndex

**from** PyQt5.QtGui **import** QStandardItemModel, QStandardItem, QColor

**import** database

**class** TestAttemptDetailsWindow(QWidget):

**def** \_\_init\_\_(self, student\_id, test\_id, attempt\_id):

super().\_\_init\_\_()

self.student\_id = student\_id

self.test\_id = test\_id

self.attempt\_id = attempt\_id

self.initUI()

**def** initUI(self):

self.layout = QVBoxLayout(self)

self.questionsTable = QTableWidget()

self.questionsTable.setColumnCount(2) # Question and Answer columns

self.questionsTable.setHorizontalHeaderLabels(['�'опрос', 'Ответ']***)***

self.questionsTable.verticalHeader().setVisible(False***)***

self.questionsTable.horizontalHeader().setStretchLastSection(True***)***

self.layout.addWidget(self.questionsTable***)***

self.loadAttemptDetails(***)***

def loadAttemptDetails(self)***:***

attempt\_details = database.get\_attempt\_details***(***

self.student\_id, self.test\_id, self.attempt\_id***)***

self.questionsTable.setRowCount(len(attempt\_details)***)***

for row, (question, student\_answer, correct\_answer) in enumerate(attempt\_details)***:***

question\_item = QTableWidgetItem(question***)***

answer\_item = QTableWidgetItem(student\_answer***)***

if student\_answer == correct\_answer***:***

answer\_item.setBackground***(***

QColor(0, 255, 0)***)***

else***:***

answer\_item.setBackground***(***

QColor(255, 0, 0)***)***

self.questionsTable.setItem(row, 0, question\_item***)***

self.questionsTable.setItem(row, 1, answer\_item***)***

class ReportsWindow(QWidget)***:***

def \_\_init\_\_(self, main\_window)***:***

super().\_\_init\_\_(***)***

self.main\_window = main\_windo***w***

self.initUI(***)***

def initUI(self)***:***

self.layout = QVBoxLayout(self***)***

self.studentsListView = QTableView(***)***

self.studentsModel = StudentsTableModel(***)***

self.studentsListView.setModel(self.studentsModel***)***

self.studentsListView.verticalHeader().setVisible(False***)***

self.studentsListView.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch***)***

self.studentsListView.setEditTriggers(QAbstractItemView.NoEditTriggers***)***

self.studentsListView.clicked.connect(self.onStudentSelected***)***

self.layout.addWidget(self.studentsListView***)***

self.testsTableView = QTableView(***)***

self.testsTableModel = TestsTableModel(***)***

self.testsTableView.setModel(self.testsTableModel***)***

self.testsTableView.verticalHeader().setVisible(False***)***

self.testsTableView.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch***)***

self.testsTableView.clicked.connect(self.onTestSelected***)***

self.testsTableView.setEditTriggers(QAbstractItemView.NoEditTriggers***)***

self.layout.addWidget(self.testsTableView***)***

self.studentAttemptsTableModel = QStandardItemModel(***)***

self.studentAttemptsTableView = QTableView(***)***

self.studentAttemptsTableView.setModel(self.studentAttemptsTableModel***)***

self.studentAttemptsTableView.verticalHeader().setVisible(False***)***

self.studentAttemptsTableView.horizontalHeader***(***

).setSectionResizeMode(QHeaderView.Stretch***)***

self.studentAttemptsTableView.setEditTriggers***(***

QAbstractItemView.NoEditTriggers***)***

self.layout.addWidget(self.studentAttemptsTableView***)***

self.viewDetailsButton = QPushButton("Просмотреть ответы"***)***

self.viewDetailsButton.clicked.connect(self.onViewDetails***)***

self.layout.addWidget(self.viewDetailsButton***)***

self.updateButton = QPushButton("Обновить данные"***)***

self.updateButton.clicked.connect(self.populateStudentsList***)***

self.layout.addWidget(self.updateButton***)***

self.populateStudentsList(***)***

def onViewDetails(self)***:***

selected\_attempt\_index = self.studentAttemptsTableView.currentIndex(***)***

if not selected\_attempt\_index.isValid()***:***

retur***n***

attempt\_id = self.studentAttemptsTableModel.data***(***

selected\_attempt\_index, Qt.UserRole***)***

self.detailsWindow = TestAttemptDetailsWindow***(***

self.selected\_student\_id, self.selected\_test\_id, attempt\_id***)***

self.detailsWindow.show(***)***

def populateStudentsList(self)***:***

students = database.get\_all\_students(***)***

self.studentsModel.setStudents(students***)***

def onStudentSelected(self, index: QModelIndex)***:***

student\_id = self.studentsModel.data(index, Qt.UserRole***)***

self.studentAttemptsTableModel.clear(***)***

self.loadStudentTests(student\_id***)***

def loadStudentTests(self, student\_id)***:***

tests = database.get\_tests\_by\_student(student\_id***)***

self.testsTableModel.setTests(tests***)***

def onTestSelected(self, index: QModelIndex)***:***

test\_id = self.testsTableModel.testId(index***)***

student\_id = self.studentsListView.currentIndex().data(Qt.UserRole***)***

self.updateTestAttempts(student\_id, test\_id***)***

def updateTestAttempts(self, student\_id, test\_id)***:***

attempts\_data = database.get\_student\_test\_attempt\_results***(***

student\_id, test\_id***)***

self.studentAttemptsTableModel.clear(***)***

self.studentAttemptsTableModel.setHorizontalHeaderLabels***(***

['Номер попытки', 'Результат']***)***

attempt\_results = {***}***

for test\_result\_id, is\_correct, \_ in attempts\_data***:***

if test\_result\_id not in attempt\_results***:***

attempt\_results[test\_result\_id] = {'total': 0, 'correct': 0***}***

attempt\_results[test\_result\_id]['total'] += ***1***

if is\_correct***:***

attempt\_results[test\_result\_id]['correct'] += ***1***

for attempt\_number, result in enumerate(attempt\_results.values(), start=1)***:***

result\_str = f"{result['correct']}/{result['total']}***"***

row = [QStandardItem(str(attempt\_number))***,***

QStandardItem(result\_str)***]***

self.studentAttemptsTableModel.appendRow(row***)***

class TestsTableModel(QStandardItemModel)***:***

def \_\_init\_\_(self)***:***

super().\_\_init\_\_(***)***

self.setHorizontalHeaderLabels(['Название теста']***)***

self.testData = [***]***

def setTests(self, tests)***:***

self.clear(***)***

self.setHorizontalHeaderLabels(['Название теста']***)***

self.testData = test***s***

for test in tests***:***

row = [QStandardItem(test['name'])***]***

self.appendRow(row***)***

def testId(self, index)***:***

if 0 <= index.row() < len(self.testData)***:***

return self.testData[index.row()]['id'***]***

return Non***e***

class StudentsTableModel(QStandardItemModel)***:***

def \_\_init\_\_(self, students=None)***:***

super().\_\_init\_\_(***)***

self.students = students or [***]***

def setStudents(self, students)***:***

self.students = student***s***

self.clear(***)***

self.setHorizontalHeaderLabels(['Ученики']***)***

self.setColumnCount(1***)***

self.setRowCount(len(self.students)***)***

for row, student in enumerate(self.students)***:***

item = QStandardItem(student["name"]***)***

self.setItem(row, 0, item***)***

def data(self, index, role=Qt.DisplayRole)***:***

if role == Qt.UserRole***:***

return self.students[index.row()]["id"***]***

return super().data(index, role***)***

# interfaces\students\_page.py

**from** PyQt5.QtWidgets **import** (QWidget, QPushButton, QVBoxLayout, QTableView, QDialog,

QStyledItemDelegate, QPushButton, QInputDialog, QMessageBox, QHeaderView, QCheckBox, QComboBox)

**from** PyQt5.QtCore **import** Qt, QModelIndex, QRect, pyqtSignal, QAbstractTableModel

**from** PyQt5.QtGui **import** QStandardItemModel, QStandardItem, QIcon, QStandardItem, QStandardItemModel

**import** database

**class** GroupManagementDialog(QDialog):

**def** \_\_init\_\_(self, win, main\_window, parent=None):

super().\_\_init\_\_(parent)

self.win = win

self.main\_window = main\_window

self.initUI()

