МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний економічний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерних наук

ВАЛЬЧИШИН Андрій Петрович

Підсистема "Студент" програмної системи тестування студентів/ "Student" subsystem of software system for students testing

напрям підготовки: 6.050103 - Програмна інженерія
фахове спрямування - Програмне забезпечення систем

Бакалаврська дипломна робота

Виконав студент групи ПЗС-42
А. П. Вальчишин

_____

Науковий керівник:
викладач КРЕПИЧ С.Я.

_____

Бакалаврську дипломну роботу
допущено до захисту:

"___" _____ 20___ р.

Завідувач кафедри
_____ **А. В. Пукас**

ТЕРНОПІЛЬ - 2016

РЕЗЮМЕ

Дипломна робота містить 90 сторінок, 20 таблиць, 30 рисунків, список використаних джерел із 23 найменувань та 2 додатка.

Метою дипломної роботи є розробка програмного забезпечення та бази даних для автоматизації процесу тестування для студента.

Об’єкт дослідження – процес автоматизованого тестування студентів по SQL запитах зі сторони студента.

Предмет досліджень – застосування сучасних технологій створення додатків для розробки програмної підсистеми «Студент» системи тестування студентів по SQL запитах.

Методи розробки базуються на технології .NET, бази даних MS SQL Server.

Одержані результати полягають в розробці підсистеми “Студент” програмної системи для тестування студентів по SQL запитах.

Ключові слова: студент, запит, програмний комплекс, база даних, діаграма класів, реляційна модель бази даних, тестування.

SUMMARY

Thesis contains 90 pages, 10 tables, 30 figures, list of sources with 23 titles and 2 applications.

The aim of the thesis is the development of software and databases to automate the testing process for student.

The object of the research is an automated process of student SQL knowledge testing for student.

The subject of research is the use of modern information technology to create a subsystem "Student" of the system for testing students on SQL queries.

Methods of development based on technology .NET, database MS SQL Server.

The results are in development of subsystem "Student" of the system for testing students on SQL queries.

Keywords: student, query, software system, database, diagram classes relational model database testing.

ЗМІСТ

ВСТУП

Актуальність дипломної роботи полягає в тому, що на сьогоднішній день не існує такої програмної системи, яка б об’єднувала процеси оцінки відповідно до завдання та виконання SQL запитів в одному програмному продукті. На даний момент проведення тестування знань студентів з мови структурованих запитів SQL пов’язане з великою кількістю труднощів. Насамперед, тестування в основному може проводитись в середовищі проектування чи діалогу з СУБД, які не пристосовані до проведення тестування, результати запитів не оцінюються автоматично а вимагають безпосередньої участі викладача, який буде проводити оцінку кожної спроби. Цей недолік робить процес тестування не ефективним та ресурсоємким, викладач змушений витрачати зусилля та час на індивідуальну перевірку виконаних завдань кожним студентом. Іншим шляхом проведення тестування знань з SQL є простий тест з порівнянням стрічки запиту з тою, яку вважаємо правильною. Цей спосіб в своєму корені невірний - адже існує безліч SQL запитів, які будуть правильною відповіддю для задачі та залежать від стилю написання кожного студента. Таким чином, обмеження в простій перевірці стрічок не охоплює більшість можливих варіантів правильної відповіді.

Існує третій спосіб - оцінка та виконання запитів здійснюється автоматично і в одному середовищі. Об’єднання цих двох процесів дозволяє оцінити правильність відповіді на основі результату виконання запиту та усуває недоліки двох попередніх способів. Однак, на даний момент аналогів подібного середовища не існує, а тому виникає необхідність в розробці окремої програмної системи, яка працювала б в прямому діалозі з СУБД, містила базовий функціонал для проведення тестування та дозволяла оцінювати правильність виконання запитів порівнюючи результати їх виконання з правильними.

Мета роботи полягає в дослідженні та детальному аналізі уже існуючих систем для тестування, які дозволяють створювати тестові завдання та перевіряти результати автоматично.

Отож, спираючись на мету, можна виділи такі основні задачі, які ставляться до розроблюваного продукту:

1. Дослідити існуючі аналоги програмних систем для тестування. Виділити їх основні позитивні та негативні риси.
2. На основі виділених рис розробити функціональні та не функціональні вимоги до розроблюваного продукту.
3. Розробити модель бази даних розроблюваного продукту.
4. Обрати необхідні технології для реалізації обраного функціоналу.
5. Розробка програмної підсистеми “Студент” для тестування знань студентів, описаних в пунктах 1-4.

Об’єкт дослідження – процес автоматизованого тестування студентів по SQL запитах зі сторони студента.

Предмет досліджень – застосування сучасних технологій створення додатків для розробки програмної підсистеми «Студент» системи тестування студентів по SQL запитах.

Практична цінність розроблюваної програмної системи полягає в автоматизації тестування студентів по SQL запитах, виконання тестових завдань, оцінки результатів.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Коротка характеристика об'єкту управління

Тестування - це спосіб визначення рівня знань і вмінь студентів за допомогою спеціальних тестових завдань. Комп'ютерне тестування здійснюється у формі самостійного діалогу студента з комп’ютером у присутності/відсутності відповідальної за організацію тестування особи, з можливістю запам'ятовування результатів тестування. Студент проходить тест з використанням комп’ютерного додатку. Завдання та відповіді попередньо сформовані викладачем, результати проходження тестів записуються та згодом можуть бути переглянуті викладачем.

Користувач, авторизувавшись, отримує список тестів, які відповідають його групі та може вибрати конкретне завдання для проходження. Завдання містить опис та ЕРД. Відповідно до цієї інформації студент записує SQL-запит у поле вводу відповіді. Коли введення завершено, запит відправляється на сервер СУБД, де він і виконується, оперуючи тестовими даними. Сервер повертає відповідь і система перевіряє її відповідність із зашифрованою правильною відповіддю, яка прикріплялась до завдання в БД організації тестування. Відповідно до перевірки та значення складності завдання, результат впливає на загальну оцінку проходження тестування студентом. Відсутність відповідей на завдання розцінюється як неправильна відповідь. Студент має змогу переглянути свої поточні результати проходження тестування в списку завдань. Всі введені відповіді студента записуються в БД організації тестування з метою надання викладачу можливості їх перегляду для формування статистичного представлення виконання завдань та підсумовування результатів для покращення якості завдань. З метою забезпечення безпеки правильна відповідь студента в статистичний результат не записується.

1.2. Опис предметної області

Предметною областю даної дипломної роботи є тестування знань студента з мови SQL, шляхом створення програмної системи, яка дозволить проходити сформовані викладачем завдання та перевірити їх виконання.

Для того, щоб повністю представити функціонал програмної системи, виділено такі основні бізнес-процеси:

- процес вибору завдання;
- процес перегляду умови завдання;
- процес виконання завдання;

Рисунок 1.1 – Діаграма бізнес-процесів розроблюваного програмного продукту

Характеристику бізнес-процесу вибору завдання наведено в таблиці 1.1.

Таблиця 1.1

Характеристика бізнес-процесу вибору завдання

Назва характеристики	Значення характеристики
Ім'я бізнес-процесу	Вибір завдання
Основні учасники	Студент
Вхідна подія	Авторизація

Вхідні документи	Список завдань

Продовження таблиці 1.1

Вихідна подія	Виконання вибраного завдання
Вихідні документи	-
Клієнт бізнес-процесу	-

На рисунку 1.2 зображено процес перегляду умови завдання.

Рисунок 1.2 – Діаграма бізнес-процесів перегляду умови завдання

Перед тим як студент приступить до виконання завдання, йому необхідно переглянути умову завдання. Умову завдання характеризують такі особливості:

- ERD – діаграма «сутність-зв'язок», яка дає змогу побачити структуру БД;
- Опис – текстовий опис завдання;

Характеристику бізнес-процесу перегляду умови завдання наведено в таблиці 1.2.

Таблиця 1.2

Характеристика бізнес-процесу перегляду умови завдання

Назва характеристики	Значення характеристики
Ім'я бізнес-процесу	Перегляд умови завдання
Основні учасники	Студент
Вхідна подія	Користувач вибрав опцію перегляду умов завдання
Вхідні документи	Умови завдання
Вихідна подія	Відображення умов завдання
Вихідні документи	-
Клієнт бізнес-процесу	-

На рисунку 1.3 зображено процес виконання завдання.

Рисунок 1.3 – Діаграма бізнес-процесів виконання завдання

Процес виконання завдання дає змогу студенту ввести свій SQL запит та получити результати перевірки. Він поділяється на такі частини:

- виконання запиту;
- порівняння результатів запиту з правильними;
- виставлення оцінки за результатами перевірки та складності завдання.

Характеристику бізнес-процесу виконання тестового завдання наведено в таблиці 1.3.

Таблиця 1.3

Характеристика бізнес-процесу виконання завдання

Назва характеристики	Значення характеристики
Ім'я бізнес-процесу	Виконання завдання
Основні учасники	Студент
Вхідна подія	Введення відповіді
Вхідні документи	Введений запит
Вихідна подія	Перевірка запиту
Вихідні документи	Оцінка
Клієнт бізнес-процесу	-

1.3. Огляд і аналіз існуючих аналогів

Сьогодні є достатньо програмних систем, за допомогою яких можна проводити тестування студентів. Ми проаналізуєм кілька таких.

Розглянемо для початку програму тестування знань “Айрен”

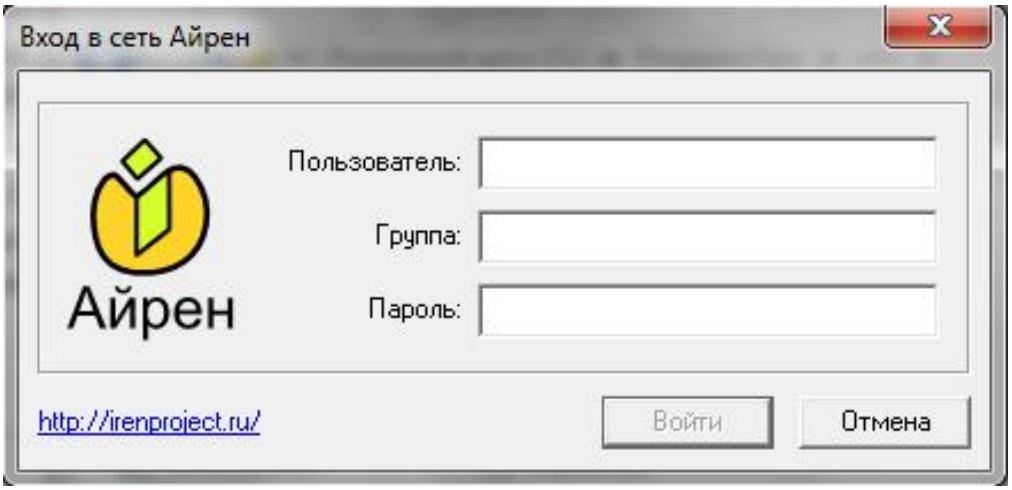

Рисунок 1.4 – Вікно авторизації у програмі «Айрен»

Ця програма має примітивний але доволі зрозумілий та простий для сприйняття інтерфейс. Присутня можливість авторизації користувача, проходження набору тестових завдань. Тестове завдання може мати різні способи вводу відповіді: від вибору варіанту до введення результату в задане поле. Недоліком цього програмного продукту є те, що у такій програмі немає можливості перевіряти та виконувати SQL запит, тому вона не підходить для виконання таких завдань, як перевірки знань студентів з SQL запитів.

