

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 обычных пользователей.

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

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	4
1. ПРЕДМЕТНАЯ ОБЛАСТЬ .....	5
1.1. Анализ предметной области.....	5
1.2. Обзор аналогов .....	5
2. ТЕХНИЧЕСКОЕ ЗАДАНИЕ.....	8
3. РЕАЛИЗАЦИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ .....	9
3.1. Моделирование ИС.....	9
3.1.1. Функциональное моделирование IDEF0 .....	9
3.1.2. Моделирование потоков данных DFD.....	11
3.1.3. Моделирование в нотации UML .....	12
3.2. Разработка интерфейса.....	14
3.3. Разработка базы данных .....	21
3.3.1. Описание сущностей и атрибутов .....	21
3.3.2. Логическая модель данных в нотации IDEF1X .....	24
3.3.3. Физическая модель данных.....	26
ЗАКЛЮЧЕНИЕ .....	27
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	28
ПРИЛОЖЕНИЕ А (ТЕХНИЧЕСКОЕ ЗАДАНИЕ) .....	29
ПРИЛОЖЕНИЕ Б (ИСХОДНЫЙ КОД).....	30

					<b>ТПЖА 09.02.07 443767 ПЗ</b>			
<b>Изм.</b>	<b>Лист</b>	<b>№ докум.</b>	<b>Подпись</b>	<b>Дата</b>				
Разраб.					Тема курсовой работы	Лит.	Лист	Листов
Провер.							2	29
Реценз.						Колледж ВятГУ группа		
Н. Контр.								
Утверд.								

## ВВЕДЕНИЕ

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 обычных 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.

					<b>ТПЖА 09.02.07 443767 ПЗ</b>	Лист
						3
Изм.	Лист	№ докум.	Подпись	Дата		

# 1. ПРЕДМЕТНАЯ ОБЛАСТЬ

## 1.1. 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.

## 1.2. 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.

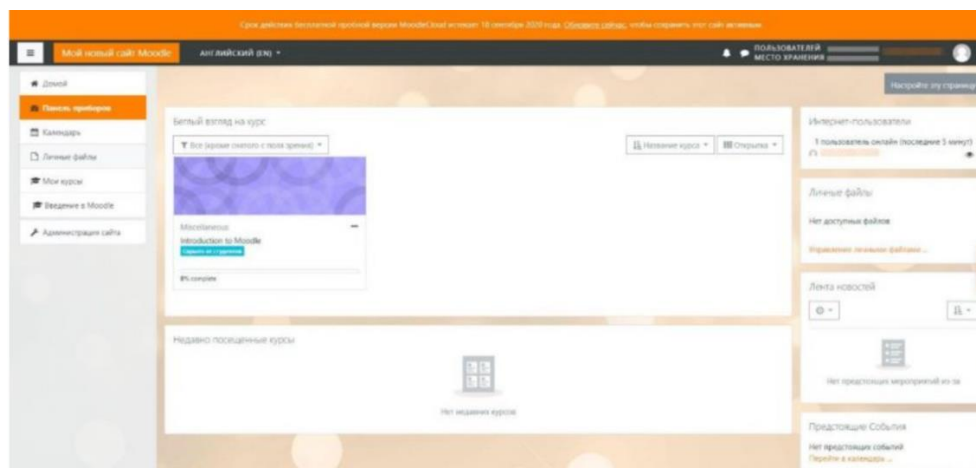


Рисунок 1. – онлайн сервис Moodle

					ТПЖА 09.02.07 443767 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		4

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.