**def** initUI(self):

self.layout = QVBoxLayout(self)

self.groupsModel = QStandardItemModel()

self.groupsModel.setHorizontalHeaderLabels(["Имя группы"])

self.groupsTable = QTableView(self)

self.groupsTable.setModel(self.groupsModel)

self.groupsTable.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)

self.groupsTable.setSelectionBehavior(QTableView.SelectRows)

self.groupsTable.setSelectionMode(QTableView.SingleSelection)

self.groupsTable.setEditTriggers(QTableView.NoEditTriggers)

self.layout.addWidget(self.groupsTable)

self.addGroupButton = QPushButton("�"обавить группу", self***)***

self.addGroupButton.clicked.connect(self.add\_group***)***

self.layout.addWidget(self.addGroupButton***)***

self.deleteGroupButton = QPushButton("Удалить группу", self***)***

self.deleteGroupButton.clicked.connect(self.delete\_group***)***

self.layout.addWidget(self.deleteGroupButton***)***

self.assignTestButton = QPushButton("Назначить тест на группу", self***)***

self.assignTestButton.clicked.connect(self.assign\_test\_to\_group***)***

self.layout.addWidget(self.assignTestButton***)***

self.refreshButton = QPushButton("Обновить данные", self***)***

self.refreshButton.clicked.connect(self.load\_groups***)***

self.layout.addWidget(self.refreshButton***)***

self.setWindowTitle("Управление группами"***)***

self.setGeometry(300, 300, 400, 300***)***

self.load\_groups(***)***

def add\_group(self)***:***

text, ok = QInputDialog.getText***(***

self, "�"обавление группы", "�'ведите название группы:")

**if** ok **and** text:

existing\_groups = [item.text()

**for** item **in** self.groupsModel.findItems(text)]

**if** existing\_groups:

QMessageBox.warning(

self, "�"руппа уже существует", "�"руппа уже существует.")

**else**:

database.add\_group(text)

self.load\_groups()

**def** delete\_group(self):

selected = self.groupsTable.selectionModel().selectedRows()

**if** selected:

row = selected[0].row()

group\_item = self.groupsModel.item(row)

group\_name = group\_item.text()

database.delete\_group(group\_name)

self.load\_groups()

**def** assign\_test\_to\_group(self):

selected = self.groupsTable.selectionModel().selectedRows()

**if** **not** selected:

QMessageBox.warning(

self,

"�"руппа не выбрана"***,***

"Пожалуйста, выберите группу, чтобы назначить заданиеst."***,***

***)***

retur***n***

row = selected[0].row(***)***

group\_item = self.groupsModel.item(row***)***

group\_name = group\_item.text(***)***

group\_id = database.get\_group\_id\_by\_name(group\_name***)***

if group\_id is not None***:***

dialog = AssignTestToGroupDialog***(***

group\_id, self.main\_window.user\_id, self***)***

dialog.exec\_(***)***

else***:***

QMessageBox.warning***(***

self, "Ошибка", "Невозможно найти выбранную группу в базе данных."

***)***

self.win.refresh\_students(***)***

def load\_groups(self)***:***

groups = database.get\_all\_groups(***)***

self.groupsModel.clear(***)***

self.groupsModel.setHorizontalHeaderLabels(["Имя группы"]***)***

for group in groups***:***

group\_name = group["name"***]***

item = QStandardItem(group\_name***)***

self.groupsModel.appendRow(item***)***

class StudentsTableModel(QAbstractTableModel)***:***

def \_\_init\_\_(self, students\_data)***:***

super().\_\_init\_\_(***)***

self.students\_data = students\_dat***a***

self.header = ["ФИО", "�"руппа", "Назначенные тесты"]

**def** rowCount(self, parent=QModelIndex()):

**return** len(self.students\_data)

**def** columnCount(self, parent=QModelIndex()):

**return** len(self.header)

**def** data(self, index, role):

**if** **not** index.isValid():

**return** None

**if** role == Qt.DisplayRole **or** role == Qt.EditRole:

row = index.row()

col = index.column()

student = self.students\_data[row]

**if** col == 0:

**return** student["name"]

**elif** col == 1:

**return** student["group"]

**elif** col == 2:

**return** student["tests"]

**return** None

**def** headerData(self, section, orientation, role=Qt.DisplayRole):

**if** role == Qt.DisplayRole:

**if** orientation == Qt.Horizontal:

**return** self.header[section]

**elif** orientation == Qt.Vertical:

**return** str(section + 1)

**return** None

**def** get\_group\_dropdown\_data(self):

groups = database.get\_all\_groups()

**return** {group["id"]: group["name"] **for** group **in** groups}

**def** flags(self, index):

**if** index.column() == 1:

**return** Qt.ItemIsEnabled | Qt.ItemIsSelectable

**else**:

**return** super().flags(index)

**class** GroupSelectionDialog(QDialog):

**def** \_\_init\_\_(self, parent=None):

super().\_\_init\_\_(parent)

self.initUI()

**def** initUI(self):

self.setWindowTitle("�'ыбор группы")

self.layout = QVBoxLayout(self)

self.groupComboBox = QComboBox(self)

**for** group **in** database.get\_all\_groups():

self.groupComboBox.addItem(group["name"], group["id"])

self.layout.addWidget(self.groupComboBox)

self.assignButton = QPushButton("Сохранить", self)

self.assignButton.clicked.connect(self.accept)

self.layout.addWidget(self.assignButton)

**def** selected\_group\_id(self):

**return** self.groupComboBox.currentData()

**class** StudentsPage(QWidget):

**def** \_\_init\_\_(self, main\_window):

super().\_\_init\_\_()

self.main\_window = main\_window

self.initUI()

**def** initUI(self):

self.layout = QVBoxLayout(self)

students\_data = database.get\_all\_students()

self.studentsModel = StudentsTableModel(students\_data)

self.studentsTable = QTableView()

self.studentsTable.setModel(self.studentsModel)

self.studentsTable.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)

self.studentsTable.setSelectionBehavior(QTableView.SelectRows)

self.studentsTable.setSelectionMode(QTableView.SingleSelection)

self.studentsTable.setEditTriggers(QTableView.NoEditTriggers)

self.groupDelegate = GroupColumnDelegate(self)

self.studentsTable.setItemDelegateForColumn(1, self.groupDelegate)

self.groupDelegate.assign\_group.connect(self.assign\_group)

self.groupDelegate.remove\_group.connect(self.remove\_group)

self.layout.addWidget(self.studentsTable)

self.manageGroupsButton = QPushButton("Управление группами", self)

self.manageGroupsButton.clicked.connect(self.open\_group\_management)

self.layout.addWidget(self.manageGroupsButton)

self.assignTestStudentButton = QPushButton(

"Назначить тест студенту", self)

self.assignTestStudentButton.clicked.connect(

self.assign\_test\_to\_student)

self.layout.addWidget(self.assignTestStudentButton)

self.refreshButton = QPushButton("Обновить данные", self)

self.refreshButton.clicked.connect(self.refresh\_students)

self.layout.addWidget(self.refreshButton)

self.refresh\_students()

**def** open\_group\_management(self):

dialog = GroupManagementDialog(self, self.main\_window)

dialog.exec\_()

**def** assign\_test\_to\_student(self):

selected = self.studentsTable.selectionModel().selectedRows()

**if** selected:

row = selected[0].row()

student\_id = self.studentsModel.students\_data[row]["id"]

AssignTestDialog(

student\_id, self.main\_window.user\_id, self).exec\_()

self.refresh\_students()

**def** refresh\_students(self):

students\_data = database.get\_all\_students()

self.studentsModel.students\_data = students\_data

self.studentsModel.layoutChanged.emit()

self.studentsTable.viewport().update()

**def** assign\_group(self, row):

dialog = GroupSelectionDialog(self)

**if** dialog.exec\_():

group\_id = dialog.selected\_group\_id()

student\_id = self.studentsModel.students\_data[row]["id"]

database.set\_student\_group(student\_id, group\_id)

self.refresh\_students()

**def** remove\_group(self, row):

student\_id = self.studentsModel.students\_data[row]["id"]

database.reset\_student\_group(student\_id)

self.refresh\_students()

