**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**КЫРГЫЗСКОЙ РЕСПУБЛИКИ**

**КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. И. Раззакова**

**ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

Кафедра: **Программное обеспечение компьютерных систем**

Дисциплина «Объектно-ориентированное программирование»

**КУРСОВАЯ РАБОТА на тему:**

**«Программная система для проверки знаний сотрудников компаний связи»**

Выполнила: студент группы ПИ-2-2 Макенова Арзыгуль,

Проверил: Мусабаев Э.Б.

**Бишкек 2024**

Аннотация

Пояснительная записка содержит 47 страниц , 28 рисунков и 12 таблиц.

В данной пояснительной записки представлены результаты разработки консольного приложения для проверки знаний сотрудников компаний связи.

В главе «Аналитическая часть» приведен анализ предметной области, описан процесс выявления требований и необходимые свойства программного продукта.

В главе «Проектирование» представлены результаты моделирования диаграммы вариантов использования (Use case), диаграммы классов и диаграммы базы данных.

В главе «Стадия реализации» приведены содержание приложение и результаты тестирования консольного приложения.

Оглавление

[Аннотация 2](#_Toc167383433)

[Введение 4](#_Toc167383434)

[ГЛАВА 1. АНАЛИТИЧЕСКАЯ ЧАСТЬ 5](#_Toc167383435)

[1.1. Актуальность проблемы 5](#_Toc167383436)

[1.3. Описание проекта 7](#_Toc167383437)

[1.4. Назначение и цели создания системы 7](#_Toc167383438)

[1.4.1. Назначение 7](#_Toc167383439)

[1.4.2. Цели 7](#_Toc167383440)

[1.4.3. Требования к функциям: 7](#_Toc167383441)

[ГЛАВА 2. ПРОЕКТНАЯ ЧАСТЬ 8](#_Toc167383442)

[2.1. Конструкторские работы 8](#_Toc167383443)

[2.1.1. Диаграмма вариантов использования 8](#_Toc167383444)

[2.1.2. Диаграмма классов 11](#_Toc167383445)

[2.1.3. Диаграмма базы данных 12](#_Toc167383446)

[ГЛАВА 3. СТАДИЯ РЕАЛИЗАЦИИ 13](#_Toc167383447)

[3.1. Руководство программиста 13](#_Toc167383448)

[Шаг 5: Кодирование 15](#_Toc167383449)

[3.1. Описание классов и их назначения 15](#_Toc167383450)

[3.1.1. Класс DatabaseManager 15](#_Toc167383451)

[3.1.2. Класс User 17](#_Toc167383452)

[3.1.3. Класс MaterialManager 18](#_Toc167383453)

[}; 20](#_Toc167383454)

[3.1.4. Класс QuestionManager 20](#_Toc167383455)

[}; 23](#_Toc167383456)

[3.1.5. Класс TestResultManager 23](#_Toc167383457)

[3.1.6. Класс FeedbackManager 25](#_Toc167383458)

[3.1.7. Класс NotidicationManager 27](#_Toc167383459)

[3.1.8. Класс Admin 28](#_Toc167383460)

[3.1.9. Класс Employee 31](#_Toc167383461)

[3.1.10. Класс Logger 32](#_Toc167383462)

[3.1.11. Класс DatabaseException 33](#_Toc167383463)

[3.1.12. Абстрактный класс IManagable 34](#_Toc167383464)

[3.1.12. Тестирование 35](#_Toc167383465)

[3.1.13 Заключение 45](#_Toc167383466)

Введение

В современном информационном обществе компании связи играют важную роль в обеспечении эффективной коммуникации и передачи данных. Однако, с увеличением объема информации и быстроты развития технологий, непрерывное обновление знаний сотрудников становится необходимостью. Автоматизированные системы для проверки знаний сотрудников компаний связи являются эффективным и инновационным решением, позволяющим повысить профессиональную компетентность и эффективность работы персонала.

Традиционные методы оценки знаний и навыков сотрудников часто ограничены временем, ресурсами и неспособны обеспечить объективность результатов. Автоматизированные системы проверки знаний позволяют решить эти проблемы, предоставляя компаниям возможность проводить комплексные и структурированные тесты, которые могут охватывать широкий спектр тем и областей.

Такие системы обычно основаны на использовании современных технологий, таких как искусственный интеллект и машинное обучение. Они способны адаптироваться к индивидуальным потребностям компании и ее сотрудников, предоставляя гибкую и персонализированную оценку знаний. Кроме того, автоматизированные системы могут предложить дополнительные обучающие материалы и рекомендации для заполнения пробелов в знаниях.

Применение автоматизированных систем для проверки знаний сотрудников компаний связи имеет множество преимуществ. Оно позволяет повысить эффективность обучения и развития персонала, а также обеспечить единые стандарты знаний внутри компании. Кроме того, такие системы способствуют улучшению качества обслуживания клиентов и снижению ошибок, что является особенно важным в сфере связи.

ГЛАВА 1. АНАЛИТИЧЕСКАЯ ЧАСТЬ

1.1. Актуальность проблемы

Создание приложения для проверки знаний сотрудников компаний связи является актуальной проблемой в настоящее время. Компании связи работают в сфере быстро развивающихся технологий, и для поддержания конкурентоспособности необходимо, чтобы их сотрудники были в курсе последних изменений и обладали высоким уровнем знаний.

Одним из ключевых ресурсов, которыми располагают компании связи, являются их сотрудники. Они выполняют широкий спектр задач, включая техническую поддержку, установку и обслуживание оборудования, работу в центрах обслуживания клиентов и т. д. Правильное и эффективное выполнение этих задач требует хорошего знания и понимания технологий, услуг и процессов, связанных с деятельностью компании связи.

Однако, проверка знаний может быть сложной и трудоемкой задачей для компаний, особенно если у них большое количество сотрудников. В этом случае приложение для проверки знаний может значительно упростить и автоматизировать процесс. Кроме того, такая система позволяет сотрудникам компаний связи легко и удобно получать необходимые знания и обучаться новым навыкам. Такой подход помогает компаниям быстро адаптироваться к изменениям в отрасли и поддерживать высокий уровень качества обслуживания клиентов. В связи с этим, создание системы для проверки знаний сотрудников компаний связи является важным шагом в повышении квалификации и эффективности работы сотрудников в данной сфере.

В данном контексте создание приложения для проверки знаний сотрудников компаний связи имеет ряд значимых преимуществ. Вот некоторые из них:

1. Оценка и повышение квалификации сотрудников
2. Стандартизация знаний
3. Мониторинг и анализ
4. Гибкость и доступность
5. Экономия времени и ресурсов
6. Адаптивное обучение
7. Мотивация и вовлеченность

В целом, приложение для проверки знаний сотрудников компаний связи является неотъемлемым инструментом для повышения квалификации, стандартизации знаний и обеспечения высокого качества обслуживания клиентов. Оно обладает гибкостью, доступностью и способствует экономии времени и ресурсов. Помимо этого, такое приложение способствует мотивации сотрудников, обновлению и адаптации содержания и является эффективным инструментом для мониторинга и анализа производительности.

В современном быстро меняющемся мире технологий и связи, где конкуренция высока, создание веб-приложения для проверки знаний становится необходимостью для компаний связи, стремящихся оставаться на переднем крае индустрии и обеспечить качественное обслуживание своим клиентам.

1.2. Спецификация проблемы.

Проблемы проверки знаний сотрудников компаний связи могут включать следующие аспекты:

1. Большие затраты ресурсов и времени на обучение и проверку знаний сотрудников (теста, отчета, анализа);
2. Несоответствующая должность сотрудника по качеству знания;
3. Возникновение барьеров в развитии компании (недопонимание в совместной работе сотрудников, низкий показатель активности работников);
4. Потеря прибыли, нанесение ущерба компании
5. Человеческий фактор (ошибки при проверке теста, утеря тестового бланка)

Все эти проблемы могут быть решены с помощью автоматизированной системы для проверки знаний сотрудников компаний связи. Такая система может обеспечить объективность оценки, гибкость и персонализацию процесса проверки, возможность обновления контента и непрерывного обучения, а также предоставление подробной обратной связи и отчетности. Автоматизированная система позволит компаниям связи эффективно управлять знаниями своего персонала, обеспечивая их постоянное развитие и соответствие современным требованиям отрасли.

1.3. Описание проекта

Полное название системы: Программная система для проверки знаний сотрудников компаний связи.

1.4. Назначение и цели создания системы

1.4.1. Назначение

Система предназначена для проверки знаний сотрудников компаний связи.

1.4.2. Цели

Основной целью для создания автоматизированной системы является повышение эффективности, профессиональности работы сотрудников, результат которого приведет к увеличению прибыли компаний связи. Автоматизированная система предназначена для того, чтобы помочь сотрудникам понять, какую ключевую роль они играют в деятельности компании.

1.4.3. Требования к функциям:

[1] Пользователями системы являются администратор и сотрудники компаний связи. Программа должна обеспечивать возможность выполнения следующих пользовательских функций:

**Администратор:**

1. Регистрация;
2. Авторизация;
3. Добавлять, редактировать, удалять материалы для обучения;
4. Добавлять, редактировать, удалять тесты;
5. Просмотреть результат тестирования;
6. Отправлять уведомления сотруднику;
7. Просмотреть обратную связь от сотрудников;
8. Распечатать результаты тестов;

**Сотрудник:**

1. Регистрация;
2. Авторизация;
3. Ознакомиться с материалами для обучения;
4. Сдать тест;
5. Просмотреть результат тестирования;
6. Оставить обратную связь;
7. Просмотреть полученные уведомления;

**Система:**

1. Идентификация / аутентификация;
2. Сохранение данных;
3. Подсчет баллов;
4. Формирование отчета;