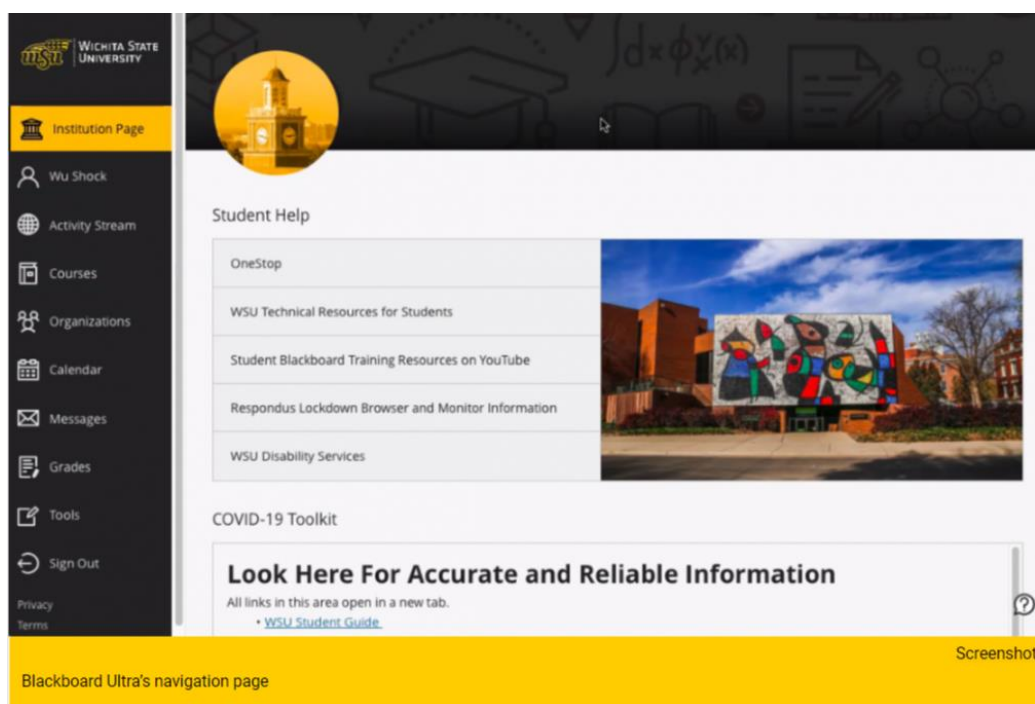


Рисунок 2. – онлайн сервис Blackboard

					ТПЖА 09.02.07 443767 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		5

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.

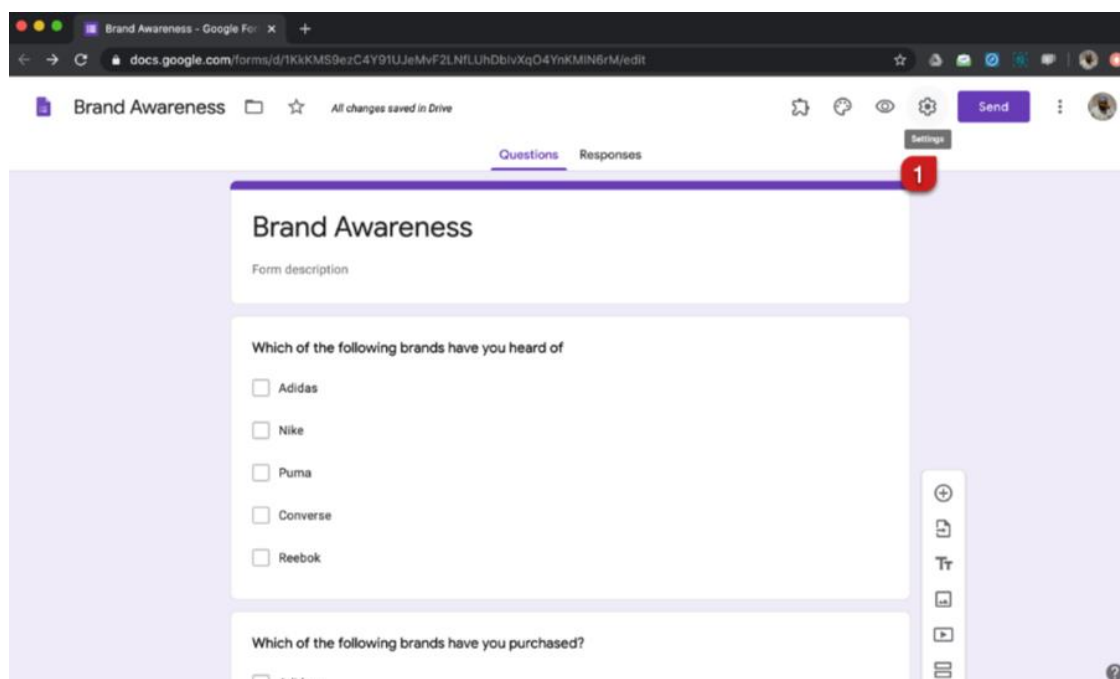


Рисунок 3. – онлайн сервис Google Forms

					<b>ТПЖА 09.02.07 443767 ПЗ</b>	<b>Лист</b>
<b>Изм.</b>	<b>Лист</b>	<b>№ докум.</b>	<b>Подпись</b>	<b>Дата</b>		<b>6</b>