**class** GroupColumnDelegate(QStyledItemDelegate):

assign\_group = pyqtSignal(int)

remove\_group = pyqtSignal(int)

**def** \_\_init\_\_(self, parent=None):

super().\_\_init\_\_(parent)

**def** paint(self, painter, option, index):

group\_name = index.model().data(index, Qt.DisplayRole)

painter.drawText(option.rect.adjusted(5, 0, -30, 0),

Qt.AlignVCenter, group\_name)

assign\_icon = QIcon("resources/icons/assign.png")

remove\_icon = QIcon("resources/icons/close.png")

icon\_size = 16

right\_edge = option.rect.right()

remove\_icon\_rect = QRect(

right\_edge - icon\_size, option.rect.top(), icon\_size, icon\_size)

assign\_icon\_rect = QRect(

right\_edge - icon\_size \* 2 - 5, option.rect.top(), icon\_size, icon\_size)

painter.drawPixmap(

assign\_icon\_rect, assign\_icon.pixmap(icon\_size, icon\_size))

painter.drawPixmap(

remove\_icon\_rect, remove\_icon.pixmap(icon\_size, icon\_size))

**def** editorEvent(self, event, model, option, index):

**if** **not** index.isValid():

**return** False

icon\_size = 16

right\_edge = option.rect.right()

assign\_button\_rect = QRect(

right\_edge - icon\_size \* 2 - 5,

option.rect.top(),

icon\_size,

option.rect.height(),)

remove\_button\_rect = QRect(

right\_edge - icon\_size, option.rect.top(), icon\_size, option.rect.height())

**if** assign\_button\_rect.contains(event.pos()):

self.assign\_group.emit(index.row())

**return** True

**elif** remove\_button\_rect.contains(event.pos()):

self.remove\_group.emit(index.row())

**return** True

**return** False

**class** AssignTestDialog(QDialog):

**def** \_\_init\_\_(self, student\_id, user\_id, parent=None):

super().\_\_init\_\_(parent)

self.student\_id = student\_id

self.user\_id = user\_id

self.assigned\_tests = set(database.get\_tests\_for\_student(student\_id))

self.initUI()

**def** initUI(self):

self.layout = QVBoxLayout(self)

self.tests = database.get\_all\_tests\_as\_dict()

**for** test\_id, test\_name **in** self.tests:

checkBox = QCheckBox(test\_name, self)

checkBox.test\_id = test\_id

checkBox.setChecked(test\_id **in** self.assigned\_tests)

self.layout.addWidget(checkBox)

self.assignButton = QPushButton("Сохранить", self)

self.assignButton.clicked.connect(self.assign\_selected\_tests)

self.layout.addWidget(self.assignButton)

**def** assign\_selected\_tests(self):

**for** i **in** range(self.layout.count()):

widget = self.layout.itemAt(i).widget()

**if** isinstance(widget, QCheckBox):

**if** widget.isChecked() **and** widget.test\_id **not** **in** self.assigned\_tests:

database.assign\_test\_to\_student(

widget.test\_id, self.student\_id, self.user\_id)

**elif** **not** widget.isChecked() **and** widget.test\_id **in** self.assigned\_tests:

database.remove\_test\_from\_student(

widget.test\_id, self.student\_id)

self.accept()

**class** AssignTestToGroupDialog(QDialog):

**def** \_\_init\_\_(self, group\_id, user\_id, parent=None):

super().\_\_init\_\_(parent)

self.group\_id = group\_id

self.user\_id = user\_id

self.initUI()

**def** initUI(self):

self.layout = QVBoxLayout(self)

self.tests = database.get\_all\_tests\_as\_dict()

self.testCheckBoxes = []

**for** test\_id, test\_name **in** self.tests:

checkBox = QCheckBox(test\_name, self)

checkBox.test\_id = test\_id

self.testCheckBoxes.append(checkBox)

self.layout.addWidget(checkBox)

self.assignButton = QPushButton(

"Назначить выбранные тесты на группу", self)

self.assignButton.clicked.connect(self.assign\_tests)

self.layout.addWidget(self.assignButton)

self.removeButton = QPushButton(

"Отозвать выбранные тесты на группу", self)

self.removeButton.clicked.connect(self.remove\_tests)

self.layout.addWidget(self.removeButton)

**def** assign\_tests(self):

**for** checkBox **in** self.testCheckBoxes:

**if** checkBox.isChecked():

database.assign\_test\_to\_group\_students(

checkBox.test\_id, self.group\_id, self.user\_id)

QMessageBox.information(

self, "Тесты назначены", "�'ыбранные тесты были назначены группе.")

self.accept()

**def** remove\_tests(self):

**for** checkBox **in** self.testCheckBoxes:

**if** checkBox.isChecked():

database.remove\_test\_assignment\_from\_group(

checkBox.test\_id, self.group\_id

)

QMessageBox.information(

self, "Тесты удалены", "�'ыбранные тесты были удалены из группы.")

self.accept()

# interfaces\test\_page.py

**from** PyQt5.QtWidgets **import** (QWidget, QVBoxLayout, QTabWidget, QFormLayout, QLabel, QLineEdit, QSpinBox, QTextEdit,

QComboBox, QPushButton, QScrollArea, QGroupBox, QHBoxLayout, QRadioButton, QCheckBox, QMessageBox, QFileDialog)

**from** PyQt5.QtCore **import** Qt

**from** PyQt5.QtGui **import** QPixmap

**import** database

**class** ViewTestPage(QWidget):

**def** \_\_init\_\_(self, tests\_page, test\_id, admin\_window):

super().\_\_init\_\_()

self.test\_id = test\_id

self.tests\_page = tests\_page

self.test\_details = database.get\_test\_details(self.test\_id)

self.test\_name = self.test\_details["name"]

self.admin\_window = admin\_window

self.initUI()

**def** initUI(self):

layout = QVBoxLayout(self)

label = QLabel(f"Тест: {self.test\_name}", self)

layout.addWidget(label)

self.scrollArea = QScrollArea(self)

self.questionsWidget = QWidget()

self.questionsLayout = QVBoxLayout(self.questionsWidget)

self.scrollArea.setWidget(self.questionsWidget)

self.scrollArea.setWidgetResizable(True)

layout.addWidget(self.scrollArea)

close\_button = QPushButton("�-акрыть")

close\_button.clicked.connect(self.close\_view)

layout.addWidget(close\_button)

self.loadTest()

**def** close\_view(self):

self.close()

self.admin\_window.stack.setCurrentWidget(self.admin\_window.test\_page)

**def** loadTest(self):

test\_details = database.get\_test\_details(self.test\_id)

**for** question **in** test\_details["questions"]:

self.addQuestion(question)

**def** addQuestion(self, question):

group\_box = QGroupBox()

group\_box\_layout = QVBoxLayout(group\_box)

**if** question.get("image\_path"):

image\_label = QLabel()

image\_label.setAlignment(Qt.AlignCenter)

pixmap = QPixmap(question["image\_path"])

image\_label.setPixmap(pixmap)

image\_label.setFixedSize(200, 200)

group\_box\_layout.addWidget(image\_label)

question\_text\_edit = QTextEdit(question["text"])

question\_text\_edit.setReadOnly(True)

group\_box\_layout.addWidget(question\_text\_edit)

**for** answer **in** question["answers"]:

**if** question["type"] == "Единственный правильный ответ":

answer\_widget = QRadioButton(answer["text"])

**else**:

answer\_widget = QCheckBox(answer["text"])

answer\_widget.setProperty("answer\_id", answer["id"])

**if** answer["is\_correct"]:

answer\_widget.setChecked(True)

answer\_widget.setDisabled(True)

group\_box\_layout.addWidget(answer\_widget)

self.questionsLayout.addWidget(group\_box)

**class** CreateTestPage(QWidget):

**def** \_\_init\_\_(self, tests\_page, creator\_id, refresh\_tests):

super().\_\_init\_\_()

self.answer\_widgets = {}

self.tests\_page = tests\_page

self.creator\_id = creator\_id

self.refresh\_tests = refresh\_tests

self.initUI()

**def** save\_test(self):

**if** **not** self.validate\_test():

**return**

test\_name = self.test\_name.text()

attempts = self.attempts\_spinbox.value()

questions = []

**for** group\_box, answer\_widgets **in** self.answer\_widgets.items():

question\_text = group\_box.findChild(QTextEdit).toPlainText()

question\_type = answer\_widgets["type"]

answers = []

**for** answer\_info **in** answer\_widgets["answers"]:

answer\_text = answer\_info["input"].text()

is\_correct = answer\_info["correct"].isChecked()

answers.append({"text": answer\_text, "is\_correct": is\_correct})

questions.append(

{"text": question\_text, "type": question\_type, "answers": answers}

)

database.save\_test\_to\_database(

test\_name, attempts, questions, self.creator\_id)

QMessageBox.information(self, "Успех", "Тест успешно сохранен..")

self.tests\_page.return\_to\_previous\_tab()

self.refresh\_tests()

**def** closeEvent(self, event):

self.tests\_page.return\_to\_previous\_tab()

event.accept()

**def** validate\_test(self):

**if** **not** self.test\_name.text():

QMessageBox.warning(self, "Ошибка проверки",

"Укажите название теста.")