Отже, дана програмна система має наступні функції:

- можливість проходження набору тестів;
- базова авторизація;
- визначення загальних результатів виконання набору тестів;
- можливість виконання тестів з різними способами вводу відповідей.

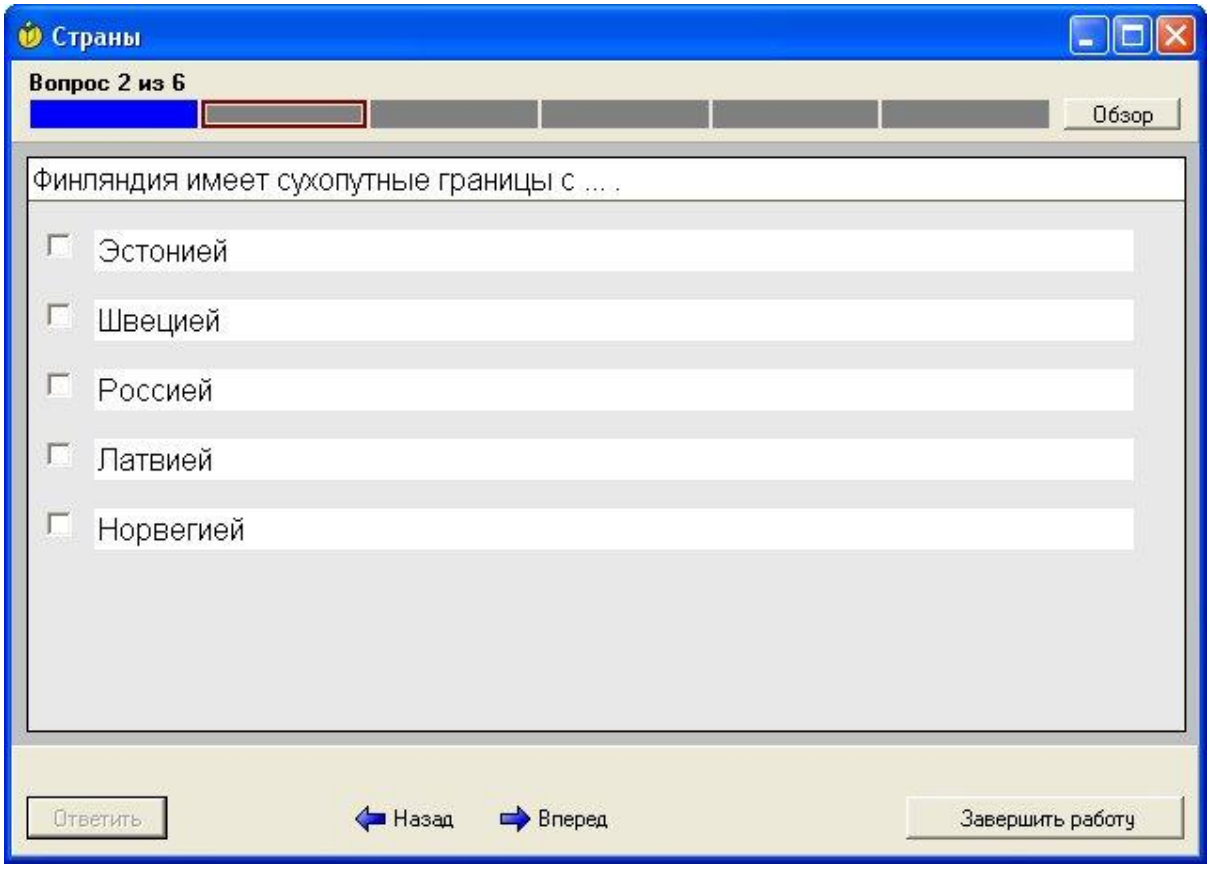

Рисунок 1.5 – Вікно проходження тестування у програмі «Айрен»

Для порівняння також розглянемо програмний продукт MyTestStudentPRO

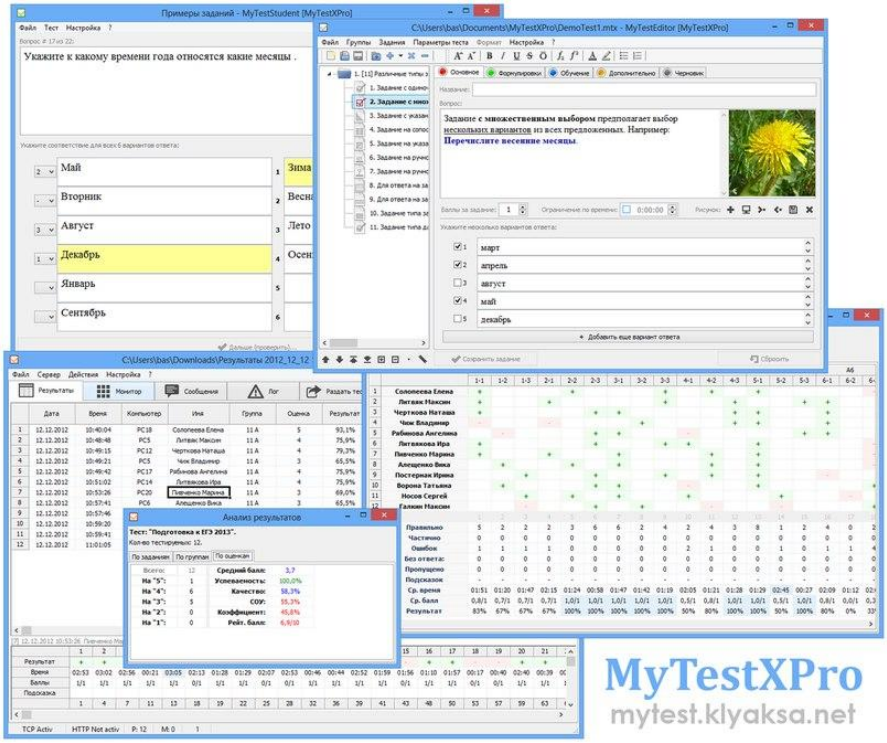

Рисунок 1.6 – Набір модулів MyTestStudentPRO

Цей продукт має два модуля: модуль для викладача, де можна створювати завдання, та модуль для студента, де останній ці тести проходить. Тести назначаються для певної групи студентів, присутня авторизація. Тестові завдання можуть містити різні способи вводу відповідей та можуть проходитись наборами. Також присутня система оцінки результатів проходження тестування в залежності від складності завдання. Даний продукт має багато різноманітних функцій, таких як:

- можливість проходження набору тестів;
- визначення загальних результатів виконання набору тестів;
- можливість виконання тестів з різними способами вводу відповідей;

- можливість авторизації користувача та виконання тестів назначених для нього;
- автоматичне оцінювання за заданою шкалою.

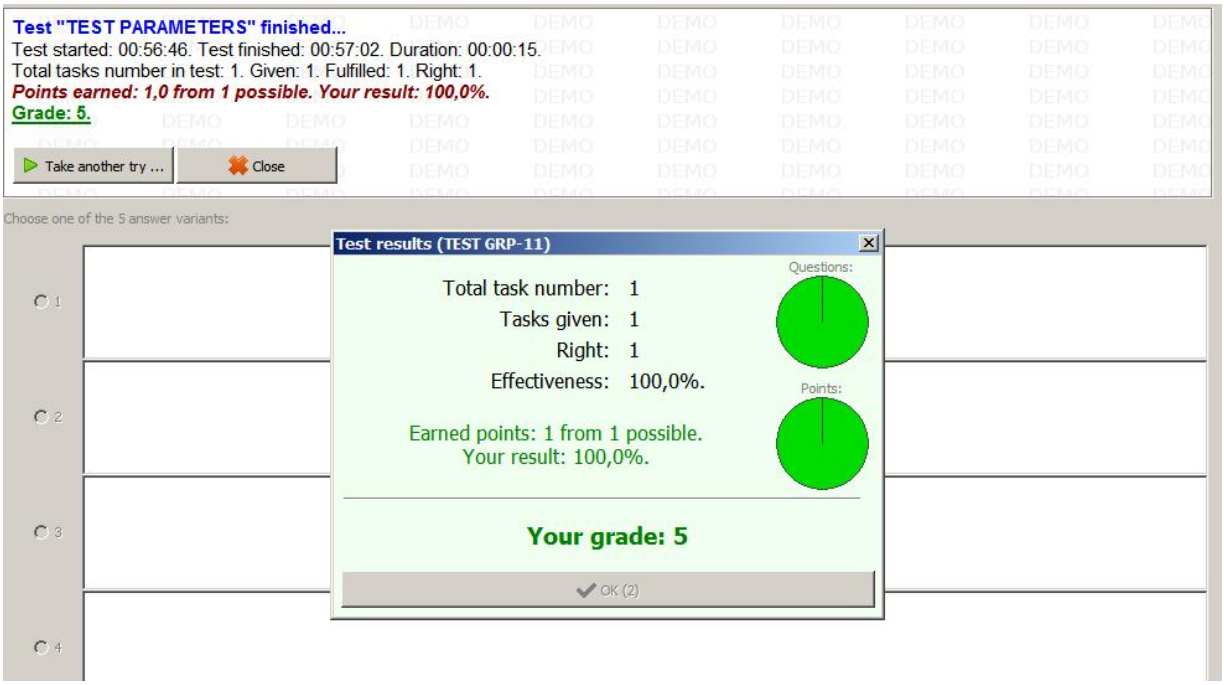

Рисунок 1.7 – Сторінка проходження завдання програми MyTestStudentPRO

Отже, можна зробити висновок, що вище перераховані програмні системи є доволі хорошими для проведення простих текстових тестів. Проте, ні одна з них не дає змогу виконувати тести, які давали б змогу виконати SQL запит та порівняти результат його виконання із правильним. Єдиним способом для таких систем є перевірка тексту самого запиту, а не результату виконання, що є дуже неякісним підходом до тестування. Тому, оптимальним рішенням даної проблеми є створення продукту для тестування студентів з можливістю ввести SQL запит, виконати на сервері та перевірити сам результат виконання з правильним результатом завдання.