## 2. ТЕХНИЧЕСКОЕ ЗАДАНИЕ

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.

					<b>ТПЖА 09.02.07 443767 ПЗ</b>	<b>Лист</b>
<b>Изм.</b>	<b>Лист</b>	<b>№ докум.</b>	<b>Подпись</b>	<b>Дата</b>		<b>7</b>



### 3. РЕАЛИЗАЦИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ

#### 3.1. Моделирование ИС

##### 3.1.1. 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.

					<b>ТПЖА 09.02.07 443767 ПЗ</b>	Лист
Изм.	Лист	№ докум.	Подпись	Дата		8

### 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.



Рисунок 4. – Функциональная модель IDEF0

### 3.1.2. Моделирование потоков данных DFD

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.

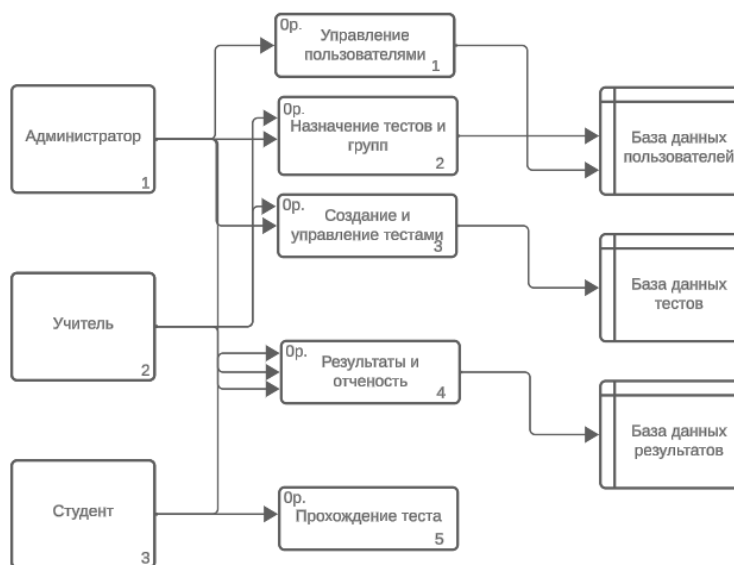


Рисунок 5. – Модель потоков данных DFD

### 3.1.3. 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.

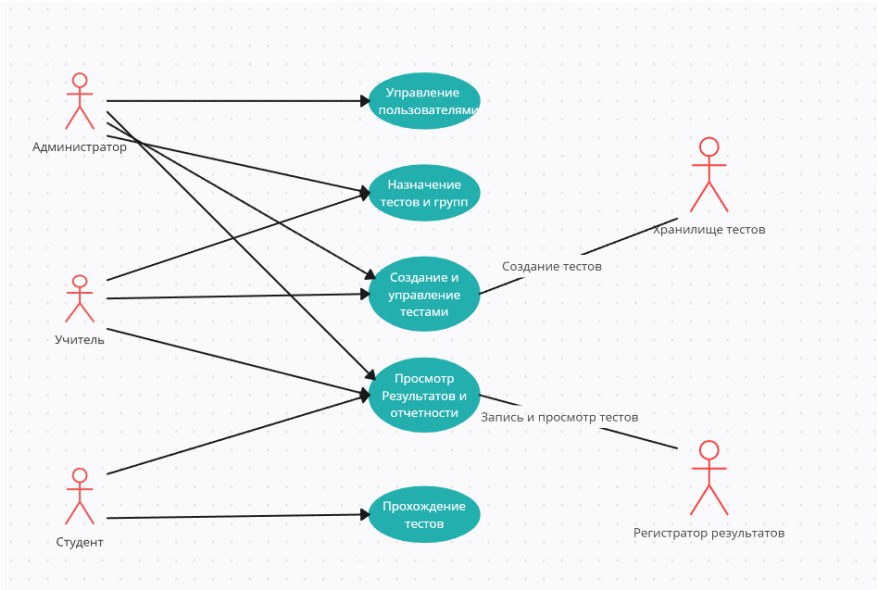


Рисунок 6. – Модель в нотации UML

### 3.2. 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)

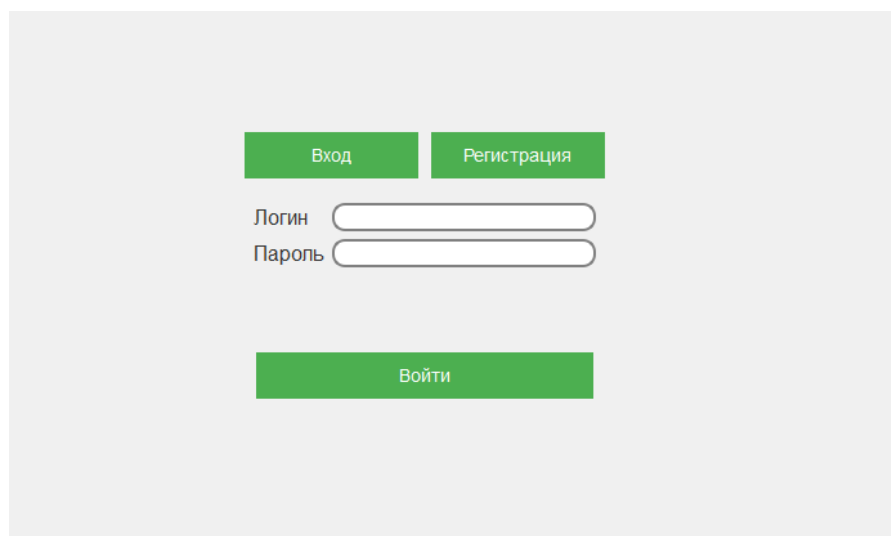


Рисунок 7. – окно «Входа».

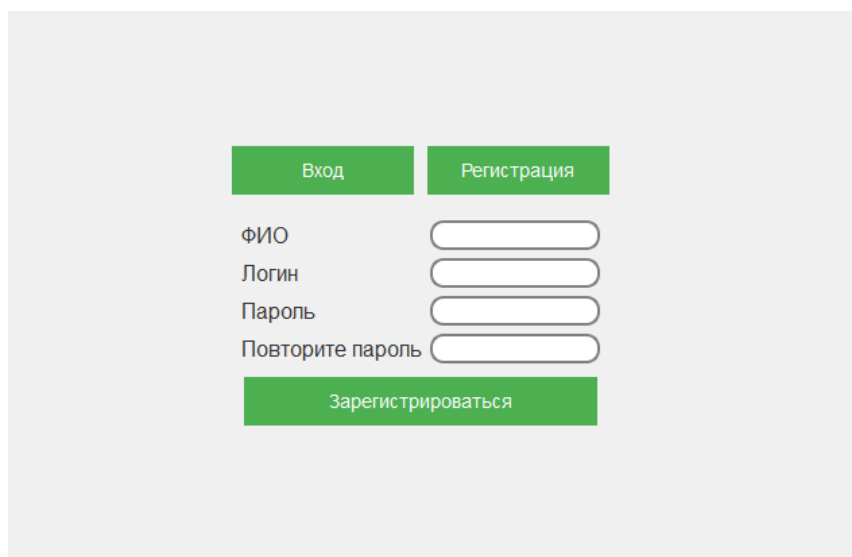


Рисунок 8. – окно «Регистрации».

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»

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.