ГЛАВА 2. ПРОЕКТНАЯ ЧАСТЬ

2.1. Конструкторские работы

2.1.1. Диаграмма вариантов использования

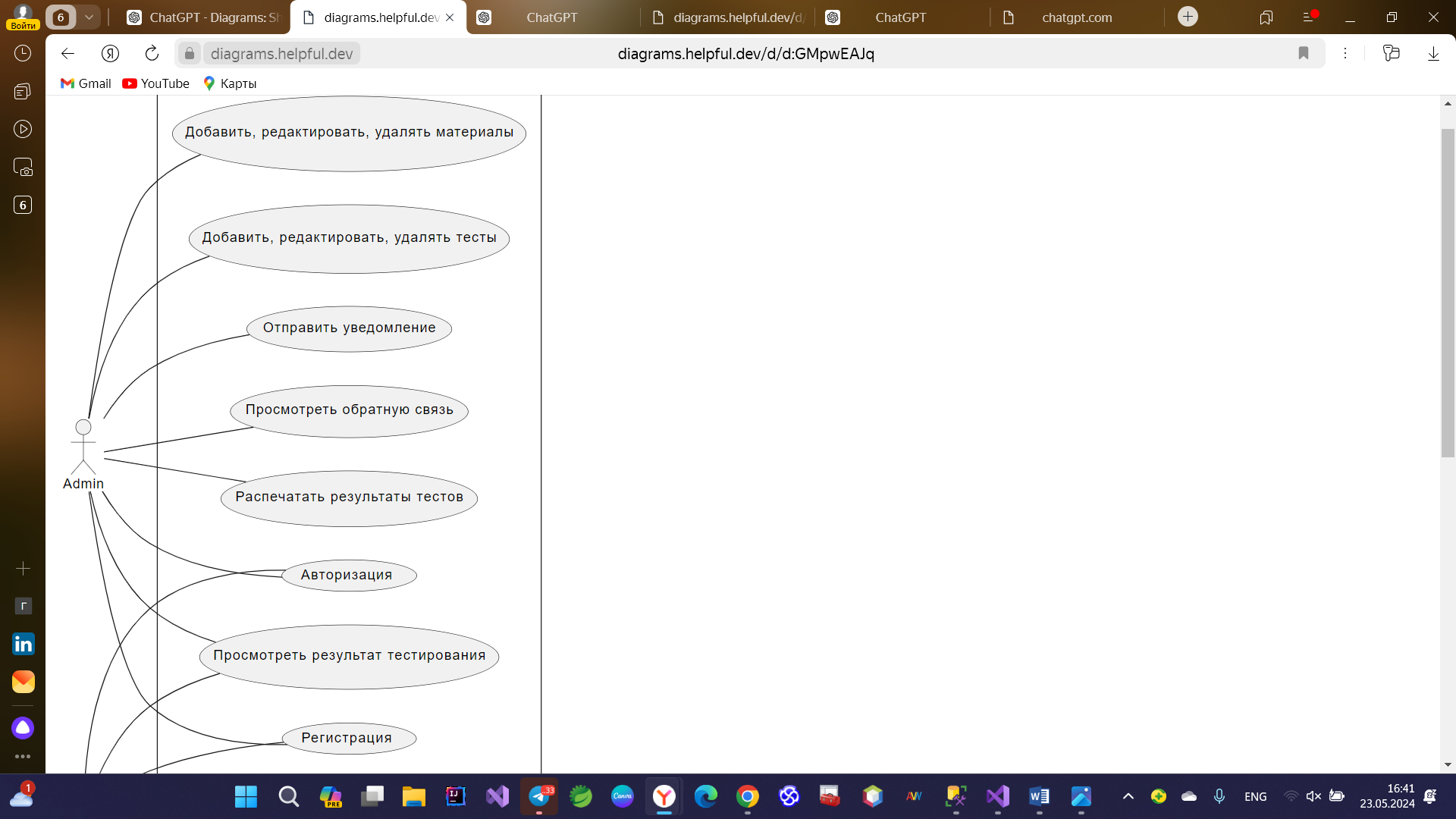
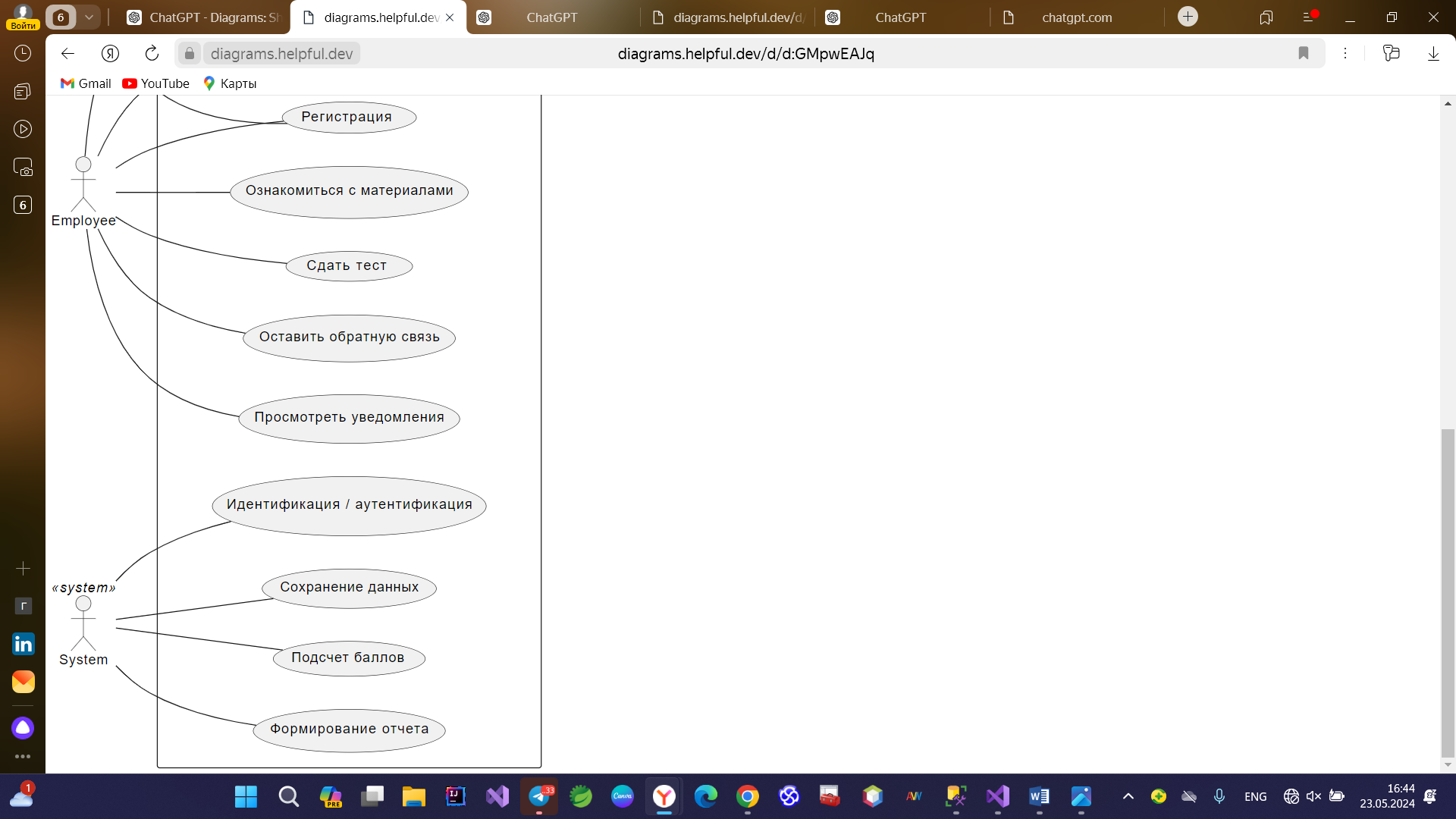
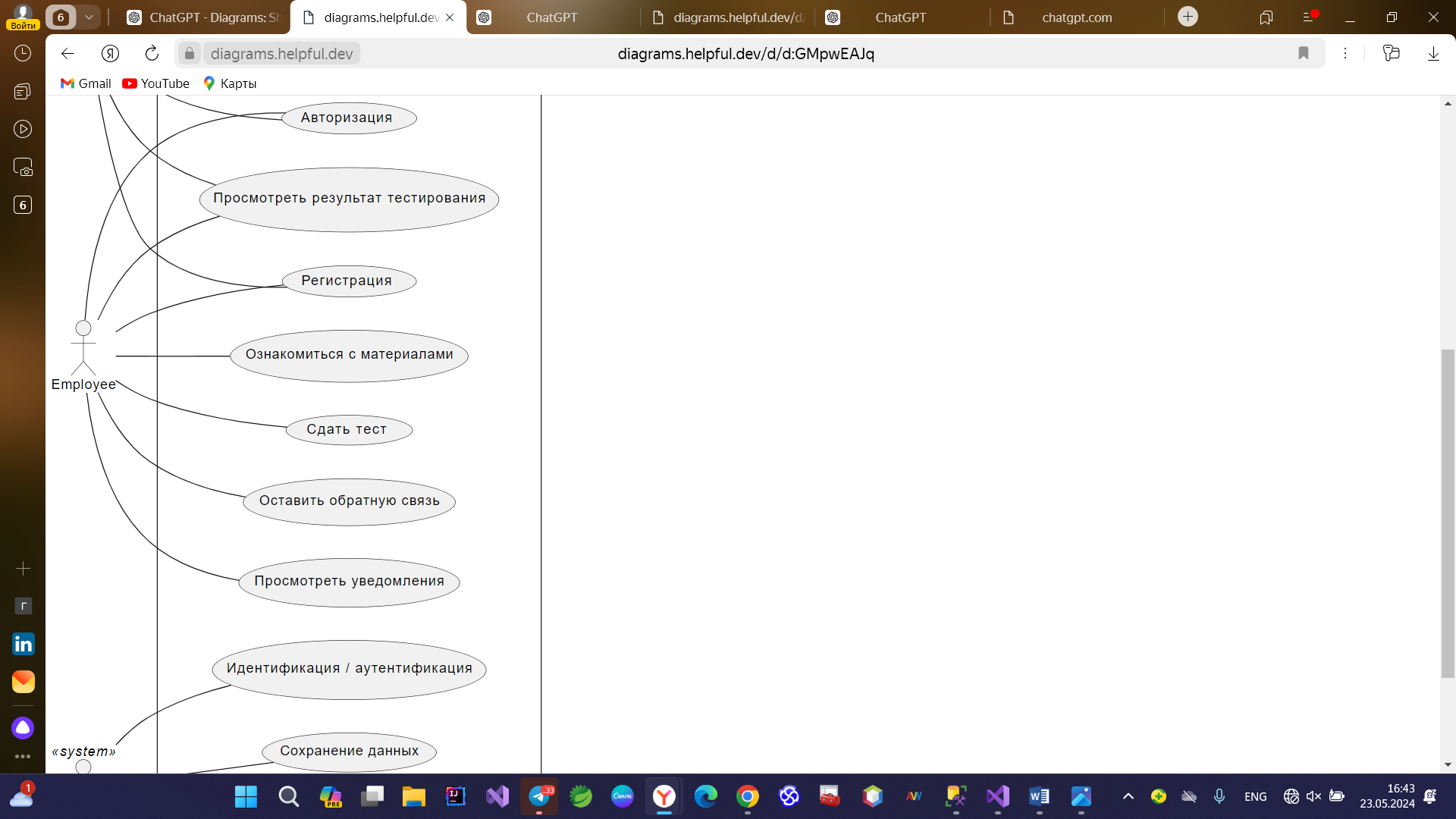
**Диаграмма вариантов использования** (Use Case Diagram) - (диаграмма вариантов использования, ВИ, Прецедент, юскейс) — является графическим средством моделирования, используемым в области разработки программного обеспечения и системного анализа. Она представляет собой способ визуализации функциональных требований системы из перспективы пользователей.