Здійснивши аналіз альтернативних рішень, створено таблицю, де відображається порівняльна характеристика аналогів.

Таблиця 1.4

Порівняльна характеристика аналогів

Фірма-розробник	Сергій Останін	Олександр Башлаков
Назва програмного продукту	Програма тестування знань “Айрен”	MyTestStudentPRO
Версії продукту	2.3	11.0.0.37
Функціональність	• Можливість проходження набору тестів • Визначення загальних результатів виконання набору тестів • Можливість виконання тестів з різними способами вводу відповідей • Базова авторизація	• Можливість проходження набору тестів • Визначення загальних результатів виконання набору тестів • Можливість виконання тестів з різними способами вводу відповідей • Можливість авторизації користувача та виконання тестів назначених для нього • Автоматичне оцінювання за заданою шкалою
Допомога користувачу	Присутня система допомоги користувачу та вікі проекту	Присутня система допомоги користувачу та вікі проекту

1.4. Специфікація вимог до модуля (системи)

Специфікація вимог для програмної системи - це повний опис поведінки системи що розробляється. Вона включає множину прецедентів які описують всі взаємодії, які користувачі мають з програмним забезпеченням. Прецеденти також відомі як функціональні вимоги. На додачу до прецедентів також включає нефункціональні вимоги. Нефункціональні вимоги є вимогами які накладають обмеження на проект, чи реалізацію. Специфікація вимог до системи включає: глосарій проекту, опис варіантів використання.

У таблиці 1.5 подано глосарій основних використовуваних термінів при створенні модуля «Студент» системи тестування знань студента.

Таблиця 1.5

Глосарій основних термінів

Термін	Опис терміну
1. Основні поняття та категорії предметної області та проекту	
База даних	Впорядкований набір логічно взаємопов'язаних даних, що використовуються спільно та призначені для задоволення інформаційних потреб користувачів. У технічному розумінні цей термін включає і систему керування базами даних.
Таблиця бази даних	Набір даних, які організовані моделлю таблиці з вертикальних стовпцями та горизонтальними рядками. В такій таблиці кількість стовпців є визначена і відповідає кількості полів сутності, а кількість радків може відрізнятись в різні моменти часу та представляють собою записи (об’єкти) сутності. Кожен рядок ідентифікується унікальним значенням в спеціальному стовпці, який називається первинним ключем.
Реляційна база даних	База даних, основана на реляційній моделі даних. Таблиці в такій базі даних звязані між собою звязками з використанням зовнішніх ключів. Така база даних, сприймається користувачем як набір нормалізованих відношень різного ступеню.

SQL (Structured query language)	Декларативна мова програмування для взаємодії з базами даних. Використовується для формування запитів зчитування, створення, оновлення, видалення і керування реляційними БД, створення схеми бази даних і її модифікації, системи контролю за доступом до бази даних.

Продовження таблиці 1.5

Модель «сутність-зв'язок» (ER-модель)	Модель даних, яка дозволяє описувати концептуальні схеми за допомогою узагальнених конструкцій блоків. ER-модель — це мета-модель даних, тобто засіб опису моделей даних.
Система керування базами даних	Комп'ютерна програма чи комплекс програм, що забезпечує користувачу можливість взаємодії з базою даних.
Тестування	Це процес технічного дослідження, призначений для оцінки якості продукту відносно контексту, в якому він має використовуватись. Тестування також включає процес пошуку помилок і дефектів та випробування компонентів системи з метою їх оцінки.
2. Користувачі системи	
Студент	Особа, що використовує модуль «Студент» системи тестування знань студента для того, щоб пройти тестування за заданими викладачем завданнями.
3. Вхідні та вихідні документи	
SQL запит	Це сформований за допомогою мови SQL запит до системи керування базами даних. Він є інструментом взаємодії користувача з базами даних.
Тестове завдання	Елементарна складова частина тесту, що містить інструкції та інформацію для розв’язання задачі студентом з метою оцінки його вмінь та навичок.
Дані	Це інформація, подана у формалізованому вигляді, прийнятному для обробки автоматичними засобами.

Далі, наведемо діаграму варіантів використання для акторів системи на рисунку 1.8.

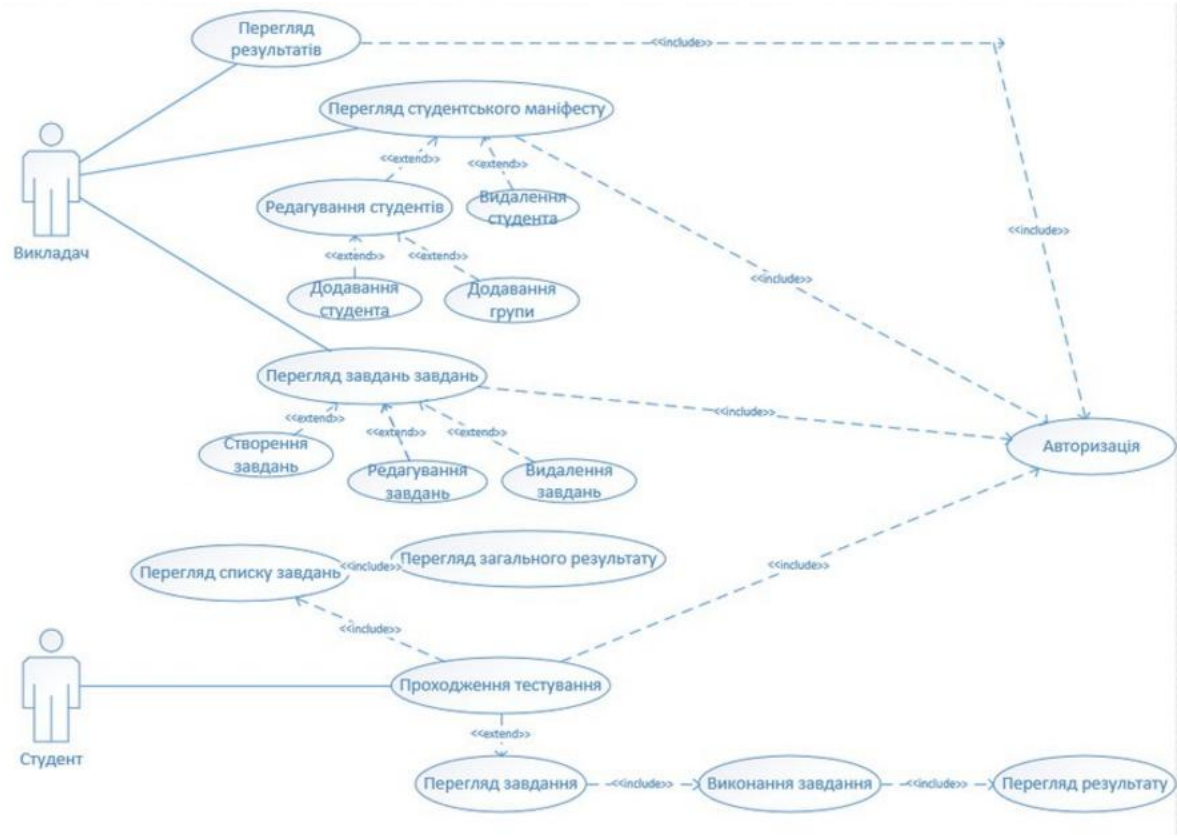

Рисунок 1.8 – Діаграма варіантів використання системи тестування знань студентів

На рисунку 1.9 представлена діаграма варіантів використання модуля «Студент» системи тестування знань студентів

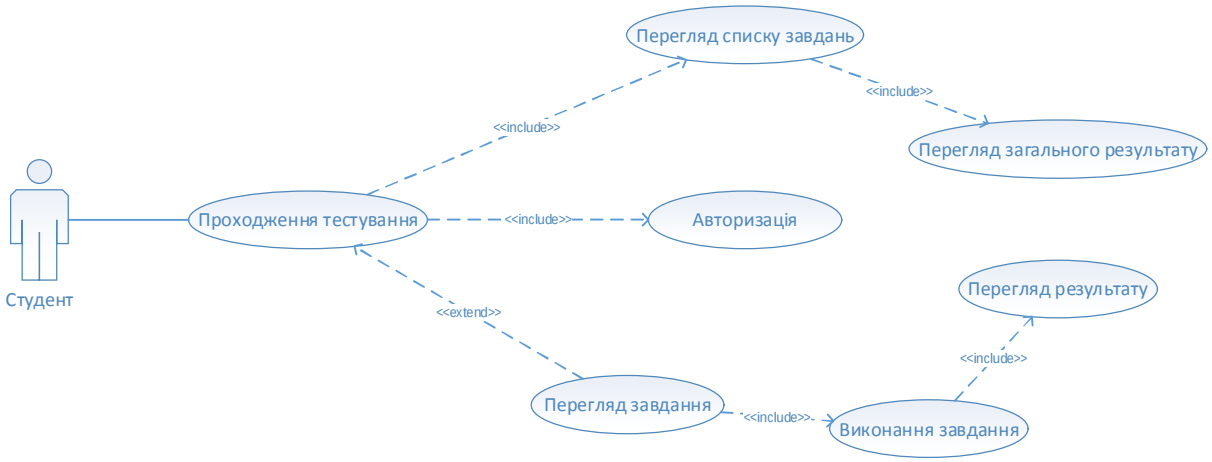

Рисунок 1.9 – Діаграма варіантів використання модуля «Студент» системи тестування знань студентів

Наступним кроком у визначенні специфікацій вимог до системи буде опис варіантів використання.

Варіанти використання представленні у таблицях 3.6-3.11

Варіант використання «Перегляд результату» передбачає перегляд користувачем результату перевірки відповіді на завдання.

Таблиця 1.6

Варіант використання «Перегляд результату»

Контекст використання	Переглянути результат виконання завдання
Дійові особи	Студент
Передумова	Виконане завдання
Триггер	Завершення виконання завдання
Сценарій	Користувач після завершення виконання завдання отримує результат
Післяумова	-

Варіант використання «Виконання завдання» передбачає введення користувачем відповіді на завдання.

Таблиця 1.7

Варіант використання «Виконання завдання»

Контекст використання	Введення запиту відповідно до завдання
Дійові особи	Студент
Передумова	Відкрите вікно перегляду завдання

Продовження таблиці 1.7

Триггер	Відкриття вікна перегляду завдання
Сценарій	Користувач переглянувши завдання вводить відповідний запит
Післяумова	Перегляд результату

Варіант використання «Авторизація» передбачає ідентифікацію користувача в системі та виявлення доступних для нього завдань.