**return** False

**if** self.attempts\_spinbox.value() <= 0:

QMessageBox.warning(

self, "Ошибка проверки", "Количество попыток должно быть больше 0.")

**return** False

**if** len(self.answer\_widgets) < 1:

QMessageBox.warning(

self, "Ошибка проверки", "Требуется хотя бы один вопрос.")

**return** False

**for** group\_box, answer\_widgets **in** self.answer\_widgets.items():

question\_text = group\_box.findChild(QTextEdit).toPlainText()

**if** **not** question\_text:

QMessageBox.warning(

self, "Ошибка проверки", "Каждый вопрос должен иметь заголовок.")

**return** False

**if** len(answer\_widgets["answers"]) < 2:

QMessageBox.warning(

self,

"Ошибка проверки",

"Каждый вопрос должен иметь не менее двух вариантов ответа.",)

**return** False

**if** **not** any(info["correct"].isChecked() **for** info **in** answer\_widgets["answers"]):

QMessageBox.warning(

self,

"Ошибка проверки",

"�' каждом вопросе должен быть отмечен хотя бы один правильный ответ.",)

**return** False

**for** answer\_info **in** answer\_widgets["answers"]:

answer\_text = answer\_info["input"].text()

**if** **not** answer\_text:

QMessageBox.warning(

self,

"Ошибка проверки",

"Каждый вариант ответа должен быть заполнен.")

**return** False

**return** True

**def** initUI(self):

layout = QVBoxLayout(self)

label = QLabel("Создание теста", self)

layout.addWidget(label)

self.tabWidget = QTabWidget(self)

layout.addWidget(self.tabWidget)

buttons\_layout = QHBoxLayout()

close\_button = QPushButton("�-акрыть редактор")

close\_button.clicked.connect(self.close)

save\_button = QPushButton("Сохранить тест")

save\_button.clicked.connect(self.save\_test)

buttons\_layout.addWidget(save\_button)

buttons\_layout.addWidget(close\_button)

layout.addLayout(buttons\_layout)

self.createManualTab()

self.createTemplateTab()

**def** createManualTab(self):

self.manualTab = QWidget()

self.manualLayout = QVBoxLayout(self.manualTab)

self.test\_name = QLineEdit(self)

self.attempts\_spinbox = QSpinBox(self)

self.question\_counter\_label = QLabel("�'опросы: 0")

add\_question\_button = QPushButton("�"обавить вопрос"***)***

add\_question\_button.clicked.connect(self.addNewQuestion***)***

test\_info\_layout = QFormLayout(***)***

test\_info\_layout.addRow("Название теста:", self.test\_name***)***

test\_info\_layout.addRow("Попытки:", self.attempts\_spinbox***)***

test\_info\_layout.addRow***(***

self.question\_counter\_label, add\_question\_button***)***

self.manualLayout.addLayout(test\_info\_layout***)***

self.tabWidget.addTab(self.manualTab, "Ручное создание"***)***

self.scrollArea = QScrollArea(self***)***

self.questionsWidget = QWidget(***)***

self.questionsLayout = QVBoxLayout(self.questionsWidget***)***

self.scrollArea.setWidget(self.questionsWidget***)***

self.scrollArea.setWidgetResizable(True***)***

self.manualLayout.addWidget(self.scrollArea***)***

def addNewQuestion(self)***:***

self.addQuestion(***)***

self.updateQuestionCounter(***)***

def addQuestion(self)***:***

group\_box = QGroupBox(***)***

group\_box\_layout = QVBoxLayout(group\_box***)***

image\_label = QLabel(***)***

image\_label.setAlignment(Qt.AlignCenter***)***

image\_label.setFixedSize(200, 200***)***

image\_label.setVisible(False***)***

header\_layout = QHBoxLayout(***)***

add\_image\_button = QPushButton("�"обавить изображениеe")

add\_image\_button.clicked.connect(

**lambda**: self.addImageToQuestion(image\_label))

header\_layout.addWidget(add\_image\_button)

remove\_image\_button = QPushButton("Удалить изображение")

remove\_image\_button.clicked.connect(

**lambda**: self.removeImageFromQuestion(image\_label))

header\_layout.addWidget(remove\_image\_button)

delete\_question\_button = QPushButton("Удалить вопрос")

delete\_question\_button.clicked.connect(

**lambda** \_, gb=group\_box: self.removeQuestion(gb))

header\_layout.addStretch(1)

header\_layout.addWidget(delete\_question\_button)

group\_box\_layout.addLayout(header\_layout)

group\_box\_layout.addWidget(image\_label)

question\_text\_edit = QTextEdit()

question\_text\_edit.setPlaceholderText(

"�'ведите сюда текст вашего вопроса...")

group\_box\_layout.addWidget(question\_text\_edit)

question\_type = QComboBox()

question\_type.addItems(

["Единственный правильный ответ", "Несколько правильных ответов"])

question\_type.currentTextChanged.connect(

**lambda** qtype, gb=group\_box: self.changeAnswerType(gb, qtype))

group\_box\_layout.addWidget(question\_type)

answers\_layout = QVBoxLayout()

group\_box\_layout.addLayout(answers\_layout)

self.questionsLayout.addWidget(group\_box)

add\_answer\_button = QPushButton("�"обавить вариант ответа"***)***

add\_answer\_button.clicked.connect***(***

lambda: self.addAnswerOption(group\_box, answers\_layout)***)***

group\_box\_layout.addWidget(add\_answer\_button***)***

self.answer\_widgets[group\_box] = ***{***

"layout": answers\_layout***,***

"type": "Единственный правильный ответ"***,***

"answers": []***,***

***}***

def addImageToQuestion(self, image\_label)***:***

options = QFileDialog.Options(***)***

options |= QFileDialog.ReadOnl***y***

file\_name, \_ = QFileDialog.getOpenFileName***(***

self***,***

"Select an image file"***,***

""***,***

"Images (\*.png \*.jpg \*.bmp \*.gif \*.jpeg);;All Files (\*)"***,***

options=options***,***

***)***

if file\_name***:***

pixmap = QPixmap(file\_name***)***

image\_label.setPixmap(pixmap***)***

image\_label.setFixedSize(200, 200***)***

image\_label.setVisible(True***)***

def removeImageFromQuestion(self, image\_label)***:***

image\_label.clear(***)***

image\_label.setVisible(False***)***

def changeAnswerType(self, group\_box, question\_type)***:***

answer\_info\_list = self.answer\_widgets[group\_box]["answers"***]***

self.answer\_widgets[group\_box]["type"] = question\_typ***e***

for answer\_info in answer\_info\_list***:***

correct\_widget = answer\_info["correct"***]***

answer\_info["layout"].removeWidget(correct\_widget***)***

correct\_widget.deleteLater(***)***

if question\_type == "Единственный правильный ответ"***:***

correct\_widget = QRadioButton(***)***

else***:***

correct\_widget = QCheckBox(***)***

answer\_info["correct"] = correct\_widge***t***

answer\_info["layout"].insertWidget(0, correct\_widget***)***

def addAnswerOption(self, group\_box, layout)***:***

answer\_widgets = self.answer\_widgets[group\_box]["answers"***]***

answer\_layout = QHBoxLayout(***)***

if self.answer\_widgets[group\_box]["type"] == "Единственный правильный ответ"***:***