<div>Пользователи</div> <div>Ученики</div> <div>Учителя</div> <div>Тесты</div> <div>Отчеты</div>		ID	ФИО (Edit)	Логин	Пароль	Роль (Edit)
	1	1	Admin User	admin	*****	ADMIN
	2	2	Teacher User	teacher	*****	TEACHER
	3	3	Student User	student	*****	STUDENT
Мой профиль	Добавить пользователя					
Настройки	Удалить пользователя					
Выйти из аккаунта	Обновить данные					

Рисунок 11. – окно «Управление пользователями».

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

<div>Пользователи</div> <div>Ученики</div> <div>Учителя</div> <div>Тесты</div> <div>Отчеты</div>		ФИО	Группа	Назначенные тесты
	1	Student User		тест
Мой профиль	Управление группами			
Настройки	Назначить тест студенту			
Выйти из аккаунта	Обновить данные			

Рисунок 12. – окно «Управление студентами».



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

	Имя учителя	Созданные тесты
1	Teacher User	

Обновить данные

Рисунок 13. – окно «Учителя».

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

	Название Теста	Описание	Сделан
1	тест		Admin User

Создать тест    Посмотреть тест    Удалить тест    Обновить данные

Рисунок 14. – окно «Тесты».

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

19)

## Меню Учителя

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

Ученики	ФИО	Группа	Назначенные тесты
Тесты	1 Student User		тест
Отчеты			

Мой профиль

Настройки

Выйти из аккаунта

Управление группами

Назначить тест студенту

Обновить данные

Рисунок 15. – окно «Студенты».

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

Ученики	Название Теста	Описание	Сделан
Тесты			
Отчеты			

Мой профиль

Настройки

Выйти из аккаунта

Создать тест

Посмотреть тест

Удалить тест

Обновить данные

Рисунок 16. – окно «Тесты».

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

## Меню Студента

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

	Название	Описание	Количество ост. попыток	Кто создал	Кто назначил
1	тест		1	Admin User	Admin User

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).

Создание теста

Ручное создание | Шаблон

Название теста:

Попытки:

Вопросы: 0 Добавить вопрос

Добавить изображение Удалить изображение Удалить вопрос

Введите сюда текст вашего вопроса...

☐ Единственный правильный ответ

Добавить вариант ответа

Сохранить тест Закрыть редактор

Рисунок 18. – окно «Создание теста».

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

Пользователи

Ученики

Учителя

Тесты

Отчеты

Мой профиль

Настройки

Выйти из аккаунта

Ученики

Student User

Студент1

Студент2

Студент3

Студент4

Студент5

Название теста

тест

Номер попытки

Результат

1

2/2

2

0/2

Просмотреть ответы

Обновить данные

Рисунок 19. – окно «Отчеты».

### 3.3.Разработка базы данных

#### 3.3.1. Описание сущностей и атрибутов

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

Таблица "users":

Атрибут	Тип данных	Описание
id	INTEGER	Уникальный идентификатор пользователя
name	TEXT	Имя пользователя
username	TEXT	Уникальное имя пользователя
password	TEXT	Пароль пользователя
role	TEXT	Роль пользователя (например, администратор, учитель, студент)

Таблица "groups":

Атрибут	Тип данных	Описание
id	INTEGER	Уникальный идентификатор группы
name	TEXT	Уникальное имя группы

Таблица "tests":

Атрибут	Тип данных	Описание
id	INTEGER	Уникальный идентификатор теста
name	TEXT	Название теста
description	TEXT	Описание теста
total_marks	INTEGER	Общее количество баллов за тест
attempts	INTEGER	Количество попыток для прохождения теста
creator_id	INTEGER	Идентификатор создателя теста (ссылка на таблицу "users")

Таблица "user\_groups":

Атрибут	Тип данных	Описание
user_id	INTEGER	Идентификатор пользователя (ссылка на таблицу "users")
group_id	INTEGER	Идентификатор группы (ссылка на таблицу "groups")

Таблица "student\_tests":

Атрибут	Тип данных	Описание
id	INTEGER	Уникальный идентификатор записи о назначении теста студенту
student_id	INTEGER	Идентификатор студента (ссылка на таблицу "users")
test_id	INTEGER	Идентификатор теста (ссылка на таблицу "tests")
assigner_id	INTEGER	Идентификатор пользователя, назначившего тест (ссылка на таблицу "users")
remaining_attempts	INTEGER	Оставшиеся попытки прохождения теста