Таблиця 1.8

Варіант використання «Авторизація»

Контекст використання	Авторизизація в системі з метою ідентифікації
Дійові особи	Студент
Передумова	Запуск додатку
Триггер	Запуск додатку
Сценарій	Користувач вводить авторизаційні дані
Післяумова	Проходження тестування

Варіант використання «Перегляд завдання» передбачає перегляд умови завдання з метою його подальшого виконання.

Таблиця 1.9

Варіант використання «Перегляд завдання»

Контекст використання	Перегляд завдання з метою його виконання
Дійові особи	Студент
Передумова	Користувач вибрав завдання
Триггер	Користувач вибрав завдання
Сценарій	Користувач переглянувши список завдань вибирає бажане завдання та переглядає його
Післяумова	Виконання завдання

Варіант використання «Перегляд загального результату» передбачає перегляд користувачем загальної успішності виконання набору тестових завдань.

Таблиця 1.10

Варіант використання «Перегляд завдання»

Контекст використання	Перегляд прогресу в проходженні тестування
Дійові особи	Студент
Передумова	Користувач авторизувався в системі
Триггер	Користувач авторизувався в системі
Сценарій	Користувач переглядає прогрес в проходженні тестування
Післяумова	Виконання завдання

Варіант використання «Перегляд списку завдань» передбачає перегляд користувачем списку завдань та вибору бажаного.

Таблиця 1.11

Варіант використання «Перегляд списку завдань»

Контекст використання	Перегляд списку завдання з метою вибору бажаного
Дійові особи	Студент
Передумова	Користувач авторизувався в системі
Триггер	Користувач авторизувався в системі
Сценарій	Користувач переглядає список завдань і вибирає бажане
Післяумова	Перегляд завдання

Прототип для функції «Авторизація» зображено на рисунку 1.10

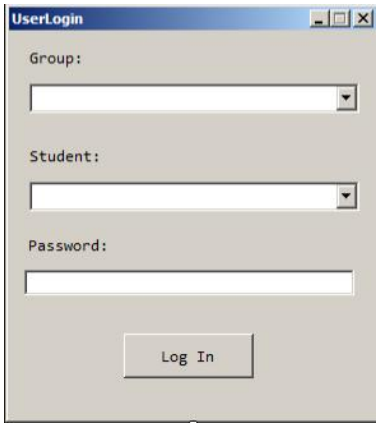

Рисунок 1.10 – Прототип для функції «Авторизація»

Прототип для функції «Перегляд загального результату» на рисунку 1.11

Рисунок 1.11 – Прототип функції «Перегляд загального результату»

Прототип для функції «Виконання завдання» на рисунку 1.12

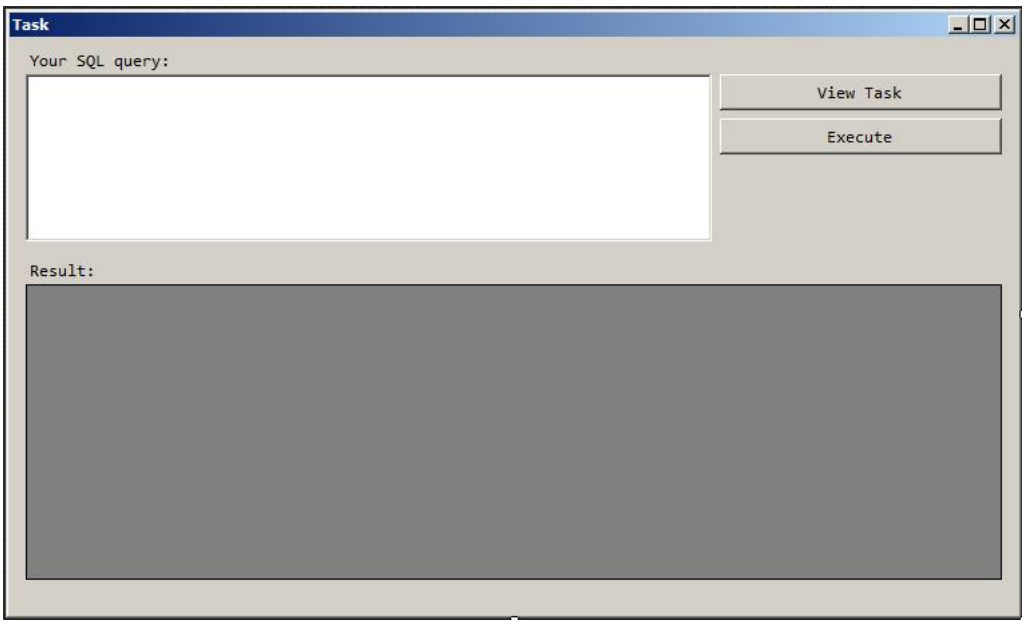

Рисунок 1.12 – Прототип для функції «Виконання завдання»

Прототип для функції «Перегляд завдання» на рисунку 1.13

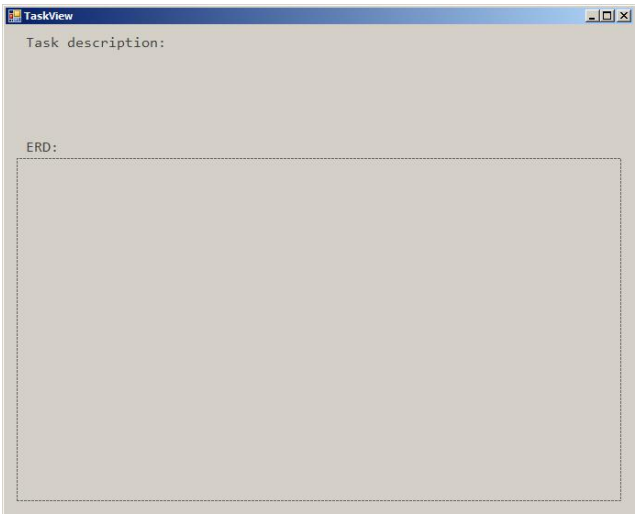

Рисунок 1.13 - Прототип для функції «Перегляд завдання»

Прототип для функцій «Перегляд списку завдань» на рисунку 1.14

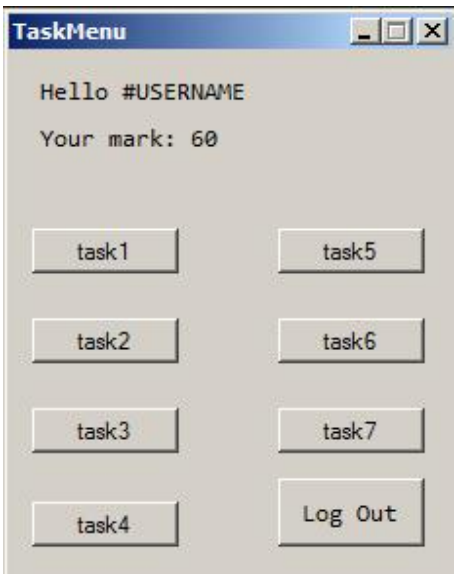

Рисунок 1.14 – Прототип для функції «Перегляд списку завдань»

У таблиці 1.12 наведено специфікацію функціональних вимог.

Таблиця 1.12

Специфікація функціональних вимог

Ідентифікатори вимоги	Назва вимоги	Атрибут вимог		
		Пріоритет	Складність	Контакт
1	Авторизація	Обов’язкове	Низька	Студент
2	Перегляд завдань	Обов’язкове	Середня	Студент
3	Перегляд результату	Обов’язкове	Середня	Студент
4	Виконання завдання	Обов’язкове	Висока	Студент
5	Виконання запиту в СУБД	Обов’язкове	Висока	Студент

Продовження таблиці 1.12

6	Порівняння відповіді сервера з правильною відповіддю	Обов’язкове	Висока	Студент
7	Запис результатів у базу даних	Обов’язкове	Висока	Студент

У таблиці 1.13 наведено специфікацію нефункціональних вимог

Таблиця 1.13

Специфікація нефункціональних вимог

Індетифікатор вимоги	Назва вимоги	Атрибути вимог		
		Пріоритет	Складність	Контакт
1. Застосовність				
1.1	Основні вимоги застосовності нової системи відносно інших систем, які знають користувачі	Рекомендована	Низька	Студент
1.2	Вимоги по відповідальності стандартам графічного інтерфейсу користувача	Обов’язкова	Низька	Студент
1.3	Час, необхідний для навчання звичайних і досвідчених користувачів	Рекомендована	Середня	Студент

Продовження таблиці 1.13

2. Надійність				
2.1	Доступність	Обов’язкова	Середня	Студент
2.2	Середній час безвідмовної роботи	Обов’язкова	Середня	Студент
2.3	Точність	Обов’язкова	Середня	Студент
3. Робочі характеристики				
3.1	Використання ресурсів	Рекомендована	Середня	Студент
4. Проектні обмеження				
4.1	Вимоги до технології програмування	Рекомендована	Середня	Студент

Значення нефункціональних вимог:

- час, необхідний для навчання звичайних користувачів – 20 хвилин;
- час, необхідний для навчання досвідчених користувачів – 15 хвилин;
- основні вимоги застосовності нової системи відносно інших систем, які знають користувачі – всі функції системи є легкими у виконанні, а структура програми не відрізняється від існуючих аналогів;
- вимоги по відповідності загальним стандартам застосовності та стандартам графічного інтерфейсу користувача – програма повинна працювати в операційній системі Windows XP і вище;
- доступність – час, що витрачається на обслуговування системи не повинен перевищувати 3% від загального часу роботи;
- середній час безвідмовної роботи – 2 години;

- використання ресурсів – мінімальні системні вимоги
  - Об’єм HDD: > 500 Мб
  - Об’єм RAM: > 512 Мб
  - Відео пам'ять: > 128 Мб
  - Процесор: > 1 ГГц
  - Клавіатура, маніпулятор миша

Висновок до першого розділу

Проведено аналіз проблеми, досліджено шляхи до її вирішення, оглянуто аналоги програмної системи, розроблено специфікацію вимог до програмного продукту.

РОЗДІЛ 2. РОЗРОБКА ПРОЕКТУ ПРОГРАМНОЇ СИСТЕМИ

2.1. Розробка архітектури програмної системи

Для розробки системи було вибрано трьохрівневу архітектуру. Трирівнева архітектура – включає в себе такі компоненти: клієнтський додаток , підключений до сервера додатків, який в свою чергу підключений до серверу бази даних .