correct\_answer\_input = QRadioButton(***)***

else***:***

correct\_answer\_input = QCheckBox(***)***

answer\_label = QLabel(f"Ответ {len(answer\_widgets) + 1}:"***)***

answer = QLineEdit(***)***

delete\_answer\_button = QPushButton("X"***)***

delete\_answer\_button.clicked.connect***(***

lambda: self.removeAnswerOption(group\_box, layout, answer\_layout)***)***

answer\_layout.addWidget(correct\_answer\_input***)***

answer\_layout.addWidget(answer\_label***)***

answer\_layout.addWidget(answer***)***

answer\_layout.addWidget(delete\_answer\_button***)***

layout.addLayout(answer\_layout***)***

answer\_widgets.append***(***

{"layout": answer\_layout, "input": answer, "correct": correct\_answer\_input}***)***

def removeAnswerOption(self, group\_box, parent\_layout, answer\_layout)***:***

while answer\_layout.count()***:***

item = answer\_layout.takeAt(0***)***

if item.widget()***:***

item.widget().deleteLater(***)***

answer\_info\_list = self.answer\_widgets[group\_box]["answers"***]***

self.answer\_widgets[group\_box]["answers"] = ***[***

info for info in answer\_info\_list if info["layout"] != answer\_layout***]***

self.updateAnswerOptionNumbers(group\_box***)***

def updateAnswerOptionNumbers(self, group\_box)***:***

answers\_list = self.answer\_widgets[group\_box]["answers"***]***

for i, answer\_info in enumerate(answers\_list, start=1)***:***

answer\_label = answer\_info["layout"].itemAt(1).widget(***)***

if isinstance(answer\_label, QLabel)***:***

answer\_label.setText(f"Ответ {i}:"***)***

def removeQuestion(self, group\_box)***:***

self.questionsLayout.removeWidget(group\_box***)***

group\_box.deleteLater(***)***

if group\_box in self.answer\_widgets***:***

for answer\_info in self.answer\_widgets[group\_box]["answers"]***:***

while answer\_info["layout"].count()***:***

item = answer\_info["layout"].takeAt(0***)***

if item.widget()***:***

item.widget().deleteLater(***)***

del self.answer\_widgets[group\_box***]***

self.updateQuestionCounter(***)***

def updateQuestionCounter(self)***:***

question\_count = len(self.questionsLayout.children()***)***

self.question\_counter\_label.setText(f"�'опросы: {question\_count}"***)***

def createTemplateTab(self)***:***

templateTab = QWidget(***)***

templateLayout = QVBoxLayout(templateTab***)***

templateLabel = QLabel("�'кладка «Шаблон» (пустая)", templateTab***)***

templateLayout.addWidget(templateLabel***)***

self.tabWidget.addTab(templateTab, "Шаблон"***)***

class TakeTestPage(QWidget)***:***

def \_\_init\_\_(self, test\_id, main\_window, student\_window)***:***

super().\_\_init\_\_(***)***

self.test\_id = test\_i***d***

self.main\_window = main\_windo***w***

self.student\_window = student\_windo***w***

self.test\_submitted = Fals***e***

self.test\_details = database.get\_test\_details(self.test\_id***)***

self.test\_name = self.test\_details["name"***]***

self.initUI(***)***

def initUI(self)***:***

layout = QVBoxLayout(self***)***

label = QLabel(f"Тест: {self.test\_name}", self***)***

layout.addWidget(label***)***

self.scrollArea = QScrollArea(self***)***

self.questionsWidget = QWidget(***)***

self.questionsLayout = QVBoxLayout(self.questionsWidget***)***

self.scrollArea.setWidget(self.questionsWidget***)***

self.scrollArea.setWidgetResizable(True***)***

layout.addWidget(self.scrollArea***)***

submit\_button = QPushButton("Отправить тест"***)***

submit\_button.clicked.connect(self.submitTest***)***

layout.addWidget(submit\_button***)***

self.loadTest(***)***

def loadTest(self)***:***

self.test\_details = database.get\_test\_details(self.test\_id***)***

for question in self.test\_details["questions"]***:***

self.addQuestion(question***)***

def addQuestion(self, question)***:***

group\_box = QGroupBox(***)***

group\_box\_layout = QVBoxLayout(group\_box***)***

if question.get("image\_path")***:***

image\_label = QLabel(***)***

image\_label.setAlignment(Qt.AlignCenter***)***

pixmap = QPixmap(question["image\_path"]***)***

image\_label.setPixmap(pixmap***)***

image\_label.setFixedSize(200, 200***)***

group\_box\_layout.addWidget(image\_label***)***

question\_label = QLabel(question["text"]***)***

group\_box\_layout.addWidget(question\_label***)***

answer\_widgets = [***]***

for answer in question["answers"]***:***

if question["type"] == "Единственный правильный ответ"***:***

answer\_widget = QRadioButton(answer["text"]***)***

else***:***

answer\_widget = QCheckBox(answer["text"]***)***

answer\_widget.setProperty("answer\_id", answer["id"]***)***

group\_box\_layout.addWidget(answer\_widget***)***

answer\_widgets.append(answer\_widget***)***

question["answer\_widgets"] = answer\_widget***s***

self.questionsLayout.addWidget(group\_box***)***

def submitTest(self)***:***

answers = [***]***

for question in self.test\_details["questions"]***:***

selected\_answers = [***]***

for widget in question["answer\_widgets"]***:***

if widget.isChecked()***:***

selected\_answers.append(widget.property("answer\_id")***)***

if not selected\_answers***:***

selected\_answers.append(None***)***

answers.append***(***

{"question\_id": question["id"], "selected\_answers": selected\_answers}***)***

self.sendTestResults(answers***)***

self.updateTestAttempts(***)***

self.test\_submitted = Tru***e***

self.student\_window.sidebar.setEnabled(True***)***

self.switchToMainMenu(***)***

def switchToMainMenu(self)***:***

self.student\_window.stack.setCurrentWidget***(***

self.student\_window.tests\_page***)***

def updateTestAttempts(self)***:***

student\_id = self.main\_window.user\_i***d***

test\_id = self.test\_i***d***

attempts\_left = database.get\_remaining\_attempts(student\_id, test\_id***)***

if attempts\_left <= 0***:***

QMessageBox.information***(***

self, "Тест завершен", "�'ы исчерпали все попытки для этого теста."***)***

def sendTestResults(self, answers)***:***

student\_id = self.main\_window.user\_i***d***

test\_id = self.test\_details["id"***]***

database.record\_test\_results(student\_id, test\_id, answers***)***

QMessageBox.information(self, "Тест отправлен"***,***

"�'аши ответы были успешно сохранены."***)***

def closeEvent(self, event)***:***

if not self.test\_submitted***:***

self.submitTest(***)***

self.student\_window.sidebar.setEnabled(True***)***

event.accept(***)***

# interfaces\tests\_page.py

**from** PyQt5.QtWidgets **import** (QWidget, QPushButton, QVBoxLayout,

QHBoxLayout, QTableWidget, QTableWidgetItem, QHeaderView, QMessageBox)

**from** PyQt5.QtCore **import** Qt

**import** database

**from** .test\_page **import** CreateTestPage, ViewTestPage

**class** TestsPage(QWidget):

**def** \_\_init\_\_(self, admin\_window):

super().\_\_init\_\_()

self.admin\_window = admin\_window

self.current\_tab\_index = 0

self.initUI()

**def** initUI(self):

layout = QVBoxLayout(self)

self.tableWidget = QTableWidget(self)

self.tableWidget.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)

self.tableWidget.setColumnCount(3)

self.tableWidget.setHorizontalHeaderLabels(

["Название Теста", "Описание", "Сделан"])

self.tableWidget.horizontalHeader().setStretchLastSection(True)

self.tableWidget.cellChanged.connect(self.onDescriptionEdited)

self.load\_tests()

layout.addWidget(self.tableWidget)

button\_layout = QHBoxLayout()

create\_button = QPushButton("Создать тест", self)

create\_button.clicked.connect(self.create\_test)

button\_layout.addWidget(create\_button)

view\_button = QPushButton("Посмотреть тест", self)

view\_button.clicked.connect(self.view\_test)

button\_layout.addWidget(view\_button)

delete\_button = QPushButton("Удалить тест", self)

delete\_button.clicked.connect(self.delete\_test)

button\_layout.addWidget(delete\_button)

refresh\_button = QPushButton("Обновить данные", self)

refresh\_button.clicked.connect(self.refresh\_tests)

button\_layout.addWidget(refresh\_button)

layout.addLayout(button\_layout)

**def** view\_test(self):

selected\_rows = self.tableWidget.selectedItems()

**if** **not** selected\_rows:

QMessageBox.warning(self, "Ошибка выбора",

"Пожалуйста, выберите тест для просмотра.")