Диаграмма вариантов использования отображает различные сценарии использования системы, которые называются вариантами использования или прецедентами использования. Вариант использования представляет собой функциональное требование, которое описывает взаимодействие между актерами (пользователями или другими системами) и системой. Он описывает, как система будет использоваться для достижения определенных целей.

На диаграмме вариантов использования прецеденты использования представлены в виде овалов или эллипсов, а актеры обозначаются в виде человеческих фигур или прямоугольников. Связи между актерами и вариантами использования показывают взаимодействие между ними.

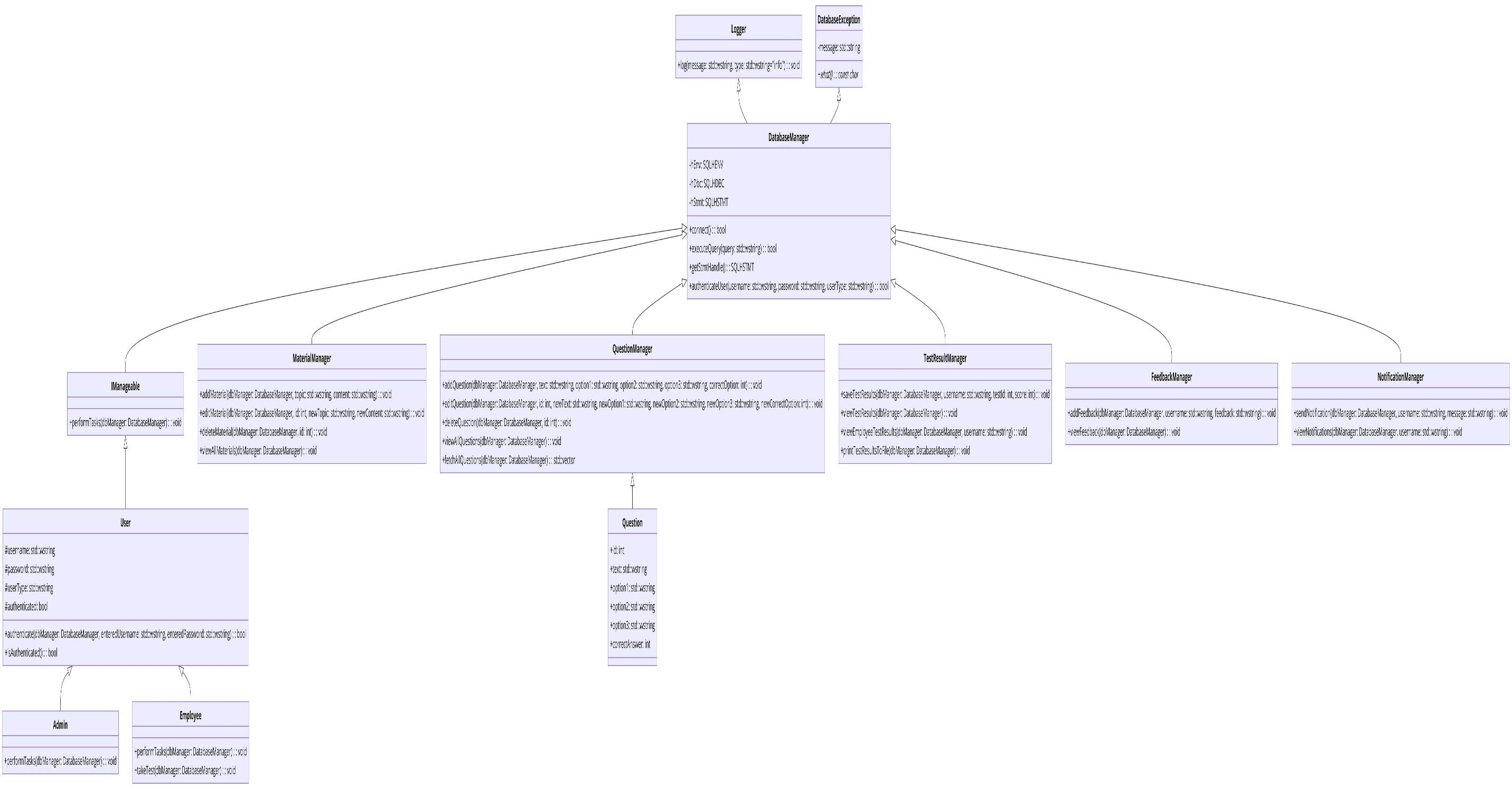
Диаграмма вариантов использования помогает командам разработчиков и аналитикам системы понять и представить функциональные требования системы, идентифицировать актеров, определить сценарии использования и связи между ними. Она также может быть использована для общения с заинтересованными сторонами и улучшения понимания требований к системе.

Ниже на рис 2.1. представлена диаграмма вариантов использование для программной системы для проверки знаний компании сотрудников связи.

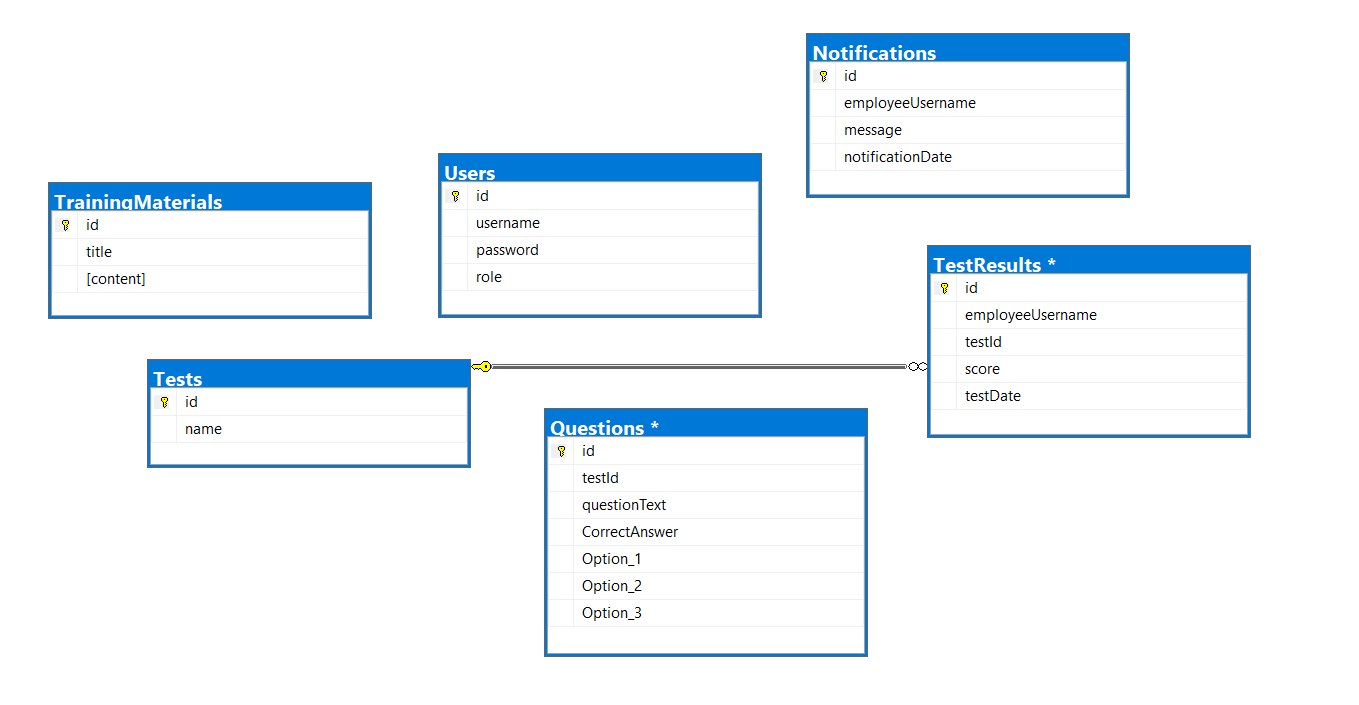
 

*Рис.2.1. Диаграмма вариантов использования*

2.1.2. Диаграмма классов



*Рис.2.2. Диаграмма классов*

2.1.3. Диаграмма базы данных

*Рис.2.3. Диаграмма базы данных*

ГЛАВА 3. СТАДИЯ РЕАЛИЗАЦИИ

3.1. Руководство программиста

Назначение и условие, необходимые для выполнения программы.

Программа предназначена для автоматизации процесса тестирования сотрудников компаний связи. Программа позволяет заносить информацию о проходивших тестирование в базу данных, получить рейтинг, сразу после тестирования выдать оценку знаний.

Для функционирования программно-аппаратного комплекса к вычислительным средствам (системе) предъявляются следующие минимальные требования:

* Процессор Pentium или AMD с тактовой частотой не менее 120 Mhz;
* Оперативную память объемом, 128 Мб, не менее;
* Операционную систему Windows XP/Vista/7;
* Свободное место на жестком диске: 200 Мб.
* Требования к составу и параметрам периферийных устройств:
* Стандартная клавиатура PS/2 или usb;
* Мышь.

Основные выполняемые функции: сдача теста, выдача результатов.

Руководство программиста для веб-приложения “Проверка знаний сотрудников компаний связи”.