Структурну схему програмної системи зображено на рисунку 4.1.

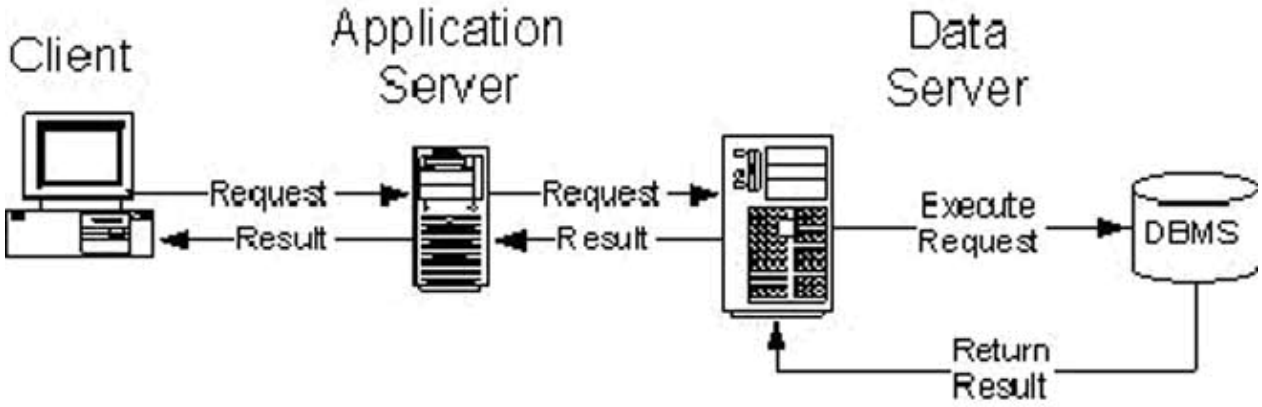

Рисунок 2.1 – Структурна схема програмної системи

Перший рівень - це власне сам додаток для кінцевого користувача. Цей рівень не має прямого зв’язку з базою даних. Тут зосереджений інтерфейс взаємодії користувача з системою.

Другий рівень – рівень сервера. Цей рівень являє собою систему управління базами даних, яка виконує запити додатку в масштабі спеціальної тестової бази даних. Результати таких запитів та можливі помилки надсилаються на перший рівень.

Сервер бази даних забезпечує зберігання даних, необхідних для проходження тестування та містить в собі саму базу даних.

Функціональну структуру системи умовно можна розділити на такі модулі:

- модуль обробки інформації про користувачів:
    1) авторизація;
- модуль обробки інформації про завдання:
    1) перегляд списку завдань;
    2) перегляд умов конкретного завдання;
- модуль виконання завдання:
    1) виконання відповіді-запиту;
    2) порівняння з правильною відповіддю;
    3) оцінка результатів;

Для того, щоб краще зрозуміти процес функціонування системи побудовано діаграми станів для кожного модуля.

Діаграма станів авторизації користувача зображено на рисунку 2.2.

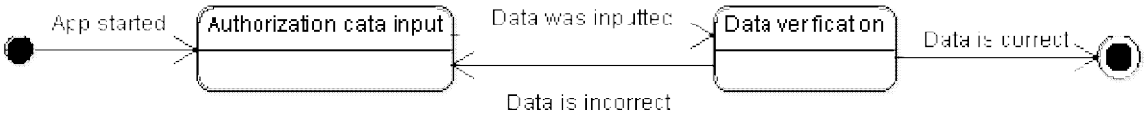

Рисунок 2.2 – Діаграма станів процесу авторизації користувача

Діаграму станів модуля обробки інформації про завдання зображено на рисунку 2.3

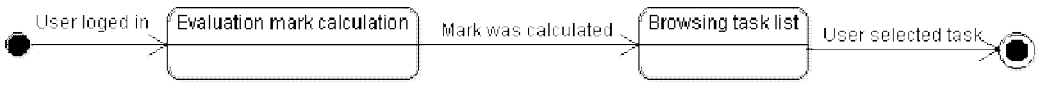

Рисунок 2.3 – Діаграма станів модуля обробки інформації про завдання

Діаграму станів модуля виконання завдання зображено на рисунку 2.4

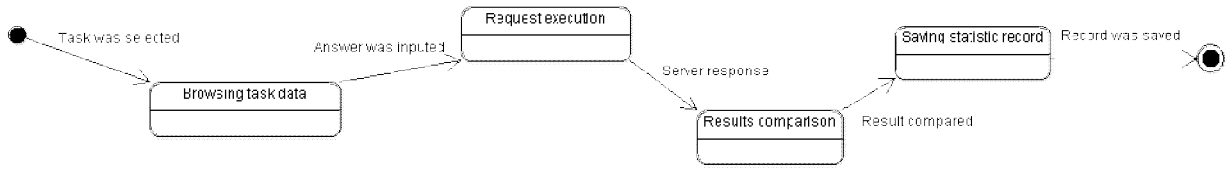

Рисунок 2.4 – Діаграма станів модуля виконання завдання

2.2. Проектування структури бази даних

Етап проектування бази даних (БД) вважається одним із самих складних етапів створення БД, він не має явно вираженого початку й закінчення.

Правильно спроектована база даних забезпечує доступ до найактуальнішої і найточнішої інформації.

Проектування бази даних відбувається в декілька етапів, а саме:

1. Організації необхідної інформації – передбачає збір інформації, яка буде використовуватись в системі
2. Розділення інформації на сутності

3. Формування полів сутностей
4. Створення зв’язків між сутностями

Проектування здійснюється за двома підходами. Перший підхід – визначення основних задач та створення для них бази. Другий підхід представляє з себе визначення предметної області та аналіз усіх даних. Об’єднання цих двох підходів є основним способом проектування структури бази даних.

Проектування бази даних слід розпочинати зі створення DFD-моделі предметної області (див. рис. 2.5).

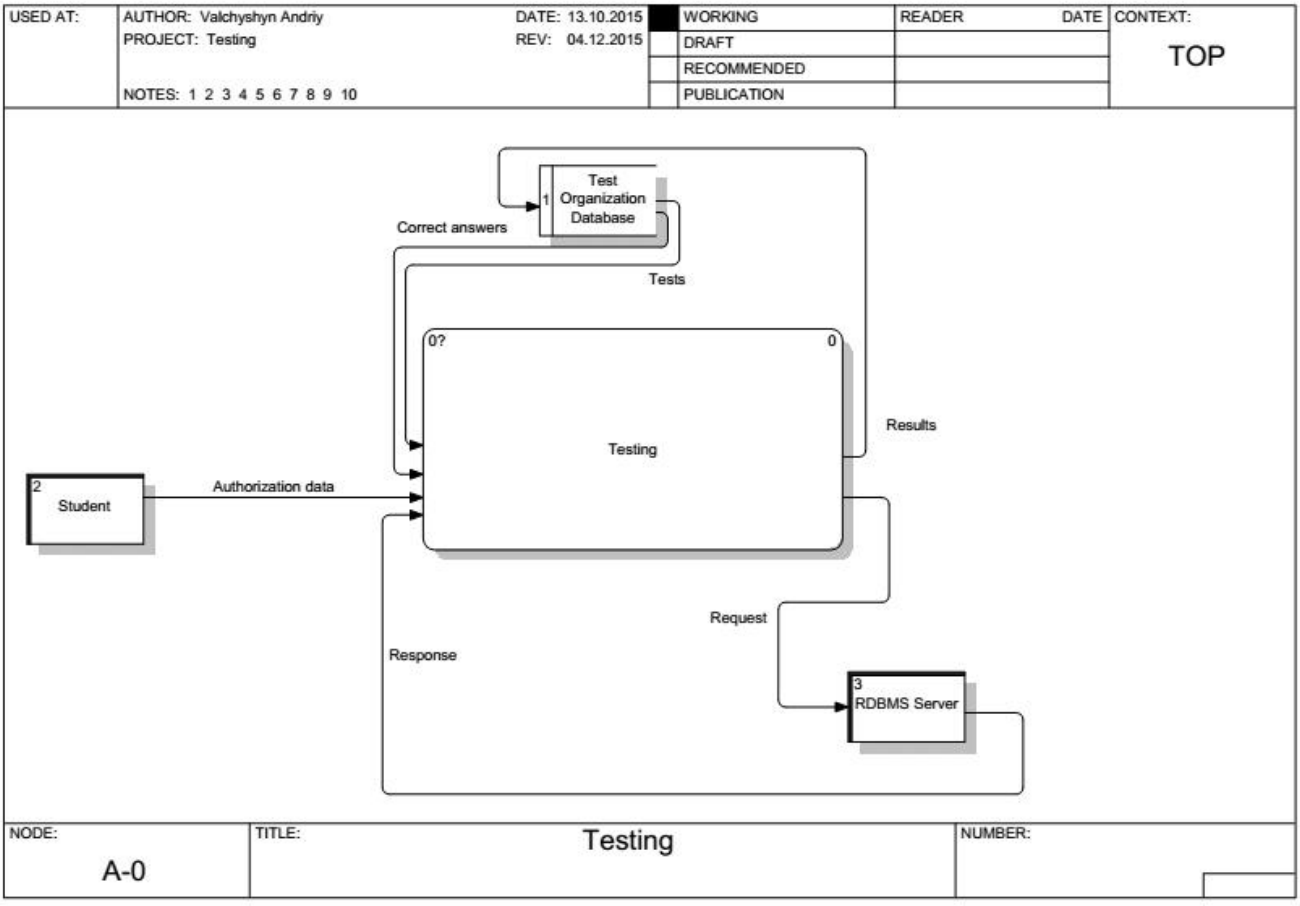

Рисунок 2.5 – Діаграма потоків даних

Діаграма декомпозиції потоків даних зображена на рисунку 2.6.