**return**

row = selected\_rows[0].row()

test\_id = self.tableWidget.item(row, 0).data(Qt.UserRole)

view\_test\_page = ViewTestPage(self, test\_id, self.admin\_window)

self.admin\_window.stack.addWidget(view\_test\_page)

self.admin\_window.stack.setCurrentWidget(view\_test\_page)

**def** load\_tests(self):

tests = self.loadTest()

self.tableWidget.setRowCount(len(tests))

**for** row, test **in** enumerate(tests):

self.setupTestRow(row, test)

**def** setupTestRow(self, row, test):

name\_item = QTableWidgetItem(test["name"])

name\_item.setData(Qt.UserRole, test["id"])

name\_item.setFlags(name\_item.flags() & ~Qt.ItemIsEditable)

self.tableWidget.setItem(row, 0, name\_item)

description\_item = QTableWidgetItem(test["description"])

self.tableWidget.setItem(row, 1, description\_item)

created\_by\_item = QTableWidgetItem(test["created\_by"])

created\_by\_item.setFlags(created\_by\_item.flags() & ~Qt.ItemIsEditable)

self.tableWidget.setItem(row, 2, created\_by\_item)

**def** onDescriptionEdited(self, row, column):

**if** column == 1:

test\_id = self.tableWidget.item(row, 0).data(Qt.UserRole)

new\_description = self.tableWidget.item(row, column).text()

database.updateTestDescription(test\_id, new\_description)

**def** loadTest(self):

test\_details = None

**if** self.admin\_window.main\_window.user\_role == "ADMIN":

test\_details = database.get\_all\_tests()

**elif** self.admin\_window.main\_window.user\_role == "TEACHER":

test\_details = database.get\_tests\_by\_teacher\_id(

self.admin\_window.main\_window.user\_id)

**return** test\_details

**def** create\_test(self):

self.current\_tab\_index = self.admin\_window.stack.currentIndex()

logged\_in\_user\_id = self.admin\_window.main\_window.logged\_in\_user\_id

create\_test\_dialog = CreateTestPage(

self, logged\_in\_user\_id, self.refresh\_tests)

self.create\_test\_dialog = create\_test\_dialog

self.admin\_window.stack.addWidget(create\_test\_dialog)

self.admin\_window.stack.setCurrentWidget(create\_test\_dialog)

**def** return\_to\_previous\_tab(self):

self.admin\_window.stack.setCurrentIndex(self.current\_tab\_index)

**def** delete\_test(self):

selected\_rows = self.tableWidget.selectedItems()

**if** **not** selected\_rows:

**return**

row = selected\_rows[0].row()

test\_id = self.tableWidget.item(row, 0).data(Qt.UserRole)

database.delete\_test(test\_id)

self.tableWidget.removeRow(row)

**def** refresh\_tests(self):

self.load\_tests()

# database.py

**import** sqlite3

**def** create\_connection():

**return** sqlite3.connect("test\_management.db")

**def** setup\_database():

conn = create\_connection()

**with** conn:

conn.execute(

"""

CREATE TABLE IF NOT EXISTS users (

id INTEGER PRIMARY KEY AUTOINCREMENT,

name TEXT NOT NULL,

username TEXT NOT NULL UNIQUE,

password TEXT NOT NULL,

role TEXT NOT NULL

);

"""

)

conn.execute(

"""

CREATE TABLE IF NOT EXISTS groups (

id INTEGER PRIMARY KEY AUTOINCREMENT,

name TEXT NOT NULL UNIQUE

);

"""

)

conn.execute(

"""

CREATE TABLE IF NOT EXISTS tests (

id INTEGER PRIMARY KEY AUTOINCREMENT,

name TEXT NOT NULL,

description TEXT,

total\_marks INTEGER,

attempts INTEGER NOT NULL,

creator\_id INTEGER, -- Foreign key referencing the users table

FOREIGN KEY(creator\_id) REFERENCES users(id)

);

"""

)

conn.execute(

"""

CREATE TABLE IF NOT EXISTS user\_groups (

user\_id INTEGER NOT NULL,

group\_id INTEGER NOT NULL,

PRIMARY KEY(user\_id, group\_id),

FOREIGN KEY(user\_id) REFERENCES users(id),

FOREIGN KEY(group\_id) REFERENCES groups(id)

);

"""

)

conn.execute(

"""

CREATE TABLE IF NOT EXISTS student\_tests (

id INTEGER PRIMARY KEY AUTOINCREMENT,

student\_id INTEGER NOT NULL,

test\_id INTEGER NOT NULL,

assigner\_id INTEGER NOT NULL,

remaining\_attempts INTEGER,

FOREIGN KEY(student\_id) REFERENCES users(id),

FOREIGN KEY(test\_id) REFERENCES tests(id),

FOREIGN KEY(assigner\_id) REFERENCES users(id),

UNIQUE(student\_id, test\_id)

);

"""

)

conn.execute(

"""

CREATE TABLE IF NOT EXISTS questions (

id INTEGER PRIMARY KEY AUTOINCREMENT,

test\_id INTEGER NOT NULL,

text TEXT NOT NULL,

type TEXT NOT NULL,

FOREIGN KEY(test\_id) REFERENCES tests(id)

);

"""

)

conn.execute(

"""

CREATE TABLE IF NOT EXISTS answers (

id INTEGER PRIMARY KEY AUTOINCREMENT,

question\_id INTEGER NOT NULL,

text TEXT NOT NULL,

is\_correct BOOLEAN NOT NULL,

FOREIGN KEY(question\_id) REFERENCES questions(id)

);

"""

)

conn.execute(

"""

CREATE TABLE IF NOT EXISTS student\_answers (

id INTEGER PRIMARY KEY AUTOINCREMENT,

test\_results\_id INTEGER NOT NULL,

question\_id INTEGER NOT NULL,

selected\_answer INTEGER NOT NULL,

FOREIGN KEY(test\_results\_id) REFERENCES test\_results(id),

FOREIGN KEY(question\_id) REFERENCES questions(id)

);

"""

)

conn.execute(

"""

CREATE TABLE IF NOT EXISTS test\_results (

id INTEGER PRIMARY KEY AUTOINCREMENT,

student\_id INTEGER NOT NULL,

test\_id INTEGER NOT NULL,

FOREIGN KEY(student\_id) REFERENCES users(id),

FOREIGN KEY(test\_id) REFERENCES tests(id)

);

"""

)

conn.execute(

"INSERT OR IGNORE INTO users (name, username, password, role) VALUES ('Admin User', 'admin', 'admin', 'ADMIN')"

)

conn.execute(

"INSERT OR IGNORE INTO users (name, username, password, role) VALUES ('Teacher User', 'teacher', 'teacher', 'TEACHER')"

)

conn.execute(

"INSERT OR IGNORE INTO users (name, username, password, role) VALUES ('Student User', 'student', 'student', 'STUDENT')"

)

**def** authenticate\_user(username, password):

conn = create\_connection()

**with** conn:

cursor = conn.cursor()

cursor.execute(

"SELECT role FROM users WHERE username=? AND password=?",

(username, password),

)

**return** cursor.fetchone()

**def** get\_all\_users():

conn = create\_connection()

**with** conn:

cursor = conn.cursor()

cursor.execute("SELECT id, name, username, password, role FROM users")

**return** cursor.fetchall()

**def** execute\_query(query, args=None):

conn = create\_connection()

**with** conn:

cursor = conn.cursor()