**Шаг 1: Установка необходимого программного обеспечения и среды разработки**

1. Установите подходящую вам по выбору среду разработки ПО.
2. Установить С++
3. Установите Visual Studio 2019
4. Скачайте MS SQL SERVER 2020

**Шаг 2: Создание Django-проекта**

**Шаг 3: Подключение к базе данных**

bool connect() {

SQLRETURN retcode = SQLAllocHandle(SQL\_HANDLE\_ENV, SQL\_NULL\_HANDLE, &hEnv);

if (retcode != SQL\_SUCCESS && retcode != SQL\_SUCCESS\_WITH\_INFO) return false;

SQLSetEnvAttr(hEnv, SQL\_ATTR\_ODBC\_VERSION, (void\*)SQL\_OV\_ODBC3, 0);

retcode = SQLAllocHandle(SQL\_HANDLE\_DBC, hEnv, &hDbc);

if (retcode != SQL\_SUCCESS && retcode != SQL\_SUCCESS\_WITH\_INFO) return false;

SQLWCHAR connectionString[] = L"DRIVER={ODBC Driver 17 for SQL Server};SERVER=ARZYSHKA;DATABASE=EmployeeTraining;UID=sa;PWD=12345678910";

retcode = SQLDriverConnectW(hDbc, NULL, connectionString, SQL\_NTS, NULL, 0, NULL, SQL\_DRIVER\_NOPROMPT);

if (retcode != SQL\_SUCCESS && retcode != SQL\_SUCCESS\_WITH\_INFO) return false;

retcode = SQLAllocHandle(SQL\_HANDLE\_STMT, hDbc, &hStmt);

return retcode == SQL\_SUCCESS || retcode == SQL\_SUCCESS\_WITH\_INFO;

}

**Шаг 4: Настройка объектов БД. Создание таблиц.**

1. Таблица Employees :

SELECT TOP (1000) [id]

,[username]

,[password]

FROM [EmployeeTraining].[dbo].[Employees]

2)Таблица TrainingMaterials :

SELECT TOP (1000) [id]

,[title]

,[content]

FROM [EmployeeTraining].[dbo].[TrainingMaterials]

3)Таблица Tests :

SELECT TOP (1000) [id]

,[name]

FROM [EmployeeTraining].[dbo].[Tests]

4)Таблица TestResults :

SELECT TOP (1000) [id]

,[employeeUsername]

,[testId]

,[score]

,[testDate]

FROM [EmployeeTraining].[dbo].[TestResults]

5)Таблица Questions :

SELECT TOP (1000) [id]

,[questionText]

,[Option1]

,[Option2]

,[Option3]

,[CorrectAnswer]

FROM [EmployeeTraining].[dbo].[Questions]

6)Таблица Notifications:

SELECT TOP (1000) [id]

,[employeeUsername]

,[notificationMessage]

FROM [EmployeeTraining].[dbo].[Notifications]

7)Таблица Feedback:

SELECT TOP (1000) [id]

,[employeeUsername]

,[feedback]

,[feedbackDate]

FROM [EmployeeTraining].[dbo].[Feedback]

Шаг 5: Кодирование

Классы: Logger, DatabaseException, DatabaseManager, IManageable, class User : public IManageable, class MaterialManager, class QuestionManager, class TestResultManager, class FeedbackManager, class NotificationManager, class Admin : public User, class Employee : public User.

## 3.1. Описание классов и их назначения

### 3.1.1. Класс DatabaseManager

Назначение класса: Класс для управления подключением к базе данных и выполнения запросов. Реализует методы подключения, выполнения запросов и аутентификации пользователей.

Описание методов класса:

|  |  |  |
| --- | --- | --- |
| № | Сигнатура | Назначение |
| 1 | bool connect() | Устанавливает соединение с базой данных. Возвращает true, если соединение успешно установлено. |
| 2 | **bool executeQuery(const wstring& query)** | Выполняет SQL-запрос и обрабатывает исключения при его выполнении. Возвращает **true**, если запрос успешно выполнен. |
| 3 | **SQLHSTMT getStmtHandle() const** | Возвращает дескриптор оператора SQL для выполнения дальнейших операций. |
| 4 | **bool authenticateUser(const wstring& username, const wstring& password, wstring& userType)** | Аутентификация пользователя по имени и паролю. Возвращает **true**, если пользователь успешно аутентифицирован. |

В данном классе реализована перегрузка методов для метода executeQuery()

В классе реализованы переопределение операций для операций подключения и выполнения запросов.

class DatabaseManager {

private:

SQLHENV hEnv;

SQLHDBC hDbc;

SQLHSTMT hStmt;

public:

DatabaseManager() : hEnv(nullptr), hDbc(nullptr), hStmt(nullptr) {}

~DatabaseManager() {

if (hStmt) SQLFreeHandle(SQL\_HANDLE\_STMT, hStmt);

if (hDbc) {

SQLDisconnect(hDbc);

SQLFreeHandle(SQL\_HANDLE\_DBC, hDbc);

}

if (hEnv) SQLFreeHandle(SQL\_HANDLE\_ENV, hEnv);

}

bool connect() {

SQLRETURN retcode = SQLAllocHandle(SQL\_HANDLE\_ENV, SQL\_NULL\_HANDLE, &hEnv);

if (retcode != SQL\_SUCCESS && retcode != SQL\_SUCCESS\_WITH\_INFO) return false;

SQLSetEnvAttr(hEnv, SQL\_ATTR\_ODBC\_VERSION, (void\*)SQL\_OV\_ODBC3, 0);

retcode = SQLAllocHandle(SQL\_HANDLE\_DBC, hEnv, &hDbc);

if (retcode != SQL\_SUCCESS && retcode != SQL\_SUCCESS\_WITH\_INFO) return false;

SQLWCHAR connectionString[] = L"DRIVER={ODBC Driver 17 for SQL Server};SERVER=ARZYSHKA;DATABASE=EmployeeTraining;UID=sa;PWD=12345678910";

retcode = SQLDriverConnectW(hDbc, NULL, connectionString, SQL\_NTS, NULL, 0, NULL, SQL\_DRIVER\_NOPROMPT);

if (retcode != SQL\_SUCCESS && retcode != SQL\_SUCCESS\_WITH\_INFO) return false;

retcode = SQLAllocHandle(SQL\_HANDLE\_STMT, hDbc, &hStmt);

return retcode == SQL\_SUCCESS || retcode == SQL\_SUCCESS\_WITH\_INFO;

}

bool executeQuery(const wstring& query) {

if (!connect()) {

Logger<string>::log("Failed to connect to database.", L"error");

return false;

}

SQLRETURN retcode = SQLExecDirect(hStmt, (SQLWCHAR\*)query.c\_str(), SQL\_NTS);

if (retcode != SQL\_SUCCESS && retcode != SQL\_SUCCESS\_WITH\_INFO) {

SQLWCHAR sqlState[1024], message[1024];

SQLINTEGER nativeError;

SQLSMALLINT msgLength;

SQLGetDiagRec(SQL\_HANDLE\_STMT, hStmt, 1, sqlState, &nativeError, message, 1024, &msgLength);

wcerr << L"SQL Error: " << sqlState << L" - " << message << endl;

throw DatabaseException("Failed to execute query");

}

return true;

}

SQLHSTMT getStmtHandle() const { return hStmt; }

bool authenticateUser(const wstring& username, const wstring& password, wstring& userType) {

wstring query = L"SELECT userType FROM Users WHERE username = '" + username + L"' AND password = '" + password + L"'";

if (!executeQuery(query)) {

return false;

}

SQLHSTMT stmt = getStmtHandle();

SQLWCHAR dbUserType[50];

if (SQLFetch(stmt) == SQL\_SUCCESS) {

SQLGetData(stmt, 1, SQL\_C\_WCHAR, dbUserType, sizeof(dbUserType), nullptr);

userType = dbUserType;

return true;

}

return false;

}

};

### 3.1.2. Класс User

Назначение класса: Базовый класс для пользователей системы, реализует интерфейс IManageable. Содержит методы для аутентификации и проверки аутентификации.

Описание методов класса:

|  |  |  |
| --- | --- | --- |
| № | Сигнатура | Назначение |
| 1 | User(const std::wstring& username, const std::wstring& password, const std::wstring& userType = L"") | Конструктор, инициализирующий пользователя. |
| 2 | bool authenticate(DatabaseManager& dbManager, const std::wstring& enteredUsername, const std::wstring& enteredPassword) | Метод для аутентификации пользователя. |
| 3 | bool isAuthenticated() const | Метод для проверки, аутентифицирован ли пользователь. |

Данный класс наследуемый класс от IManageable

class User : public IManageable {

protected:

wstring username;

wstring password;

wstring userType;

bool authenticated;

public:

User(const wstring& username, const wstring& password, const wstring& userType = L"") : username(username), password(password), userType(userType), authenticated(false) {}

bool authenticate(DatabaseManager& dbManager, const wstring& enteredUsername, const wstring& enteredPassword) {

wstring dbUserType;

if (dbManager.authenticateUser(enteredUsername, enteredPassword, dbUserType)) {

authenticated = true;

userType = dbUserType;

return true;

}

authenticated = false;

return false;

}

bool isAuthenticated() const { return authenticated; }

virtual ~User() {}

};

### 3.1.3. Класс MaterialManager

Назначение класса: Класс для управления обучающими материалами. Реализует методы для добавления, редактирования, удаления и просмотра всех материалов.