Таблица "questions":

Атрибут	Тип данных	Описание
id	INTEGER	Уникальный идентификатор вопроса
test_id	INTEGER	Идентификатор теста, к которому относится вопрос (ссылка на таблицу "tests")
text	TEXT	Текст вопроса
type	TEXT	Тип вопроса (например, выбор из вариантов, открытый вопрос)

Таблица "answers":

Атрибут	Тип данных	Описание
id	INTEGER	Уникальный идентификатор ответа
question_id	INTEGER	Идентификатор вопроса, к которому относится ответ (ссылка на таблицу "questions")
text	TEXT	Текст ответа
is_correct	BOOLEAN	Признак правильности ответа

Таблица "student\_answers":

Атрибут	Тип данных	Описание
id	INTEGER	Уникальный идентификатор записи о выборе студентом ответа
test_results_id	INTEGER	Идентификатор результата теста (ссылка на таблицу "test_results")
question_id	INTEGER	Идентификатор вопроса (ссылка на таблицу "questions")
selected_answer	INTEGER	Идентификатор выбранного студентом ответа (ссылка на таблицу "answers")

Таблица "test\_results":

Атрибут	Тип данных	Описание
id	INTEGER	Уникальный идентификатор результата теста
student_id	INTEGER	Идентификатор студента (ссылка на таблицу "users")
test_id	INTEGER	Идентификатор теста (ссылка на таблицу "tests")

### 3.3.2. Логическая модель данных в нотации IDEF1X

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.