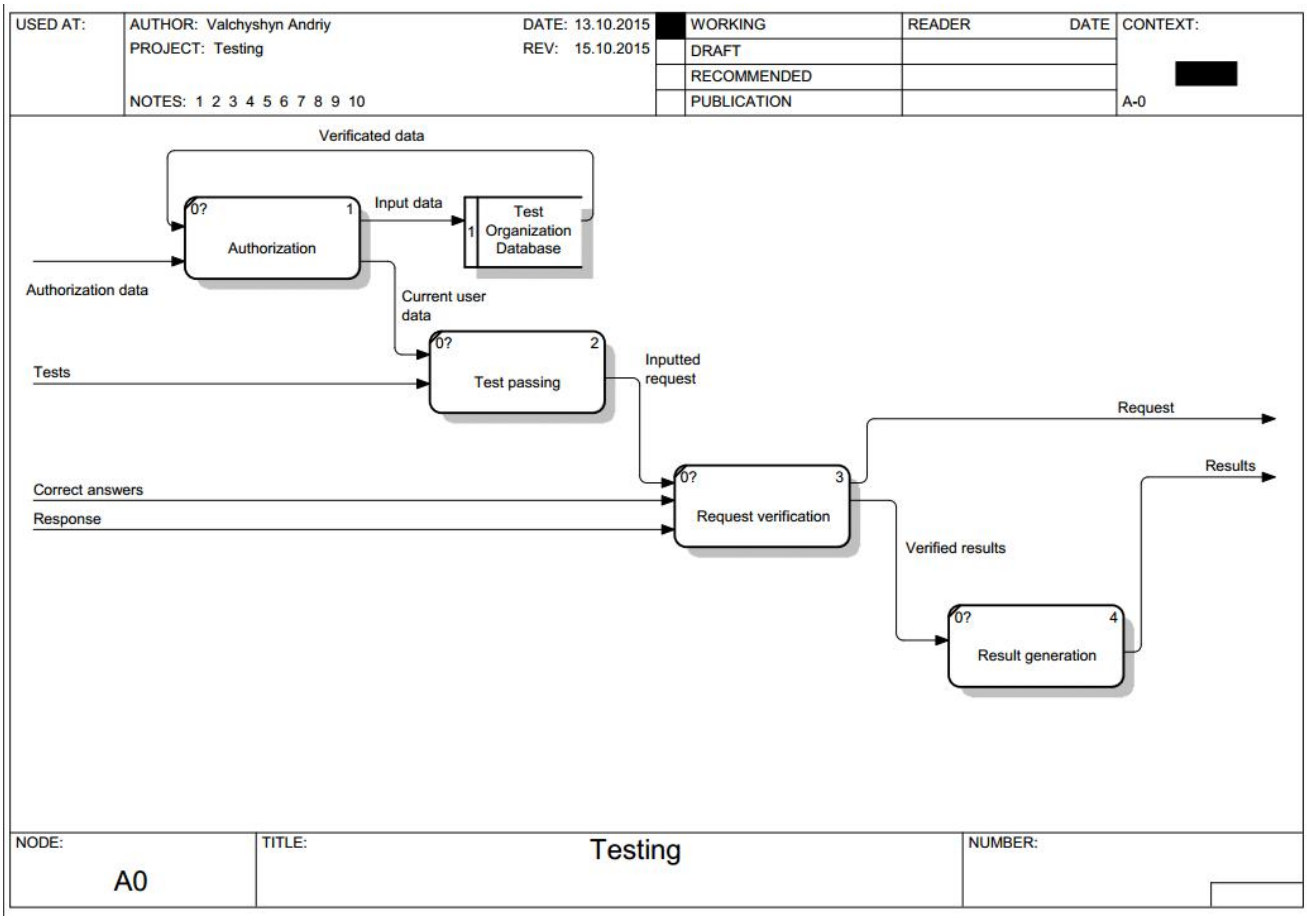

Рисунок 2.6 – Діаграма декомпозиції потоків даних

Як показано на рисунку 2.6 робота, програмної системи здійснюється наступним чином :

1. Входи (input), потоки, які поступають у систему і переробляються нею у вихідні величини, на діаграмі зображені ліворуч:

- ✓ Authorization data (авторизаційні дані);
- ✓ Tests (інформація про завдання);
- ✓ Correct answers (правильні відповіді);
- ✓ Response (відповідь СУБД на запит).

2. Виходи (output), продукти діяльності системи, тобто результати перетворення вхідних величин тощо, на діаграмі зображені праворуч:

- ✓ Result (результат);

- ✓ Request (запит до СУБД);

Процес проходження тестування відбувається за допомогою студента, який взаємодіє з програмною системою. Студент авторизується, отримує список завдань та вибирає одне з них Після цього він має змогу переглянути умову завдання та приступити до його виконання. Після введення відповіді та підтвердження, відповідь-запит надсилається на виконання до СУБД, відповідь на який аналізується та порівнюється з правильними відповідями. Після отримання результату перевірки студент приступає до виконання інших завдань, у разі негативного результату студент може спробувати виконати завдання знову.

Наступним кроком є виявлення основних сутностей та зв’язків між ними. Для цього створено діаграму елементів та зв’язків (див. рис. 4.7).

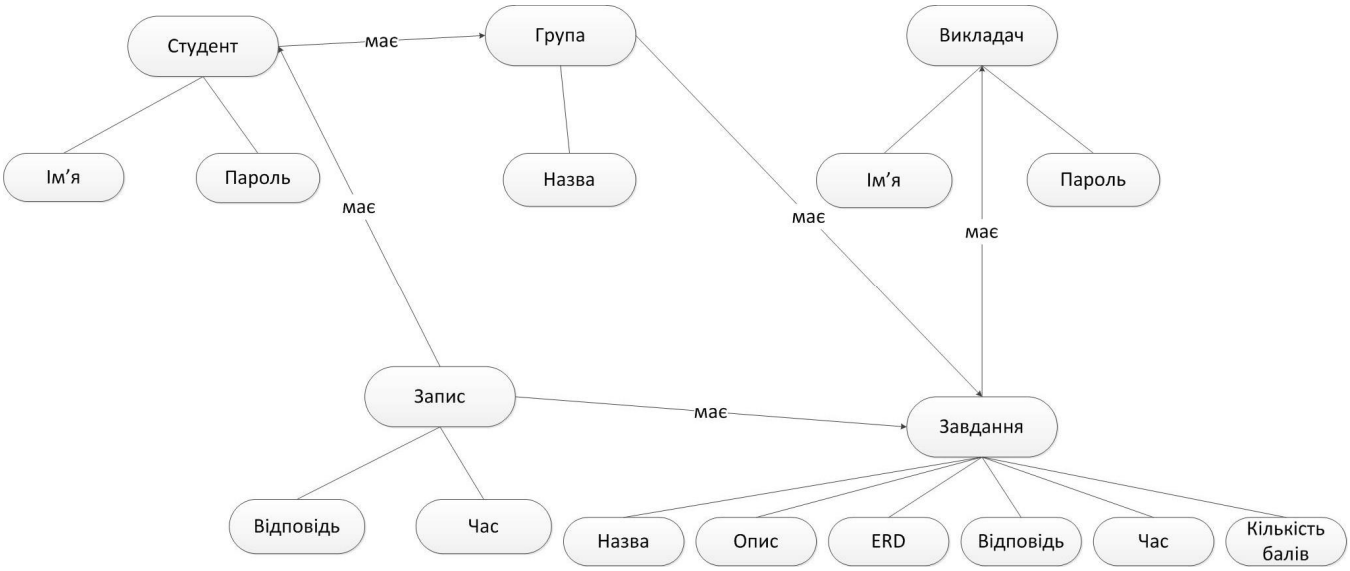

Рисунок 2.7 – Діаграма елементів та зв’язків

Після цього було створено таблицю ідентифікаторів, яку подано у таблиці 2.1.

Таблиця 2.1

Таблиця ідентифікаторів

Об’єкт	Властивість	Тип	Розмірність	Ідентифікатор
Студент				Student
	Ім’я	String	20	Name
	Пароль	String	16	Password
Група				Group
	Назва	String	20	Name
Викладач				Teacher
	Ім’я	String	20	Name
	Пароль	String	16	Password
Запис				Record
	Відповідь	String	50	Answer
	Час	String	10	Time
Завдання				Task
	Назва	String	30	Name
	Опис	String	300	Description
	ERD	String	50	ERD
	Відповідь	String	100	Answer
	Час	String	20	Time
	Значення	Float	20	Value

Наступним кроком є створення діаграми класів UML (див. рис. 2.8).

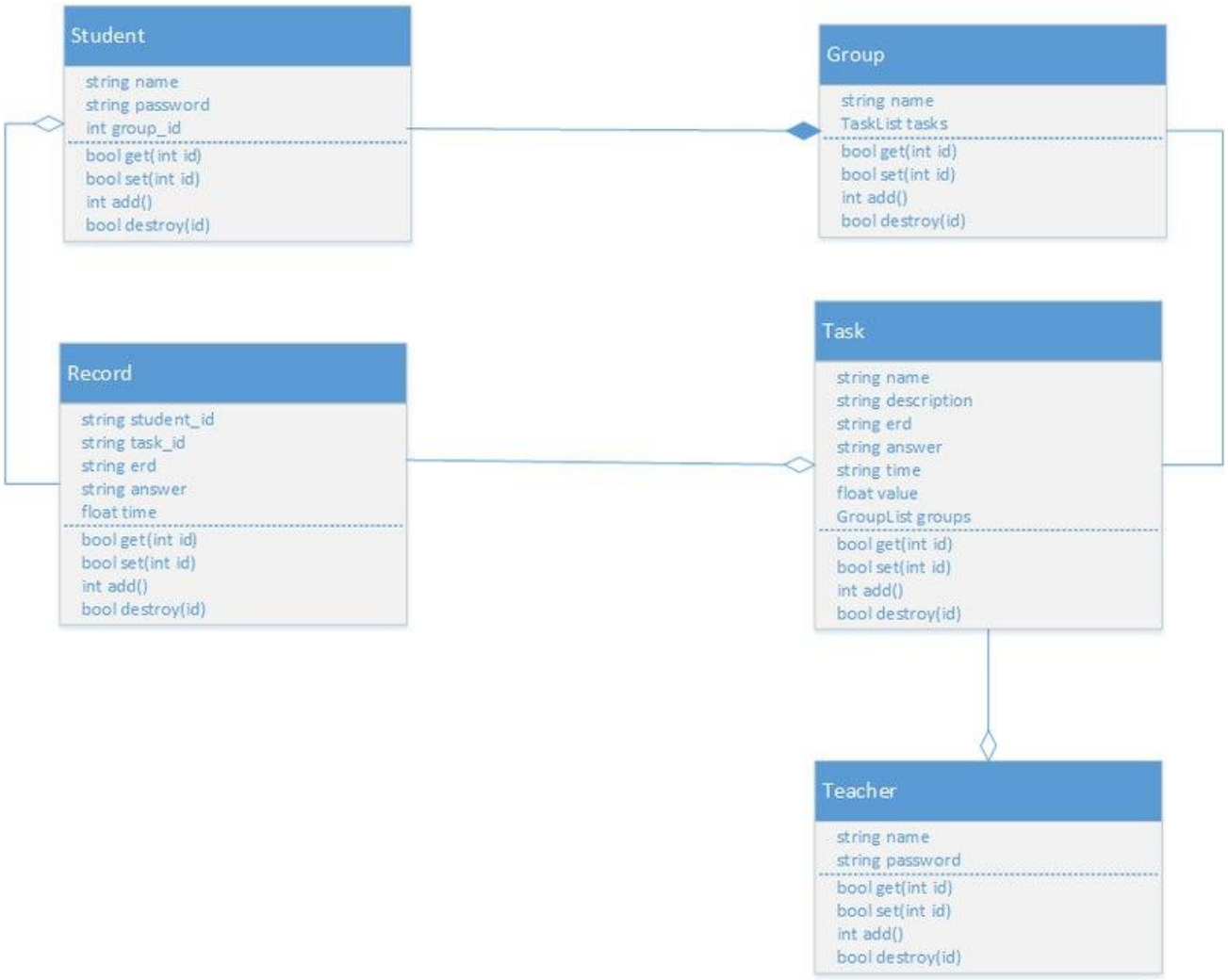

Рисунок 2.8 – Діаграма класів UML

Найбільше поширеним засобом моделювання даних є діаґрами “сутність-взаємозв'язок” (ERD). З їхньою допомогою визначаються важливі для предметної області об'єкти (сутності), їх властивості (атрибути) і відношення
один з одним (зв'язки). ERD безпосередньо використовуються для проектування реляційних баз даних.