Описание методов класса:

|  |  |  |
| --- | --- | --- |
| № | Сигнатура | Назначение |
| 1 | void addMaterial(DatabaseManager& dbManager, const std::wstring& topic, const std::wstring& content) | Метод для добавления нового материала. |
| 2 | void editMaterial(DatabaseManager& dbManager, int id, const std::wstring& newTopic, const std::wstring& newContent) | Метод для редактирования материала. |
| 3 | void deleteMaterial(DatabaseManager& dbManager, int id) | Метод для удаления материала. |
| 4 | void viewAllMaterials(DatabaseManager& dbManager) | Метод для просмотра всех материалов. |

class MaterialManager {

public:

void addMaterial(DatabaseManager& dbManager, const wstring& topic, const wstring& content) {

wstring query = L"INSERT INTO TrainingMaterials (title, content) VALUES ('" + topic + L"', '" + content + L"')";

if (!dbManager.executeQuery(query)) {

wcout << L"Failed to add material." << endl;

}

else {

wcout << L"Material added successfully." << endl;

}

}

void editMaterial(DatabaseManager& dbManager, int id, const wstring& newTopic, const wstring& newContent) {

wstring query = L"UPDATE TrainingMaterials SET title = '" + newTopic + L"', content = '" + newContent + L"' WHERE id = " + to\_wstring(id);

if (!dbManager.executeQuery(query)) {

wcout << L"Failed to update material." << endl;

}

else {

wcout << L"Material updated successfully." << endl;

}

}

void deleteMaterial(DatabaseManager& dbManager, int id) {

wstring query = L"DELETE FROM TrainingMaterials WHERE id = " + to\_wstring(id);

if (!dbManager.executeQuery(query)) {

wcout << L"Failed to delete material." << endl;

}

else {

wcout << L"Material deleted successfully." << endl;

}

}

void viewAllMaterials(DatabaseManager& dbManager) {

wstring query = L"SELECT \* FROM TrainingMaterials";

if (!dbManager.executeQuery(query)) {

wcerr << L"Failed to execute SQL query." << endl;

return;

}

SQLHSTMT stmt = dbManager.getStmtHandle();

SQLINTEGER id;

SQLWCHAR title[255], content[1024];

wcout << L"Material ID\tTitle\tContent" << endl;

while (SQLFetch(stmt) == SQL\_SUCCESS) {

SQLGetData(stmt, 1, SQL\_C\_SLONG, &id, 0, nullptr);

SQLGetData(stmt, 2, SQL\_C\_WCHAR, title, 255, nullptr);

SQLGetData(stmt, 3, SQL\_C\_WCHAR, content, 1024, nullptr);

wcout << id << L"\t" << title << L"\t" << content << endl;

}

}

### };

### 3.1.4. Класс QuestionManager

Назначение класса: Класс для управления вопросами тестов. Реализует методы для добавления, редактирования, удаления и просмотра всех вопросов, а также метод для получения всех вопросов.

Описание методов класса:

|  |  |  |
| --- | --- | --- |
| № | Сигнатура | Назначение |
| 1 | struct Question | Вложенная структура для представления вопроса. |
| 2 | std::vector<Question> fetchAllQuestions(DatabaseManager& dbManager) | Метод для получения всех вопросов из базы данных. |
| 3 | void addQuestion(DatabaseManager& dbManager, const std::wstring& text, const std::wstring& option1, const std::wstring& option2, const std::wstring& option3, int correctOption) | Метод для добавления нового вопроса. |
| 4 | void editQuestion(DatabaseManager& dbManager, int id, const std::wstring& newText, const std::wstring& newOption1, const std::wstring& newOption2, const std::wstring& newOption3, int newCorrectOption) | Метод для редактирования вопроса. |
| 5 | void deleteQuestion(DatabaseManager& dbManager, int id) | Метод для удаления вопроса. |
| 6 | void viewAllQuestions(DatabaseManager& dbManager) | Метод для просмотра всех вопросов. |

class QuestionManager {

public:

struct Question {

int id;

wstring text;

wstring option1;

wstring option2;

wstring option3;

int correctAnswer;

};

vector<Question> fetchAllQuestions(DatabaseManager& dbManager) {

wstring query = L"SELECT id, questionText, Option1, Option2, Option3, CorrectAnswer FROM Questions";

if (!dbManager.executeQuery(query)) {

wcerr << L"Failed to fetch questions." << endl;

return {};

}

SQLHSTMT stmt = dbManager.getStmtHandle();

SQLINTEGER id, correctAnswer;

SQLWCHAR text[1024], option1[255], option2[255], option3[255];

vector<Question> questions;

while (SQLFetch(stmt) == SQL\_SUCCESS) {

SQLGetData(stmt, 1, SQL\_C\_SLONG, &id, 0, nullptr);

SQLGetData(stmt, 2, SQL\_C\_WCHAR, text, 1024, nullptr);

SQLGetData(stmt, 3, SQL\_C\_WCHAR, option1, 255, nullptr);

SQLGetData(stmt, 4, SQL\_C\_WCHAR, option2, 255, nullptr);

SQLGetData(stmt, 5, SQL\_C\_WCHAR, option3, 255, nullptr);

SQLGetData(stmt, 6, SQL\_C\_SLONG, &correctAnswer, 0, nullptr);

questions.push\_back({ id, text, option1, option2, option3, correctAnswer });

}

return questions;

}

void addQuestion(DatabaseManager& dbManager, const wstring& text, const wstring& option1, const wstring& option2, const wstring& option3, int correctOption) {

wstring query = L"INSERT INTO Questions (questionText, Option1, Option2, Option3, CorrectAnswer) VALUES ('"

+ text + L"', '" + option1 + L"', '" + option2 + L"', '" + option3 + L"', " + to\_wstring(correctOption) + L")";

if (!dbManager.executeQuery(query)) {

wcout << L"Failed to add question." << endl;

}

else {

wcout << L"Question added successfully." << endl;

}

}

void editQuestion(DatabaseManager& dbManager, int id, const wstring& newText, const wstring& newOption1, const wstring& newOption2, const wstring& newOption3, int newCorrectOption) {

wstring query = L"UPDATE Questions SET questionText = '" + newText + L"', Option1 = '" + newOption1 + L"', Option2 = '"

+ newOption2 + L"', Option3 = '" + newOption3 + L"', CorrectAnswer = " + to\_wstring(newCorrectOption) + L" WHERE id = " + to\_wstring(id);

if (!dbManager.executeQuery(query)) {

wcout << L"Failed to update question." << endl;

}

else {

wcout << L"Question updated successfully." << endl;

}

}

void deleteQuestion(DatabaseManager& dbManager, int id) {

wstring query = L"DELETE FROM Questions WHERE id = " + to\_wstring(id);

if (!dbManager.executeQuery(query)) {

wcout << L"Failed to delete question." << endl;

}

else {

wcout << L"Question deleted successfully." << endl;

}

}

void viewAllQuestions(DatabaseManager& dbManager) {

wstring query = L"SELECT \* FROM Questions";

if (!dbManager.executeQuery(query)) {

wcerr << L"Failed to execute SQL query." << endl;

return;

}

SQLHSTMT stmt = dbManager.getStmtHandle();

SQLINTEGER id, correctAnswer;

SQLWCHAR text[1024], option1[255], option2[255], option3[255];

wcout << L"Question ID\tText\tOption 1\tOption 2\tOption 3\tCorrect Answer" << endl;

while (SQLFetch(stmt) == SQL\_SUCCESS) {

SQLGetData(stmt, 1, SQL\_C\_SLONG, &id, 0, nullptr);

SQLGetData(stmt, 2, SQL\_C\_WCHAR, text, 1024, nullptr);

SQLGetData(stmt, 3, SQL\_C\_WCHAR, option1, 255, nullptr);

SQLGetData(stmt, 4, SQL\_C\_WCHAR, option2, 255, nullptr);

SQLGetData(stmt, 5, SQL\_C\_WCHAR, option3, 255, nullptr);

SQLGetData(stmt, 6, SQL\_C\_SLONG, &correctAnswer, 0, nullptr);

wcout << id << L"\t" << text << L"\t" << option1 << L"\t" << option2 << L"\t" << option3 << L"\t" << correctAnswer << endl;

}

}

### };

### 3.1.5. Класс TestResultManager

Назначение класса: Класс для управления результатами тестов. Реализует методы для сохранения, просмотра и печати результатов тестов, а также для просмотра результатов тестов конкретного сотрудника.

Описание методов класса:

|  |  |  |
| --- | --- | --- |
| № | Сигнатура | Назначение |
| 1 | void saveTestResults(DatabaseManager& dbManager, const std::wstring& username, int testId, int score) | Метод для сохранения результатов теста. |
| 2 | void viewTestResults(DatabaseManager& dbManager) | Метод для просмотра всех результатов тестов. |
| 3 | void viewEmployeeTestResults(DatabaseManager& dbManager, const std::wstring& username) | Метод для просмотра результатов тестов конкретного сотрудника. |
| 4 | void printTestResultsToFile(DatabaseManager& dbManager) | Метод для печати результатов тестов в файл. |

class TestResultManager {

public:

void saveTestResults(DatabaseManager& dbManager, const wstring& username, int testId, int score) {

wstring query = L"INSERT INTO TestResults (employeeUsername, testId, score, testDate) VALUES ('" + username + L"', " + to\_wstring(testId) + L", " + to\_wstring(score) + L", GETDATE())";

if (!dbManager.executeQuery(query)) {

wcout << L"Failed to save test results." << endl;

}

else {

wcout << L"Test results saved successfully." << endl;

}

}

void viewTestResults(DatabaseManager& dbManager) {

wstring query = L"SELECT employeeUsername, testId, score, testDate FROM TestResults ORDER BY testDate DESC";

if (!dbManager.executeQuery(query)) {

wcerr << L"Failed to fetch test results." << endl;

return;

}

SQLHSTMT stmt = dbManager.getStmtHandle();

SQLWCHAR username[255];

SQLINTEGER testId, score;

SQL\_TIMESTAMP\_STRUCT testDate;

wcout << L"Username\tTest ID\tScore\tTest Date" << endl;

while (SQLFetch(stmt) == SQL\_SUCCESS) {

SQLGetData(stmt, 1, SQL\_C\_WCHAR, &username, sizeof(username), NULL);

SQLGetData(stmt, 2, SQL\_C\_SLONG, &testId, 0, NULL);

SQLGetData(stmt, 3, SQL\_C\_SLONG, &score, 0, NULL);

SQLGetData(stmt, 4, SQL\_C\_TYPE\_TIMESTAMP, &testDate, 0, NULL);

wchar\_t dateStr[100];

swprintf(dateStr, 100, L"%04d-%02d-%02d %02d:%02d:%02d",

testDate.year, testDate.month, testDate.day,

testDate.hour, testDate.minute, testDate.second);

wcout << username << L"\t" << testId << L"\t" << score << L"\t" << dateStr << endl;

}

}

void viewEmployeeTestResults(DatabaseManager& dbManager, const wstring& username) {

wstring query = L"SELECT testId, score, testDate FROM TestResults WHERE employeeUsername = '" + username + L"' ORDER BY testDate DESC";

if (!dbManager.executeQuery(query)) {

wcerr << L"Failed to fetch test results for employee." << endl;

return;

}

SQLHSTMT stmt = dbManager.getStmtHandle();

SQLINTEGER testId, score;