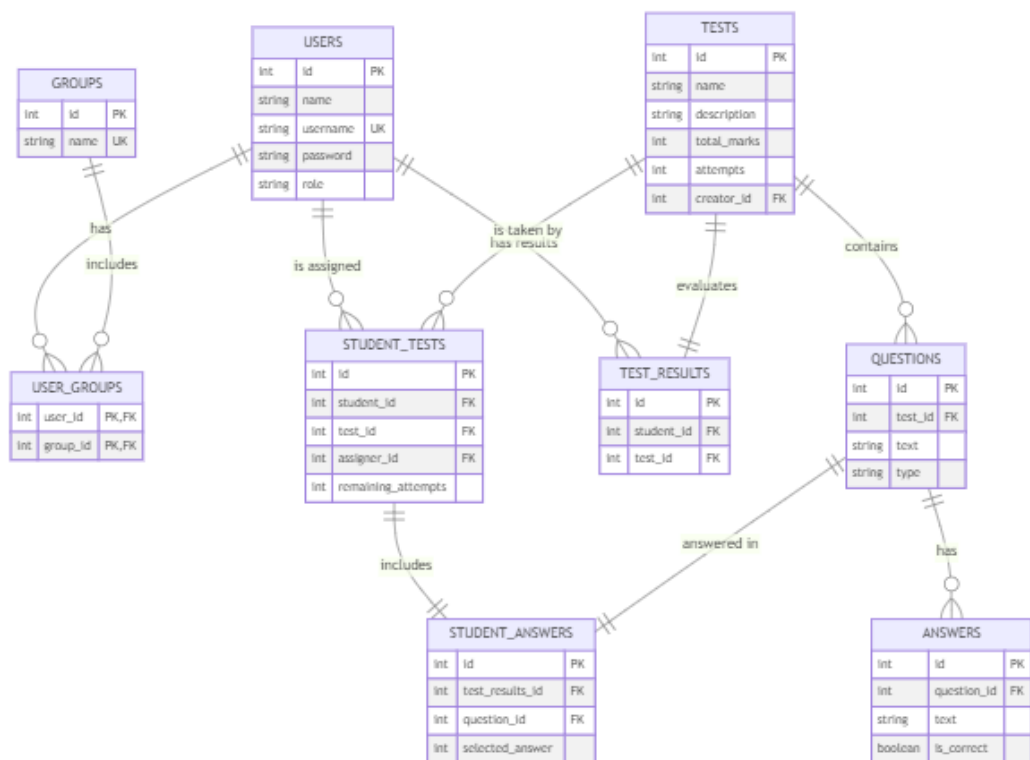


Рисунок 20. – Логическая модель данных

### 3.3.3. 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.

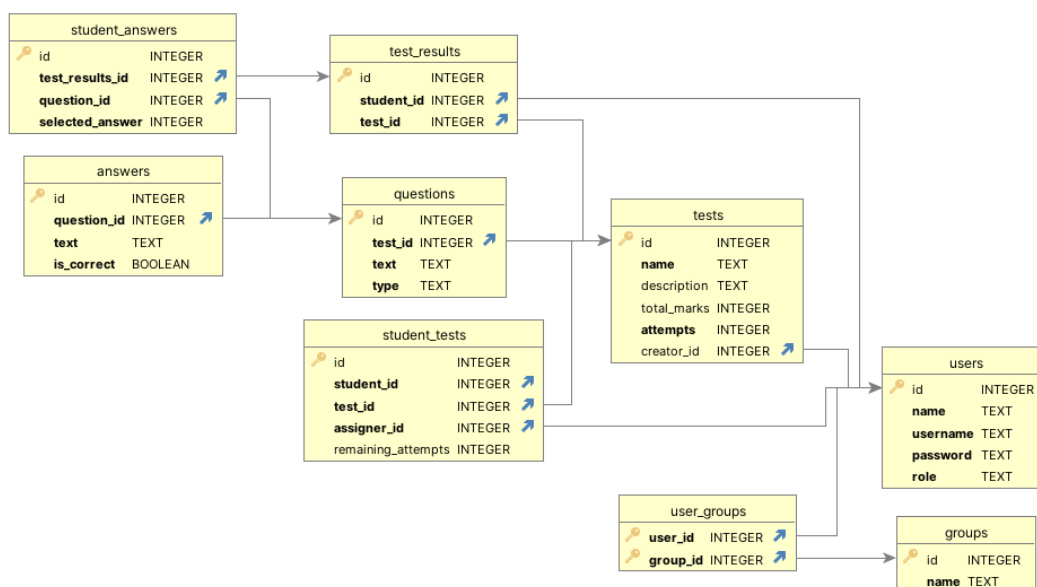


Рисунок 21. – Физическая модель данных

## ЗАКЛЮЧЕНИЕ

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.

					<b>ТПЖА 09.02.07 443767 ПЗ</b>	<i>Лист</i>
<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>		26

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

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)**

					<b>ТПЖА 09.02.07 443767 ПЗ</b>	<i>Лист</i>
						<b>28</b>
<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>		

**ПРИЛОЖЕНИЕ Б**  
**(ИСХОДНЫЙ КОД)**

					<b>ТПЖА 09.02.07 443767 ПЗ</b>	<i>Лист</i>
						<b>29</b>
<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>		

```

# 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("❓" + "добавить нового пользователя")
        self.layout = QFormLayout(self)

        self.name = QLineEdit(self)
        self.username = QLineEdit(self)
        self.password = QLineEdit(self)

```



```

self.role = QComboBox(self)
self.role.addItem("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.addItem(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)
h)

```

```

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'Вы 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 = QTableWidgetItem(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
обновленау.")

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("🔒 Выйти 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(['Вопрос',
'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_window
        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():
            return

        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 = tests
        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 None

class StudentsTableModel(QStandardItemModel):
    def __init__(self, students=None):
        super().__init__()
        self.students = students or []

    def setStudents(self, students):
        self.students = students
        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, "❖" + "группа уже существует", "❖" + "группа 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,
                "❖" + "группа не выбрана",
                "Пожалуйста, выберите группу, чтобы назначить
заданиест.")
            return

        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_data
        self.header = ["ФИО", "❖" + "группа", "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", "Выбранные 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", "Выбранные 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 изображение")
    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.addItem(
        ["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.ReadOnly

    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_type

    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_widget
        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("❖ вкладка «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_id
        self.main_window = main_window
        self.student_window = student_window
        self.test_submitted = False

```

```

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_widgets
    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 = True
    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_id
        test_id = self.test_id

        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_id
        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, 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 = QTableWidgetItem(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
        FOREIGN KEY(creator_id) REFERENCES users(id)
    );
    """
)

table

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_())

```