Тому на підставі аналізу предметної області у роботі спроектовано структуру бази даних у вигляді ERD діаграми (рисунок 2.9).

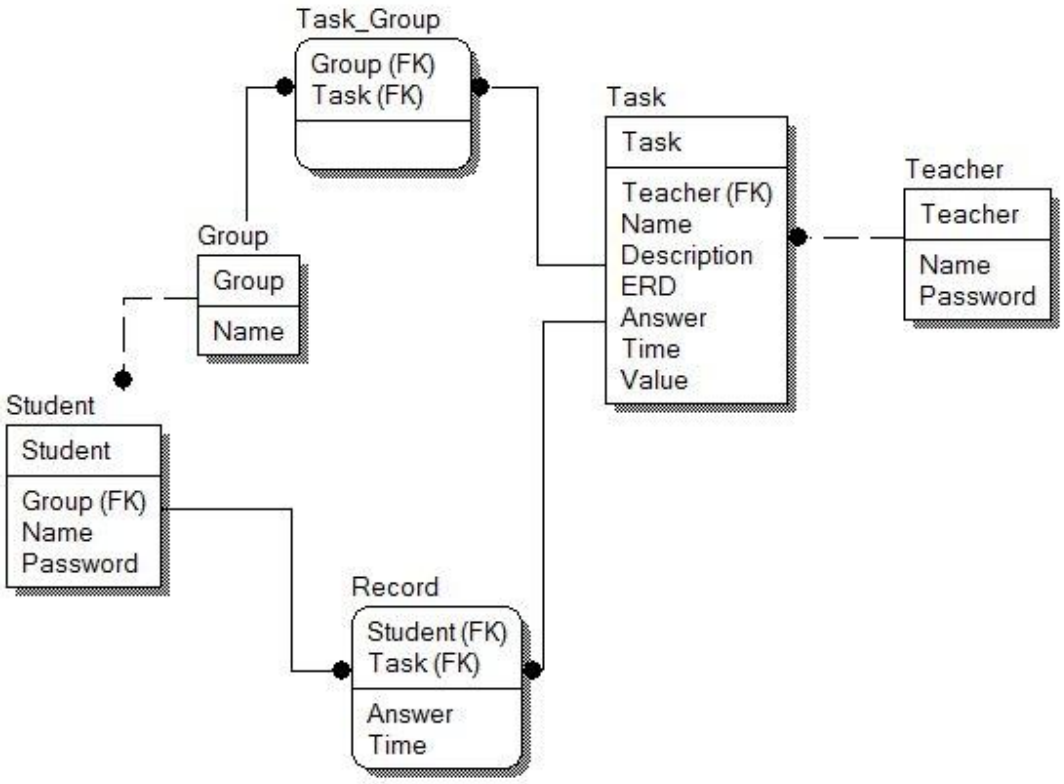

Рисунок 2.9 – ERD-діаграма

Проектування структури бази даних дозволяє визначити та повністю описати всі відношення та поля і є завершальною ланкою перед етапом програмної реалізації бази даних.

Наступним кроком буде планування індексів. Індекси – це спеціальні таблиці, які використовуються пошуковою системою бази даних для швидшого отримання шуканої інформації. Вони використовуються як вказівники на дані в таблицях. Використання індексів пришвидшують виконання запитів пошуку, однак запити додавання на редагування даних виконуються повільніше.

Всі зовнішні ключі являються індексами:

- group_id – зовнішній ключ таблиць Student та Task_Group
- student_id – зовнішній ключ таблиці Record

- task_id – зовнішній ключ таблиць Record і Task_Group
- teacher_id – зовнішній ключ таблиці Task

Також для оптимізації пошуку було вирішено індексувати наступні поля:

- value – поле важливості завдання в таблиці Task
- name – поле імені студента в таблиці Student

Висновок до другого розділу

Спроектовано підсистему “Студент” програмної системи тестування студентів по SQL запитах, проведено аналіз технічної сторони виконання поставленого завдання. Розроблено архітектуру системи, здійснено проектування бази даних.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1. Програмна реалізація проекту

C# є об'єктно-орієнтованою мовою програмування, розробленою компанією Microsoft для створення додатків на платформі Microsoft .NET Framework. Ця мова програмування характеризується строгою типізацією та пріоретизує створення безпечних та надійних додатків. За допомогою C# можна розробити додатків для Windows, веб та розподілених, клієнт-серверних додатків чи та додатків роботи з базами.

Середовищем розробки виступає Microsoft Visual Studio, яке містить просунутий редактор коду, інструменти розробки користуватського інтерфейсу, вбудований відлагоджувач та безліч інших інструментів та функцій для полегшення процесу розробки.

Синтаксис мови C# належить до C-побідних синтаксисів, та загалом схожий на Java і C++. Стиль синтаксису з фігурними дужками, характерний для мови C#, легко впізнають ті, хто знайомий з мовами C, C++ та Java.

C# підтримує всі принципи ООП. Всі основні змінні і методи фрейморку інкапсульовані в класи. Синтаксис C# підтримує такі не притаманні Java функції як типи nullable, делегати, лямда-вирази та прямий доступ до пам’яті. Так як і Java, C# не реалізовує множинне наслідування і використовує як альтернативу інтерфейси. Існує підтримка ітеративних конструкцій, які являються собою колекцій класів та дозволяють визначати власну поведінку, яка може легко використовуватися в коді. Вирази Language-Integrated Query (LINQ) роблять строго типізований запит зручною мовною конструкцією першого класу.

Додатки .Net є компільованими, оскільки для їх запуску код необхідно скомпілювати з мови програмування в проміжну мову CIL, яка є схожою на байт-

код Java. CIL дозволяє додатку бути незалежним від платформи і працює, використовуючи just-in-time компіляцію. Тому програми, що працюють під .NET отримують збільшену швидкість при повторному використанні.

Однією з функцій .Net є garbage collector, «збирач сміття», який керує виділенням і звільненням пам'яті для додатку. Механізм оптимізації збирача сміття визначає найкращий час для виконання очищення, який залежить від інтенсивності використання пам’яті. Подібне очищення передбачає пошук в керованій пам’яті об'єктів, які більше не використовуються додатком і звільнення виділеної для них пам'яті.

Таким чином, беручи до уваги що система оцінювання знань студентів потребує високої безпеки та надійності та враховуючи переваги мови C# та платформи .Net, було прийнято рішення їх використання в процесі розробки додатку.

Система тестування знань студентів складається з двох модулів: модуля “Студент” та модуля “Викладач”, які звязуються між собою через посередництво СУБД. Кожен модуль можна розглядати як додаток з клієнт-серверною архітектурою, оскільки він побудований на основі постійного діалогу з СУБД. Загалом в системі фігурують дві бази даних: база даних для організації тестування та база даних для якої було створено завдання та в діалозі з якої студент буде намагатись вирішити поставлені завдання. Завдання модуля “Студент” полягає в ідентифікації студента, виданню приписаних до його групи завдань для виконання, надання можливості приймати спроби з їх проходження, їх оцінювання та збереження результатів. Даний модуль було розроблено за допомогою мови програмування C# на платформі .Net. Вхідними для модуля даними є список груп, студентів, завдань. Джерелом всіх цих даних є база даних, з метою безпеки паролі студентів та правильні відповіді зберігаються в БД в зашифрованому алгоритмом SHA1 вигляді. Вихідними для модуля даними є статистичні записи з усіма

спробами студентів виконати завдання. Стрічки запитів також подаються в зашифрованому вигляді.

Систему можна розділити на бібліотеки, форми, класи і конфігураційний файл. Конфігураційний файл містить в собі стрічки підключення до баз даних. Кожна форма імплементує певне вікно користувацького інтерфейсу та функціонал який стоїть за ним.

Логіку та функціонал додатку крім форм реалізовують також бібліотеки класів. Загалом модуль містить такі бібліотеки:

1. Бібліотека DataLib — бібліотека для роботи з базами даних, містить в собі класи DB.cs, ManageDB.cs та відповідники сутностей в базі даних. За допомогою класу DB.cs здійснюється підключення до бази даних, в якій буде здійснюватись тестування. Лістинг коду цього класу:

```
namespace DataLib
{
  public class DB
   {
SqlConnection myConnection = new
SqlConnection(ConfigurationManager.ConnectionStrings["AdventureWorks"].ConnectionString);
      SqlCommand comm = new SqlCommand();
      public DataTable Read(String SQL)
      {
         myConnection.Open();
         comm.Connection = myConnection;
         comm.CommandText = SQL;
         using (SqlDataReader reader = comm.ExecuteReader())
```

```
        {
            if (reader.HasRows)
            {
                DataTable dt = new DataTable();
                dt.Load(reader);
                myConnection.Close();
                return dt;
            }
            else
            {
                throw new Exception("NOT SELECT");
            }
        }
    }
  }
}
```

Код класу ManageDB.cs загалом схожий з кодом DB.cs та відрізняється використанням іншої стрічки підключення. Цей клас використовується для підключення до бази даних організації тестування. Лістинг коду цього класу:

```
namespace DataLib
{
  public class ManageDB
  {
```

```
SqlConnection myConnection = new
SqlConnection(ConfigurationManager.ConnectionStrings["testing"].ConnectionString);
    SqlCommand comm = new SqlCommand();
    public DataTable Read(String SQL)
    {
        myConnection.Open();
        comm.Connection = myConnection;
        comm.CommandText = SQL;
        using (SqlDataReader reader = comm.ExecuteReader())
        {
            if (reader.HasRows)
            {
                DataTable dt = new DataTable();
                dt.Load(reader);
                myConnection.Close();
                return dt;
            }
            else
            {
                throw new Exception("NOT SELECT");
            }
        }
    }
}}
```

Клас Student.cs є відповідником таблиці в базі даних та дозволяє здійснювати базові операції додавання, зчитування, редагування та видалення даних з таблиці. Цей клас містить поля відповідні з полями таблиці.