SQL\_TIMESTAMP\_STRUCT testDate;

wcout << L"Test ID\tScore\tTest Date" << endl;

while (SQLFetch(stmt) == SQL\_SUCCESS) {

SQLGetData(stmt, 1, SQL\_C\_SLONG, &testId, 0, NULL);

SQLGetData(stmt, 2, SQL\_C\_SLONG, &score, 0, NULL);

SQLGetData(stmt, 3, SQL\_C\_TYPE\_TIMESTAMP, &testDate, 0, NULL);

wchar\_t dateStr[100];

swprintf(dateStr, 100, L"%04d-%02d-%02d %02d:%02d-%02d",

testDate.year, testDate.month, testDate.day,

testDate.hour, testDate.minute, testDate.second);

wcout << testId << L"\t" << score << L"\t" << dateStr << endl;

}

}

void printTestResultsToFile(DatabaseManager& dbManager) {

wstring query = L"SELECT employeeUsername, testId, score, testDate FROM TestResults ORDER BY testDate DESC";

if (!dbManager.executeQuery(query)) {

wcerr << L"Failed to fetch test results." << endl;

return;

}

wofstream outfile("test\_results.txt");

outfile.imbue(locale(outfile.getloc(), new codecvt\_utf8<wchar\_t>)); // Set UTF-8 encoding for output file

outfile << L"Username\tTest ID\tScore\tTest Date\n";

SQLHSTMT stmt = dbManager.getStmtHandle();

SQLWCHAR username[255];

SQLINTEGER testId, score;

SQL\_TIMESTAMP\_STRUCT testDate;

while (SQLFetch(stmt) == SQL\_SUCCESS) {

SQLGetData(stmt, 1, SQL\_C\_WCHAR, &username, sizeof(username), NULL);

SQLGetData(stmt, 2, SQL\_C\_SLONG, &testId, 0, NULL);

SQLGetData(stmt, 3, SQL\_C\_SLONG, &score, 0, NULL);

SQLGetData(stmt, 4, SQL\_C\_TYPE\_TIMESTAMP, &testDate, 0, NULL);

wchar\_t dateStr[100];

swprintf(dateStr, 100, L"%04d-%02d-%02d %02d:%02d:%02d",

testDate.year, testDate.month, testDate.day,

testDate.hour, testDate.minute, testDate.second);

outfile << username << L"\t" << testId << L"\t" << score << L"\t" << dateStr << L"\n";

}

outfile.close();

wcout << L"Results printed to 'test\_results.txt'" << endl;

}

};

### 3.1.6. Класс FeedbackManager

Назначение класса: Класс для управления отзывами сотрудников. Реализует методы для добавления и просмотра отзывов.

Описание методов класса:

|  |  |  |
| --- | --- | --- |
| № | Сигнатура | Назначение |
| 1 | void addFeedback(DatabaseManager& dbManager, const std::wstring& username, const std::wstring& feedback) | Метод для добавления отзыва. |
| 2 | void viewFeedback(DatabaseManager& dbManager) | Метод для просмотра всех отзывов. |

class FeedbackManager {

public:

void addFeedback(DatabaseManager& dbManager, const wstring& username, const wstring& feedback) {

wstring query = L"INSERT INTO Feedback (employeeUsername, feedback, feedbackDate) VALUES ('" + username + L"', '" + feedback + L"', GETDATE())";

if (!dbManager.executeQuery(query)) {

wcout << L"Failed to add feedback." << endl;

}

else {

wcout << L"Feedback added successfully." << endl;

}

}

void viewFeedback(DatabaseManager& dbManager) {

wstring query = L"SELECT employeeUsername, feedback, feedbackDate FROM Feedback ORDER BY feedbackDate DESC";

if (!dbManager.executeQuery(query)) {

wcerr << L"Failed to fetch feedback." << endl;

return;

}

SQLHSTMT stmt = dbManager.getStmtHandle();

SQLWCHAR username[255], feedback[1024];

SQL\_TIMESTAMP\_STRUCT feedbackDate;

wcout << L"Username\tFeedback\tFeedback Date" << endl;

while (SQLFetch(stmt) == SQL\_SUCCESS) {

SQLGetData(stmt, 1, SQL\_C\_WCHAR, &username, sizeof(username), NULL);

SQLGetData(stmt, 2, SQL\_C\_WCHAR, &feedback, sizeof(feedback), NULL);

SQLGetData(stmt, 3, SQL\_C\_TYPE\_TIMESTAMP, &feedbackDate, 0, NULL);

wchar\_t dateStr[100];

swprintf(dateStr, 100, L"%04d-%02d-%02d %02d:%02d:%02d",

feedbackDate.year, feedbackDate.month, feedbackDate.day,

feedbackDate.hour, feedbackDate.minute, feedbackDate.second);

wcout << username << L"\t" << feedback << L"\t" << dateStr << endl;

}

}

};

### 3.1.7. Класс NotidicationManager

Назначение класса: Класс для управления уведомлениями. Реализует методы для отправки уведомлений и просмотра уведомлений конкретного сотрудника.

Описание методов класса:

|  |  |  |
| --- | --- | --- |
| № | Сигнатура | Назначение |
| 1 | void sendNotification(DatabaseManager& dbManager, const std::wstring& username, const std::wstring& message) | Метод для отправки уведомления. |
| 2 | void viewNotifications(DatabaseManager& dbManager, const std::wstring& username) | Метод для просмотра уведомлений конкретного сотрудника. |

class NotificationManager {

public:

void sendNotification(DatabaseManager& dbManager, const wstring& username, const wstring& message) {

wstring query = L"INSERT INTO Notifications (employeeUsername, notificationMessage) VALUES ('" + username + L"', '" + message + L"')";

if (!dbManager.executeQuery(query)) {

wcout << L"Failed to send notification." << endl;

}

else {

wcout << L"Notification sent successfully." << endl;

}

}

void viewNotifications(DatabaseManager& dbManager, const wstring& username) {

wstring query = L"SELECT notificationMessage FROM Notifications WHERE employeeUsername = '" + username + L"' ORDER BY id DESC";

if (!dbManager.executeQuery(query)) {

wcerr << L"Failed to fetch notifications." << endl;

return;

}

SQLHSTMT stmt = dbManager.getStmtHandle();

SQLWCHAR notificationMessage[1024];

wcout << L"Notifications for " << username << L":" << endl;

while (SQLFetch(stmt) == SQL\_SUCCESS) {

SQLGetData(stmt, 1, SQL\_C\_WCHAR, &notificationMessage, sizeof(notificationMessage), NULL);

wcout << L"- " << notificationMessage << endl;

}

}

};

### 3.1.8. Класс Admin

Назначение класса: Класс наследуется от User и реализует задачи администратора, такие как добавление, редактирование, удаление и просмотр материалов, вопросов, результатов тестов, отзывов и отправка уведомлений.

Описание методов класса:

|  |  |  |
| --- | --- | --- |
| № | Сигнатура | Назначение |
| 1 | Admin(const std::wstring& username, const std::wstring& password, DatabaseManager& dbMgr) | Конструктор, инициализирующий администратора. |
| 2 | void performTasks(DatabaseManager& dbManager) override | Метод для выполнения задач администратора. |

class Admin : public User {

private:

DatabaseManager& dbManager;

MaterialManager materialManager;

QuestionManager questionManager;

TestResultManager testResultManager;

FeedbackManager feedbackManager;

NotificationManager notificationManager;

public:

Admin(const wstring& username, const wstring& password, DatabaseManager& dbMgr) : User(username, password, L"admin"), dbManager(dbMgr) {}

void performTasks(DatabaseManager& dbManager) override {

int choice = 0;

do {

wcout << L"1. Add Material\n2. Add Question\n3. Edit Material\n4. Edit Question\n5. Delete Material\n6. Delete Question\n7. View All Materials\n8. View All Questions\n9. View Test Results\n10. Print Test Results\n11. View Feedback\n12. Send Notification\n13. Exit\nEnter your choice: ";

wcin >> choice;

wcin.ignore();

switch (choice) {

case 1:

{

wstring topic, content;

wcout << L"Enter material topic: ";

getline(wcin, topic);

wcout << L"Enter material content: ";

getline(wcin, content);

materialManager.addMaterial(dbManager, topic, content);

}

break;

case 2:

{

wstring text, option1, option2, option3;

int correctOption;

wcout << L"Enter question text: ";

getline(wcin, text);

wcout << L"Enter option 1: ";

getline(wcin, option1);

wcout << L"Enter option 2: ";

getline(wcin, option2);

wcout << L"Enter option 3: ";

getline(wcin, option3);

wcout << L"Enter the correct option number (1-3): ";

wcin >> correctOption;

questionManager.addQuestion(dbManager, text, option1, option2, option3, correctOption);

}

break;

case 3:

{

int id;

wstring newTopic, newContent;

wcout << L"Enter the ID of the material to edit: ";

wcin >> id;

wcin.ignore();

wcout << L"Enter new material topic: ";

getline(wcin, newTopic);

wcout << L"Enter new material content: ";

getline(wcin, newContent);

materialManager.editMaterial(dbManager, id, newTopic, newContent);

}

break;

case 4:

{

int id, newCorrectOption;

wstring newText, newOption1, newOption2, newOption3;

wcout << L"Enter the ID of the question to edit: ";

wcin >> id;

wcin.ignore();

wcout << L"Enter new question text: ";

getline(wcin, newText);

wcout << L"Enter new option 1: ";

getline(wcin, newOption1);

wcout << L"Enter new option 2: ";

getline(wcin, newOption2);

wcout << L"Enter new option 3: ";

getline(wcin, newOption3);

wcout << L"Enter the new correct option number (1-3): ";

wcin >> newCorrectOption;

questionManager.editQuestion(dbManager, id, newText, newOption1, newOption2, newOption3, newCorrectOption);

}

break;

case 5:

{

int id;

wcout << L"Enter the ID of the material to delete: ";

wcin >> id;

materialManager.deleteMaterial(dbManager, id);

}

break;

case 6:

{

int id;

wcout << L"Enter the ID of the question to delete: ";

wcin >> id;

questionManager.deleteQuestion(dbManager, id);

}

break;

case 7:

materialManager.viewAllMaterials(dbManager);

break;

case 8:

questionManager.viewAllQuestions(dbManager);

break;

case 9:

testResultManager.viewTestResults(dbManager);

break;

case 10:

testResultManager.printTestResultsToFile(dbManager);

break;

case 11:

feedbackManager.viewFeedback(dbManager);

break;

case 12:

{

wstring employeeUsername, notificationMessage;

wcout << L"Enter employee username: ";

getline(wcin, employeeUsername);

wcout << L"Enter notification message: ";

getline(wcin, notificationMessage);

notificationManager.sendNotification(dbManager, employeeUsername, notificationMessage);

}

break;

case 13:

wcout << L"Exiting..." << endl;

return;

default:

wcout << L"Invalid choice. Please try again." << endl;

break;

}

} while (choice != 13);

}

};

### 3.1.9. Класс Employee

Назначение класса: Класс наследуется от User и реализует задачи сотрудника, такие как просмотр материалов, прохождение тестов, просмотр результатов тестов, просмотр уведомлений и отправка отзывов.