**if** args:

cursor.execute(query, args)

**else**:

cursor.execute(query)

conn.commit()

**return** cursor.fetchall()

**def** add\_user(name, username, password, role):

query = "INSERT INTO users (name, username, password, role) VALUES (?, ?, ?, ?)"

args = (name, username, password, role)

execute\_query(query, args)

**def** delete\_user(username):

query = "DELETE FROM users WHERE username = ?"

args = (username,)

execute\_query(query, args)

**def** update\_name(username, name):

query = "UPDATE users SET name = ? WHERE username = ?"

args = (name, username)

execute\_query(query, args)

**def** update\_role(username, new\_role):

query = "UPDATE users SET role = ? WHERE username = ?"

args = (new\_role, username)

execute\_query(query, args)

**def** add\_group(name):

query = "INSERT INTO groups (name) VALUES (?)"

args = (name,)

execute\_query(query, args)

**def** delete\_group(name):

query = "DELETE FROM groups WHERE name = ?"

args = (name,)

execute\_query(query, args)

**def** get\_all\_groups():

query = "SELECT id, name FROM groups"

conn = create\_connection()

**with** conn:

cursor = conn.cursor()

cursor.execute(query)

groups = [{"id": row[0], "name": row[1]} **for** row **in** cursor.fetchall()]

**return** groups

**def** get\_tests\_for\_student(student\_id):

query = """

SELECT test\_id FROM student\_tests WHERE student\_id = ?

"""

args = (student\_id,)

**return** [row[0] **for** row **in** execute\_query(query, args)]

**def** assign\_test\_to\_student(test\_id, student\_id, assigner\_id):

conn = create\_connection()

**with** conn:

cursor = conn.cursor()

cursor.execute("SELECT attempts FROM tests WHERE id = ?", (test\_id,))

result = cursor.fetchone()

attempts = result[0]

query = """

INSERT INTO student\_tests (student\_id, test\_id, assigner\_id, remaining\_attempts)

VALUES (?, ?, ?, ?)

ON CONFLICT(student\_id, test\_id) DO UPDATE SET remaining\_attempts = ?

"""

args = (student\_id, test\_id, assigner\_id, attempts, attempts)

cursor.execute(query, args)

conn.commit()

**def** authenticate\_user(username, password):

conn = create\_connection()

**with** conn:

cursor = conn.cursor()

cursor.execute(

"SELECT id, role FROM users WHERE username=? AND password=?",

(username, password),

)

**return** cursor.fetchone()

**def** get\_all\_students():

conn = create\_connection()

**with** conn:

cursor = conn.cursor()

cursor.execute(

"""

SELECT u.id, u.name, u.username, u.role, g.name as group\_name, GROUP\_CONCAT(t.name) as tests

FROM users u

LEFT JOIN user\_groups ug ON u.id = ug.user\_id

LEFT JOIN groups g ON ug.group\_id = g.id

LEFT JOIN student\_tests st ON u.id = st.student\_id

LEFT JOIN tests t ON st.test\_id = t.id

WHERE u.role = 'STUDENT'

GROUP BY u.id

"""

)

students = [

{

"id": row[0],

"name": row[1],

"username": row[2],

"role": row[3],

"group": row[4],

"tests": row[5],

}

**for** row **in** cursor.fetchall()

]

**return** students

**def** get\_all\_users\_as\_dicts():

conn = create\_connection()

**with** conn:

cursor = conn.cursor()

cursor.execute("SELECT id, name, username, password, role FROM users")

users = [

{

"id": row[0],

"name": row[1],

"username": row[2],

"password": row[3],

"role": row[4],

}

**for** row **in** cursor.fetchall()

]

**return** users

**def** get\_all\_tests\_as\_dict():

query = "SELECT id, name FROM tests"

**return** execute\_query(query)

# -----------------------

**def** set\_student\_group(student\_id, group\_id):

conn = create\_connection()

**with** conn:

cursor = conn.cursor()

cursor.execute(

"""

INSERT OR REPLACE INTO user\_groups (user\_id, group\_id) VALUES (?, ?)

""",

(student\_id, group\_id),

)

conn.commit()

**def** set\_student\_group(student\_id, group\_id):

conn = create\_connection()

**with** conn:

cursor = conn.cursor()

cursor.execute(

"SELECT \* FROM user\_groups WHERE user\_id = ?", (student\_id,))

existing\_group = cursor.fetchone()

**if** existing\_group **is** None:

cursor.execute(

"INSERT INTO user\_groups (user\_id, group\_id) VALUES (?, ?)",

(student\_id, group\_id),

)

**else**:

cursor.execute(

"UPDATE user\_groups SET group\_id = ? WHERE user\_id = ?",

(group\_id, student\_id),

)

conn.commit()

**def** remove\_test\_assignment\_from\_group(test\_id, group\_id):

conn = create\_connection()

**with** conn:

cursor = conn.cursor()

cursor.execute(

"""

DELETE FROM student\_tests

WHERE test\_id = ? AND student\_id IN (

SELECT user\_id FROM user\_groups WHERE group\_id = ?

)

""",

(test\_id, group\_id),

)

conn.commit()

**def** assign\_test\_to\_group\_students(test\_id, group\_id, assigner\_id):

conn = create\_connection()

**with** conn:

cursor = conn.cursor()

cursor.execute("SELECT attempts FROM tests WHERE id = ?", (test\_id,))

result = cursor.fetchone()

attempts = result[0]

query = """

INSERT INTO student\_tests (student\_id, test\_id, assigner\_id, remaining\_attempts)

SELECT user\_id, ?, ?, ?

FROM user\_groups WHERE group\_id = ?

ON CONFLICT(student\_id, test\_id) DO UPDATE SET remaining\_attempts = ?

"""

args = (test\_id, assigner\_id, attempts, group\_id, attempts)

cursor.execute(query, args)

conn.commit()

**def** get\_teachers\_tests():

query = """

SELECT u.name AS teacher\_name, GROUP\_CONCAT(t.name) AS tests

FROM users u

LEFT JOIN tests t ON u.id = t.creator\_id

WHERE u.role = 'TEACHER'

GROUP BY u.name

"""

conn = create\_connection()

**with** conn:

cursor = conn.cursor()

cursor.execute(query)

teachers\_tests = [

{"teacher\_name": row[0], "tests": row[1]} **for** row **in** cursor.fetchall()

]

**return** teachers\_tests

**def** get\_tests\_by\_teacher\_id(teacher\_id):

query = """

SELECT t.id, t.name, t.description, t.total\_marks, t.attempts, u.name as created\_by

FROM tests t

LEFT JOIN users u ON t.creator\_id = u.id

WHERE t.creator\_id = ?

"""

args = (teacher\_id,)

conn = create\_connection()

**with** conn:

cursor = conn.cursor()

cursor.execute(query, args)

tests = [

dict(zip([column[0] **for** column **in** cursor.description], row))

**for** row **in** cursor.fetchall()

]

**return** tests

**def** get\_all\_tests():

query = """

SELECT t.id, t.name, t.description, t.total\_marks, t.attempts, u.name as created\_by

FROM tests t

LEFT JOIN users u ON t.creator\_id = u.id

"""

conn = create\_connection()

**with** conn:

cursor = conn.cursor()

cursor.execute(query)

tests = [

dict(zip([column[0] **for** column **in** cursor.description], row))

**for** row **in** cursor.fetchall()

]

**return** tests

**def** updateTestDescription(test\_id, new\_description):

query = "UPDATE tests SET description = ? WHERE id = ?"

args = (new\_description, test\_id)

execute\_query(query, args)

**def** get\_user\_info(user\_id):

query = "SELECT name, username, password FROM users WHERE id = ?"

args = (user\_id,)

conn = create\_connection()

**with** conn:

cursor = conn.cursor()

cursor.execute(query, args)

user\_info = cursor.fetchone()

**return** (

{"name": user\_info[0], "username": user\_info[1],

"password": user\_info[2]}

**if** user\_info

**else** None

)

**def** update\_user\_info(user\_id, name, username, password):

query = "UPDATE users SET name = ?, username = ?, password = ? WHERE id = ?"

args = (name, username, password, user\_id)

execute\_query(query, args)

**def** get\_assigned\_tests\_for\_student(student\_id):

query = """

SELECT

t.id, t.name, t.description, t.total\_marks, t.attempts,

creator.name as creator\_name, assigner.name as assigner\_name,

st.remaining\_attempts

FROM student\_tests st

JOIN tests t ON st.test\_id = t.id

LEFT JOIN users creator ON t.creator\_id = creator.id

LEFT JOIN users assigner ON st.assigner\_id = assigner.id

WHERE st.student\_id = ?

"""

args = (student\_id,)

conn = create\_connection()

tests = []

**with** conn:

cursor = conn.cursor()

cursor.execute(query, args)

tests = [

dict(zip([column[0] **for** column **in** cursor.description], row))

**for** row **in** cursor.fetchall()

]

**return** tests

**def** remove\_test\_from\_student(test\_id, student\_id):

conn = create\_connection()

**with** conn:

cursor = conn.cursor()

query = "DELETE FROM student\_tests WHERE student\_id = ? AND test\_id = ?"

args = (student\_id, test\_id)

cursor.execute(query, args)

conn.commit()

**def** save\_test\_to\_database(test\_name, attempts, questions, creator\_id):

conn = create\_connection()

**with** conn:

cursor = conn.cursor()

cursor.execute(

"INSERT INTO tests (name, attempts, creator\_id) VALUES (?, ?, ?)",

(test\_name, attempts, creator\_id),

)

test\_id = cursor.lastrowid

**for** question **in** questions:

cursor.execute(

"INSERT INTO questions (test\_id, text, type) VALUES (?, ?, ?)",

(test\_id, question["text"], question["type"]),

)

question\_id = cursor.lastrowid

**for** answer **in** question["answers"]:

cursor.execute(

"INSERT INTO answers (question\_id, text, is\_correct) VALUES (?, ?, ?)",

(question\_id, answer["text"], answer["is\_correct"]),

)

conn.commit()

**def** delete\_test(test\_id):

query = "DELETE FROM tests WHERE id = ?"

args = (test\_id,)

execute\_query(query, args)

**def** get\_group\_id\_by\_name(group\_name):

conn = create\_connection()

**with** conn:

cursor = conn.cursor()

cursor.execute("SELECT id FROM groups WHERE name = ?", (group\_name,))

result = cursor.fetchone()

**return** result[0] **if** result **else** None

**def** get\_tests\_by\_teacher(teacher\_id):

query = """

SELECT t.id, t.name

FROM tests t

JOIN users u ON t.creator\_id = u.id

WHERE u.id = ?

"""

args = (teacher\_id,)

conn = create\_connection()

**with** conn:

cursor = conn.cursor()

cursor.execute(query, args)

**return** [{"id": row[0], "name": row[1]} **for** row **in** cursor.fetchall()]

**def** get\_test\_details(test\_id):

query = """

SELECT t.id, t.name, t.attempts, t.creator\_id,

q.id, q.text, q.type,

a.id, a.text, a.is\_correct

FROM tests t

LEFT JOIN questions q ON t.id = q.test\_id

LEFT JOIN answers a ON q.id = a.question\_id

WHERE t.id = ?

ORDER BY q.id, a.id

"""

args = (test\_id,)

conn = create\_connection()

**with** conn:

cursor = conn.cursor()

cursor.execute(query, args)

rows = cursor.fetchall()

**if** **not** rows:

**return** None

test\_info = {

"id": None,

"name": None,

"attempts": None,

"creator\_id": None,

"questions": [],

}

question = None

**for** row **in** rows:

**if** test\_info["id"] **is** None:

test\_info["id"] = row[0]

test\_info["name"] = row[1]

test\_info["attempts"] = row[2]

test\_info["creator\_id"] = row[3]

**if** question **is** None **or** question["id"] != row[4]:

question = {"id": row[4], "text": row[5],

"type": row[6], "answers": []}

test\_info["questions"].append(question)

**if** row[7] **is** **not** None:

answer = {"id": row[7], "text": row[8], "is\_correct": row[9]}

question["answers"].append(answer)

**return** test\_info

**def** record\_test\_results(student\_id, test\_id, answers):

conn = create\_connection()

**with** conn:

cursor = conn.cursor()

cursor.execute(

"SELECT id, remaining\_attempts FROM student\_tests WHERE student\_id = ? AND test\_id = ?",

(student\_id, test\_id)

)

result = cursor.fetchone()

**if** result:

student\_test\_id, remaining\_attempts = result

**if** remaining\_attempts > 0:

cursor.execute(

"INSERT INTO test\_results (student\_id, test\_id) VALUES (?, ?)",

(student\_id, test\_id)

)

test\_results\_id = cursor.lastrowid

**for** question **in** answers:

question\_id = question["question\_id"]

selected\_answers = question["selected\_answers"]

**for** answer\_id **in** selected\_answers:

cursor.execute(

"INSERT INTO student\_answers (test\_results\_id, question\_id, selected\_answer) VALUES (?, ?, ?)",

(test\_results\_id, question\_id, answer\_id),

)

cursor.execute(

"UPDATE student\_tests SET remaining\_attempts = remaining\_attempts - 1 WHERE id = ?",

(student\_test\_id,),

)

conn.commit()

**def** get\_remaining\_attempts(student\_id, test\_id):

query = """

SELECT remaining\_attempts FROM student\_tests

WHERE student\_id = ? AND test\_id = ?

"""

args = (student\_id, test\_id)

conn = create\_connection()

**with** conn:

cursor = conn.cursor()

cursor.execute(query, args)

result = cursor.fetchone()

**return** result[0] **if** result **else** None

**def** get\_tests\_by\_student(student\_id):

query = """

SELECT DISTINCT t.id, t.name

FROM tests t

JOIN test\_results tr ON t.id = tr.test\_id

WHERE tr.student\_id = ?

"""

conn = create\_connection()

**with** conn:

cursor = conn.cursor()

cursor.execute(query, (student\_id,))

**return** [{"id": row[0], "name": row[1]} **for** row **in** cursor.fetchall()]

**def** get\_student\_test\_attempt\_results(student\_id, test\_id):

query = """

SELECT tr.id, a.is\_correct, sa.selected\_answer

FROM student\_answers sa

JOIN answers a ON sa.question\_id = a.question\_id AND sa.selected\_answer = a.id

JOIN test\_results tr ON sa.test\_results\_id = tr.id

WHERE tr.student\_id = ? AND tr.test\_id = ?

"""

args = (student\_id, test\_id)

conn = create\_connection()

**with** conn:

cursor = conn.cursor()

cursor.execute(query, args)

**return** cursor.fetchall()

# main.py

**from** PyQt5.QtWidgets **import** QApplication, QMainWindow, QStackedWidget

**import** sys

**from** interfaces **import** LoginWindow, AdminWindow, TeacherWindow, StudentWindow

**import** database

**def** load\_stylesheet(path):

**with** open(path, "r") **as** f:

**return** f.read()

**class** MainWindow(QMainWindow):

**def** \_\_init\_\_(self):

super().\_\_init\_\_()

self.logged\_in\_user\_id = None

self.user\_role = None

self.user\_id = None

self.central\_widget = QStackedWidget()

self.setCentralWidget(self.central\_widget)

style = load\_stylesheet("resources/styles/main\_style.css")

self.setStyleSheet(style)

self.login\_window = LoginWindow(self)

self.central\_widget.addWidget(self.login\_window)

self.setGeometry(300, 300, 1024, 768)

self.setWindowTitle("Система управления тестами и результатами")

self.central\_widget.setCurrentWidget(self.login\_window)

**def** initAdminWindow(self):

self.admin\_window = AdminWindow(self)

self.central\_widget.addWidget(self.admin\_window)

**def** initTeacherWindow(self):

self.teacher\_window = TeacherWindow(self)

self.central\_widget.addWidget(self.teacher\_window)

**def** initStudentWindow(self):

self.student\_window = StudentWindow(self)

self.central\_widget.addWidget(self.student\_window)

**if** \_\_name\_\_ == "\_\_main\_\_":

database.setup\_database()

app = QApplication(sys.argv)

main\_window = MainWindow()

main\_window.show()

sys.exit(app.exec\_())