Описание методов класса:

|  |  |  |
| --- | --- | --- |
| № | Сигнатура | Назначение |
| 1 | Employee(const std::wstring& username, const std::wstring& password, DatabaseManager& dbMgr) | Конструктор, инициализирующий сотрудника. |
| 2 | void performTasks(DatabaseManager& dbManager) override | Метод для выполнения задач сотрудника. |

class Employee : public User {

private:

DatabaseManager& dbManager;

MaterialManager materialManager;

TestResultManager testResultManager;

QuestionManager questionManager;

FeedbackManager feedbackManager;

NotificationManager notificationManager;

public:

Employee(const wstring& username, const wstring& password, DatabaseManager& dbMgr) : User(username, password, L"employee"), dbManager(dbMgr) {}

void performTasks(DatabaseManager& dbManager) override {

int choice;

do {

wcout << L"1. View Materials\n2. Take Test\n3. View My Test Results\n4. View Notifications\n5. Submit Feedback\n6. Exit\nEnter your choice: ";

wcin >> choice;

wcin.ignore();

switch (choice) {

case 1:

materialManager.viewAllMaterials(dbManager);

break;

case 2:

takeTest(dbManager);

break;

case 3:

testResultManager.viewEmployeeTestResults(dbManager, username);

break;

case 4:

notificationManager.viewNotifications(dbManager, username);

break;

case 5:

{

wstring feedback;

wcout << L"Enter your feedback: ";

getline(wcin, feedback);

feedbackManager.addFeedback(dbManager, username, feedback);

}

break;

case 6:

wcout << L"Exiting..." << endl;

return;

default:

wcout << L"Invalid choice. Please try again." << endl;

break;

}

} while (choice != 6);

}

void takeTest(DatabaseManager& dbManager) {

vector<QuestionManager::Question> questions = questionManager.fetchAllQuestions(dbManager);

int score = 0;

for (const auto& question : questions) {

wcout << L"Question: " << question.text << endl;

wcout << L"1. " << question.option1 << endl;

wcout << L"2. " << question.option2 << endl;

wcout << L"3. " << question.option3 << endl;

int answer;

wcout << L"Enter your answer (1-3): ";

wcin >> answer;

if (answer == question.correctAnswer) {

score++;

}

}

wcout << L"Your score: " << score << L" out of " << questions.size() << endl;

testResultManager.saveTestResults(dbManager, username, 1, score); // Assuming testId is 1 for simplicity

}

};

### 3.1.10. Класс Logger

Назначение класса: Шаблонный класс для логирования сообщений в файл. Поддерживает логирование сообщений типа std::string и std::wstring.

Описание методов класса:

|  |  |  |
| --- | --- | --- |
| № | Сигнатура | Назначение |
| 1 | static void log(const T& message, const std::wstring& type = L"info") | Метод для записи логов в файл. Поддерживает два типа сообщений: std::string и std::wstring. |
| 2 | static std::wstring convertToWString(const std::string& str) | Приватный метод для преобразования std::string в std::wstring. |
| 3 | static const std::wstring& convertToWString(const std::wstring& wstr) | Приватный метод для возврата ссылки на std::wstring без изменений. |

В классе использованы статические поля

template<typename T>

class Logger {

public:

static void log(const T& message, const std::wstring& type = L"info") {

static\_assert(std::is\_same<T, std::wstring>::value || std::is\_same<T, std::string>::value,

"Logger only supports std::string and std::wstring");

wofstream logFile("log.txt", ios::app);

logFile.imbue(std::locale(logFile.getloc(), new std::codecvt\_utf8<wchar\_t>)); // Set UTF-8 encoding

logFile << type << L": " << convertToWString(message) << endl;

logFile.close();

}

private:

static std::wstring convertToWString(const std::string& str) {

std::wstring wstr(str.begin(), str.end());

return wstr;

}

static const std::wstring& convertToWString(const std::wstring& wstr) {

return wstr;

}

};

### 3.1.11. Класс DatabaseException

Назначение класса: Класс для обработки исключений базы данных. Наследуется от std::exception.

Описание методов класса:

|  |  |  |
| --- | --- | --- |
| № | Сигнатура | Назначение |
| 1 | DatabaseException(const std::string& msg) | Конструктор, принимающий сообщение об ошибке. |
| 2 | const char\* what() const noexcept override | Метод для получения сообщения об ошибке. |

В данном классе реализованы исключения для обработки ошибок базы данных.

class DatabaseException : public exception {

private:

string message;

public:

DatabaseException(const string& msg) : message(msg) {}

const char\* what() const noexcept override {

return message.c\_str();

}

};

### 3.1.12. Абстрактный класс IManagable

Назначение класса: Класс задает интерфейс выполнения задач для классов Admin и Employee.

Описание методов класса:

|  |  |  |
| --- | --- | --- |
| № | Сигнатура | Назначение |
| 1 | virtual void performTasks(DatabaseManager& dbManager) = 0 | Чисто виртуальный метод для выполнения задач. |

В абстрактном классе использованы виртуальные методы для задания интерфейса выполнения задач

class IManageable {

public:

virtual void performTasks(DatabaseManager& dbManager) = 0;

virtual ~IManageable() {}

};

3.1.12. Тестирование

Для входа в систему надо зайти под логином и паролем соответствующего типу пользователю. Конфиденциальные данные хранятся в базе данных.

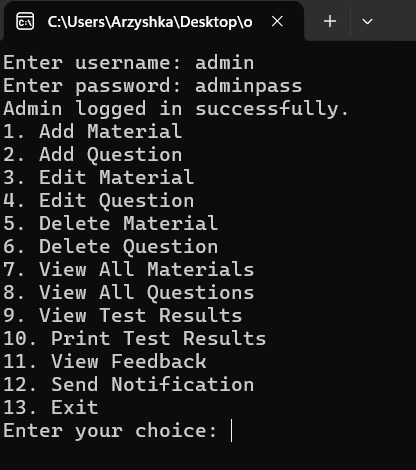


Рисунок 3.1 Успешный вход администратора и список доступных ему операций

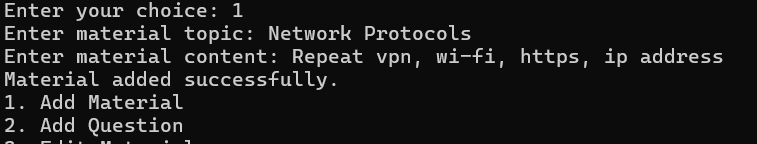


Рисунок 3.2 Выбор «Добавление материала»

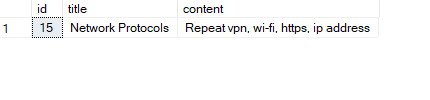


Рисунок 3.2 Сохранение добавленного материала в БД

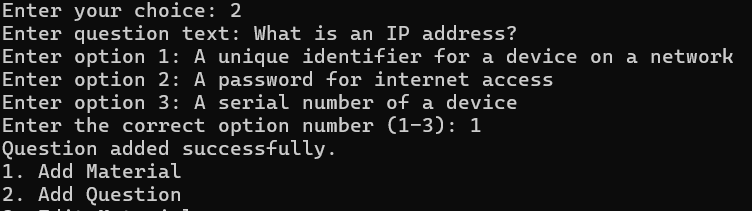


Рисунок 3.3 Добавление вопросов с вариантам ответа

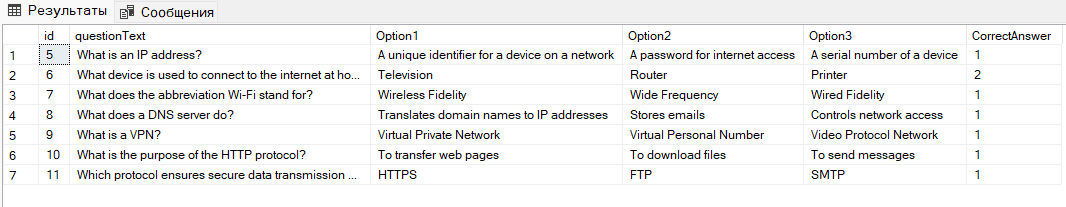


Рисунок 3.4 Сохранение вопросов с вариантами ответа в БД

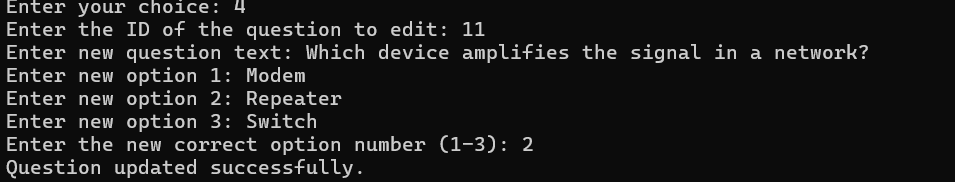


Рисунок 3.5 Обновление вопроса под ID 11

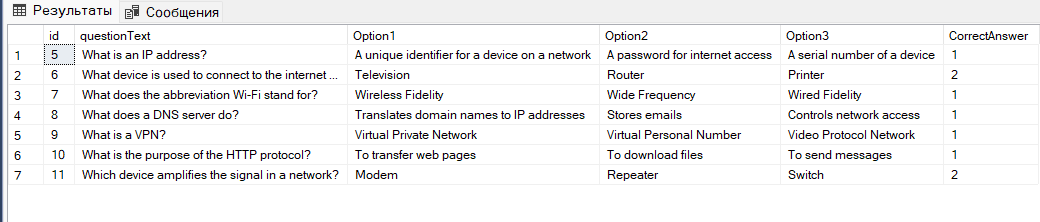


Рисунок 3.6 Обновление 11 вопроса и в БД

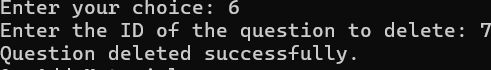


Рисунок 3.7 Удаление вопроса под ID 7

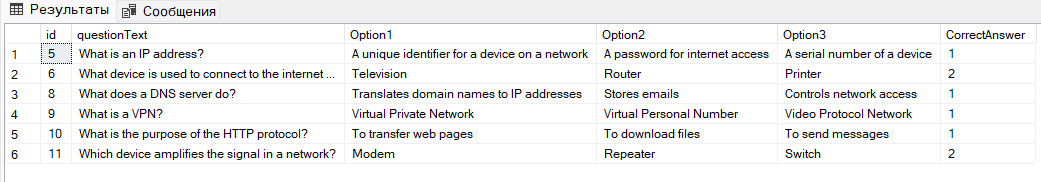


Рисунок 3.8 БД после удаления 7 вопроса

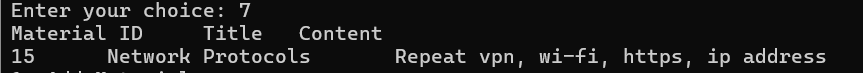


Рисунок 3.9 Выбор действия «Просмотр всех материалов»

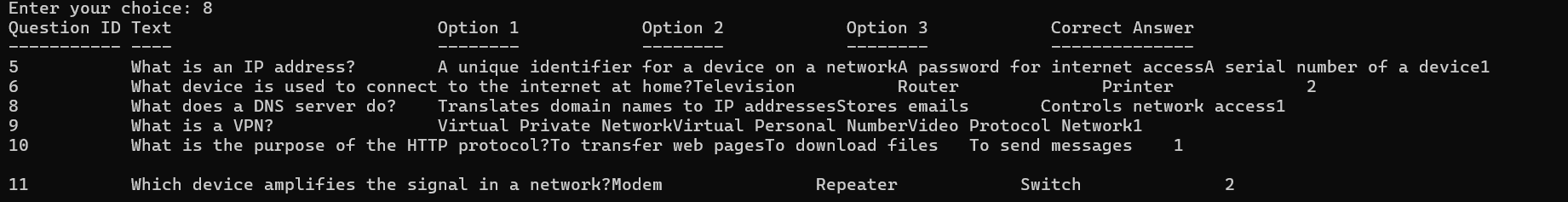


Рисунок 3.10 Выбор действия «Просмотр всех вопросов»



Рисунок 3.11 Вход сотрудника в систему и список доступных ему действий



Рисунок 3.12 Выбор действия сотрудника «Просмотр всех материалов»

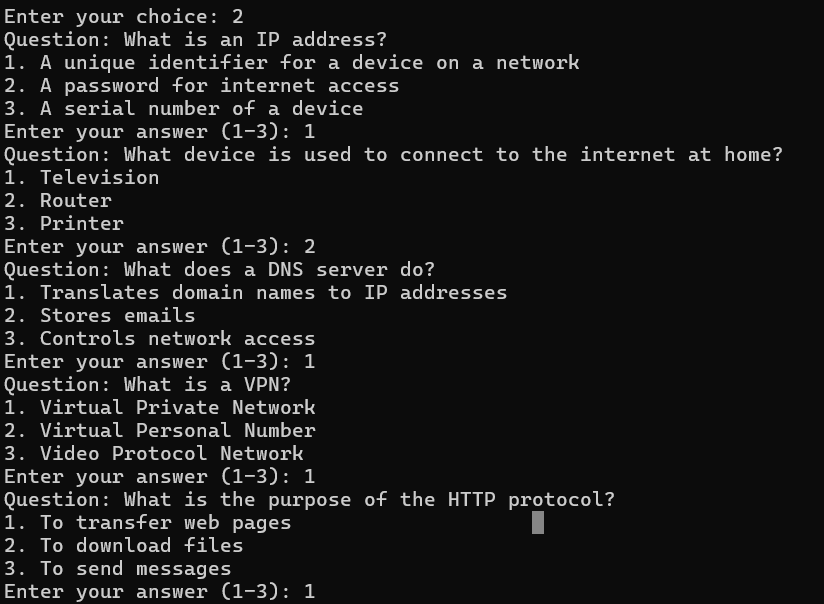


Рисунок 3.13 Выбор действия сотрудника «Сдать тест»

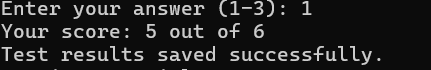


Рисунок 3.14 Подсчет результата теста сотрудника

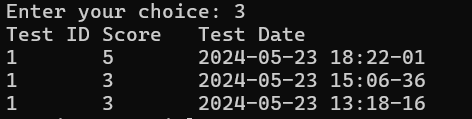


Рисунок 3.15 Выбор действия сотрудника «Посмотреть мои результаты тестов»

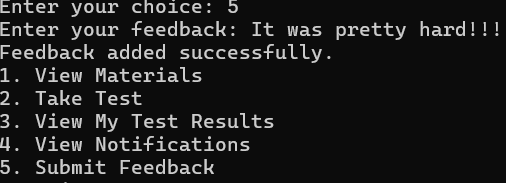


Рисунок 3.16 Выбор действия сотрудника «Оставить обратную связь»

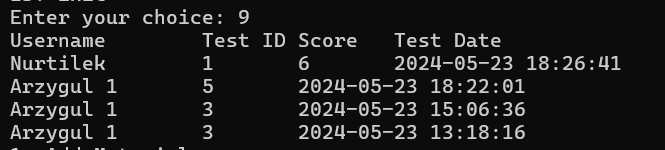


Рисунок 3.17 Выбор действия администратора «Посмотреть результаты сотрудников»

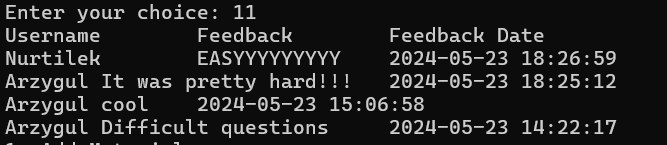


Рисунок 3.18 Выбор действия администратора «Сообщения обратной связи»



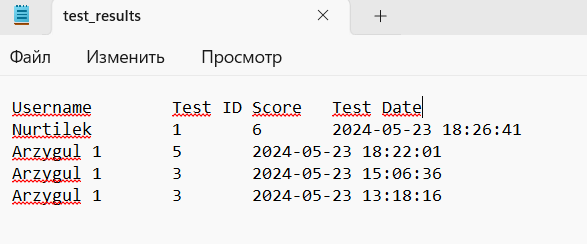


Рисунок 3.19 Выбор администратора «Печать результата» и содержание файла test\_results.txt

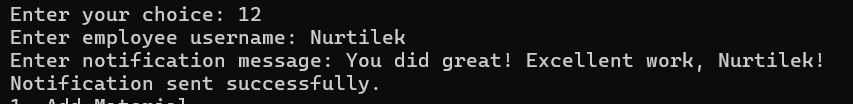


Рисунок 3.20 Выбор действия администратора «Отправить уведомление сотруднику»

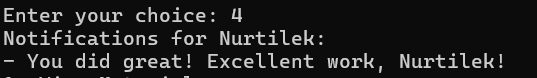


Рисунок 3.21 Выбор действия сотрудника «Посмотреть уведомление»

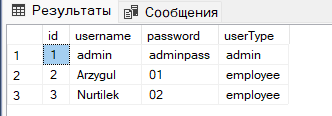


Рисунок 3.22 Хранение данных работников в БД

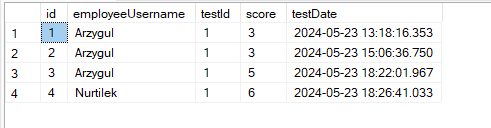


Рисунок 3.23 Хранение результатов тестов сотрудников в БД

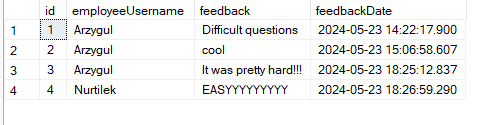


Рисунок 3.24 Хранение сообщений обратной связи в БД

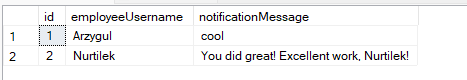


Рисунок 3.25 Хранение уведомлений от администратора в БД

3.1.13 Заключение

В представленной реализации кода для системы управления обучением сотрудников я задействовала и успешно реализовала несколько важных концепций объектно-ориентированного программирования, такие как наследование, перегрузка методов, шаблоны, исключения, виртуальные методы и статические поля.

Наследование: В коде используется наследование для создания иерархии классов пользователей системы. Класс User является базовым для классов Admin и Employee, которые наследуют от него и расширяют его функциональность, реализуя метод performTasks.

Перегрузка методов: В классе User и его наследниках используются перегруженные методы, такие как authenticate, для предоставления различного функционала аутентификации пользователей.

Шаблоны: Шаблонный класс Logger предоставляет возможность логирования сообщений с поддержкой разных типов данных (std::string и std::wstring), демонстрируя использование шаблонов для создания гибких и многократно используемых компонентов.

Исключения: В коде предусмотрена обработка исключений с использованием класса DatabaseException, который наследуется от std::exception и предоставляет возможность обработки ошибок, связанных с базой данных.

Виртуальные методы: Абстрактный класс IManageable содержит чисто виртуальный метод performTasks, который должен быть реализован в каждом из классов-наследников (Admin и Employee). Это позволяет обеспечить единый интерфейс для выполнения задач разными типами пользователей.

Статические поля: Статические методы в классе Logger, такие как log, обеспечивают возможность логирования без необходимости создания экземпляра класса, что упрощает использование этого функционала в различных частях программы.

Программа реализует основную функциональность системы управления обучением сотрудников, включая аутентификацию пользователей, управление учебными материалами и вопросами, сохранение и просмотр результатов тестов, отправку уведомлений и сбор отзывов. В итоге, в ходе данного проекта я научилась продумывать и качественному применению принципов объектно-ориентированного программирования для построения любых систем, которая может быть использована в реальных условиях для управления процессом обучения сотрудников в организации.

Код проекта можно просмотреть по ссылке: [ссылка](mailto:git@github.com:makenovaarzygul/oop_coursework3.git)