Властивості цього класу:

string name — поле імені студента

string password — поле зашифрованого пароля студента

int group_id — поле id групи до якої належить студент

Основні методи цього класу:

bool get(int id) — метод зчитування даних з таблиці в клас за заданим id

bool set(int id) — метод запису даних в таблицю за заданим id

int add() — метод додавання нового запису в таблиці з даними цього класу, повертає id запису

bool destroy(id) — метод видалення таблиці за заданим id запису

Клас Teacher.cs є відповідником таблиці в базі даних та дозволяє здійснювати базові операції додавання, зчитування, редагування та видалення даних з таблиці. Цей клас містить поля відповідні з полями таблиці.

Властивості цього класу:

string name — поле імені викладача

string password — поле зашифрованого пароля викладача

Основні методи цього класу:

bool get(int id) — метод зчитування даних з таблиці в клас за заданим id

bool set(int id) — метод запису даних в таблицю за заданим id

int add() — метод додавання нового запису в таблиці з даними цього класу, повертає id запису

bool destroy(id) — метод видалення таблиці за заданим id запису

Клас Group.cs є відповідником таблиці в базі даних та дозволяє здійснювати базові операції додавання, зчитування, редагування та видалення даних з таблиці. Цей клас містить поля відповідні з полями таблиці.

Властивості цього класу:

string name — поле назви групи

TaskList tasks — список завдань, приписаних до групи

Основні методи цього класу:

bool get(int id) — метод зчитування даних з таблиці в клас за заданим id

bool set(int id) — метод запису даних в таблицю за заданим id

int add() — метод додавання нового запису в таблиці з даними цього класу, повертає id запису

bool destroy(id) — метод видалення таблиці за заданим id запису

Клас Task.cs є відповідником таблиці в базі даних та дозволяє здійснювати базові операції додавання, зчитування, редагування та видалення даних з таблиці. Цей клас містить поля відповідні з полями таблиці.

Властивості цього класу:

string name — поле назви завдання

string description — поле опису завдання

string erd — поле шляху до зображення erd

string answer — зашифроване поле результату правильного запиту як відповіді на завдання

string time — значення часу, назначеного на виконання задвання

float value — значення кількості балів оцінки виконання завдання

GroupList groups — список груп, яким приписане це завдання

Основні методи цього класу:

bool get(int id) — метод зчитування даних з таблиці в клас за заданим id

bool set(int id) — метод запису даних в таблицю за заданим id

int add() — метод додавання нового запису в таблиці з даними цього класу, повертає id запису

bool destroy(id) — метод видалення таблиці за заданим id запису

Клас Record.cs є відповідником таблиці в базі даних та дозволяє здійснювати базові операції додавання, зчитування, редагування та видалення даних з таблиці. Цей клас містить поля відповідні з полями таблиці.

Властивості цього класу:

string student_id — поле id студента

string task_id — поле id завдання

string erd — поле шляху до зображення erd

float time — поле часу, витраченого на виконання завдання

string answer — зашифрована стрічка запиту

Основні методи цього класу:

bool get(int id) — метод зчитування даних з таблиці в клас за заданим id

bool set(int id) — метод запису даних в таблицю за заданим id

int add() — метод додавання нового запису в таблиці з даними цього класу, повертає id запису

bool destroy(id) — метод видалення таблиці за заданим id запису

2. Бібліотека Hash — бібліотека, яка відповідає за хешування даних алгоритмом SHA1. Ця бібліотека містить один клас HashSum.cs. Лістинг цього класу наведено нижче:

```
public class HashSum
  {
    private DataTable dt;
    public HashSum(DataTable dt, String secret = "PZS")
    {
      if (dt.TableName == "")
```

```
    {
        dt.TableName = secret;
    }
    this.dt = dt;
}
public HashSum(SqlDataReader reader, String secret = "PZS")
{
    DataTable dt = new DataTable();
    dt.Load(reader);
    dt.TableName = secret;
    this.dt = dt;
}
private byte[] SerializeData()
{
    // Serialize the table
    DataContractSerializer serializer = new DataContractSerializer(typeof(DataTable));
    MemoryStream memoryStream = new MemoryStream();
    XmlWriter writer = XmlDictionaryWriter.CreateBinaryWriter(memoryStream);
    serializer.WriteObject(memoryStream, this.dt);
    byte[] serializedData = memoryStream.ToArray();
    return serializedData;
}
private byte[] CalculateHashValue(byte[] SerializedData)
```

```
{
    // Calculte the serialized data's hash value
    SHA1CryptoServiceProvider SHA = new SHA1CryptoServiceProvider();
    byte[] hash = SHA.ComputeHash(SerializedData);
    return hash;
}
private String GetHash(byte[] hash)
{
    return Convert.ToBase64String(hash);
}
public String GetHashString()
{
    byte[] serializedData = this.SerializeData();
    byte[] hashValue = this.CalculateHashValue(serializedData);
    return this.GetHash(hashValue);
}
```

На основі розробленого додатка були сформовані системні вимоги до комп’ютерів клієнта та сервера.

Мінімальні вимоги до сервера :

Комп'ютер з процесором x86 з тактовою частотою 1,0 ГГц і вище , 1 Гб оперативної пам'яті, 15 Гб жорсткої пам'яті.

Рекомендовані вимоги до сервера :

2 комп'ютера з процесором з тактовою частотою 2,0 ГГц і вище , 4 Гб оперативної пам'яті, 30 Гб жорсткої пам'яті.

Мінімальні вимоги до клієнта :

Комп'ютер з процесором x86 з тактовою частотою 1,0 ГГц і вище , 512 Мб оперативної пам'яті, 3 Гб жорсткої пам'яті.

Рекомендовані вимоги до клієнта:

Комп'ютер з процесором x86 з тактовою частотою 2,0 ГГц і вище , 1 Гб оперативної пам'яті, 5 Гб жорсткої пам'яті.

Для роботи системи необхідне наступне встановлене програмне забезпечення: Microsoft .Net Framework 4, MS SQL Server 2014, Microsoft Visual C++ Redistributable x86. Для роботи з клієнтською частиною системи потрібний комп'ютер , підключений по протоколу TCP / IP до мережі , в якій знаходиться сервер.

3.2. Програмна реалізація проекту

Засобом для створення таблиць та подальшої реалізації бази даних було обрано Microsoft SQL Server Management Studio 2014. Цей програмний засіб по замовчуванню постачається разом з СУБД Microsoft SQL Server 2014 та є рекомендованим Microsoft ПЗ для взаємодії з нею. Вибір середовищ Microsoft Visual Studio, Microsoft та SQL Server Management Studio дозволяє підтримувати високий рівень підтримки та сумісності розробки БД та програмних систем оскільки всі програмні засоби були розроблені однією компанією з використанням спільних стандартів.

SQL Server Management Studio являє собою утиліту, яка входить в пакет Microsoft SQL Server та використовується як середовище розробки і адміністрування баз даних та налаштування всіх компонентів пакету. Цей програмний засіб включає в себе графічний оболонку для управління об’єктами і налаштуваннями сервера та редактор T-SQL скриптів. Важливою компонентою

системи є Object Explorer, який дозволяє переглядати, редагувати та керувати об’єктами та їх ієрархіями.

Microsoft SQL Server може використовуватись як локальна чи серверна СУБД для роботи з віддаленими користувачами та підходить роботи як з низьким так і великим навантаженням. Розгорнута система підтримки, єдина довідкова система MSDN та прикладів баз даних дають змогу проводити ефективно усувати можливі проблеми та вивчати новий функціонал.

Серед недоліків слід виділити платний спосіб розповсюдження ПЗ та обмежені можливості інтеграції з засобами конкуруючих компаній та технологій.

Таким чином, зваживши всі переваги та недоліки в якості СУБД було обрано Microsoft SQL Server 2014 та SQL Server Management Studio 2014 в якості середовища реалізації БД.

Програмна система тестування знань студентів з SQL запитів передбачає розгортання двох баз даних: бази даних, в якій зберігатимуться дані для проведення тестування та база даних, яка буде використовуватись для виконання завдання студентом. В той час як перша містить чітко визначену структуру та таблиці, друга база даних не є конкретизованою та обирається на розсуд викладача. Єдиним обмеженням слід вважати необхідність створення завдання та виконання його студентом в одній і тій ж базі даних. Тому мною в якості такої бази даних було використано стандартний приклад AdventureWorksDW2014, який надається компанією Microsoft для вивчення функціоналу та в якості тестових даних. Він містить безліч таблиць та записів які ідеально підходять для проведення тестування оскільки дозволяю створити безліч різних завдань, які не будуть пересікатись.

Отже, в базі даних організації тестування повинні бути реалізовані таблиці Group, Student, Teacher, Task, Record, Task_Group.

Спочатку створим нову базу даних. Для цього в засобі SQL Server Management Studio викличемо відповідну функцію використовуючи Object

Exporer, натиснувши праву клавішу на підрозділі Databases поточного зв’язку з SQL Server та обравши варіант ‘New Database’. Ця послідовність зображена на рисунку 3.1. Ця фунція дозволяє створювати нові бази даних для подальшої роботи з ними. Інші варіанти можуть передбачати імпорт/експорт та відновлення баз даних із файлів чи інших джерел.

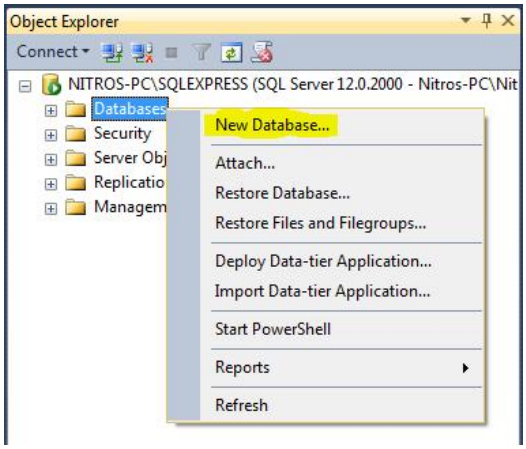

Рисунок 3.1 – Створення нової бази даних

Результат приведе до виклику вікна конфігурації об’єкту нової бази даних як на рисунку 3.2. Необхідно ввести назву ‘testing’ та підтвердити виконання.

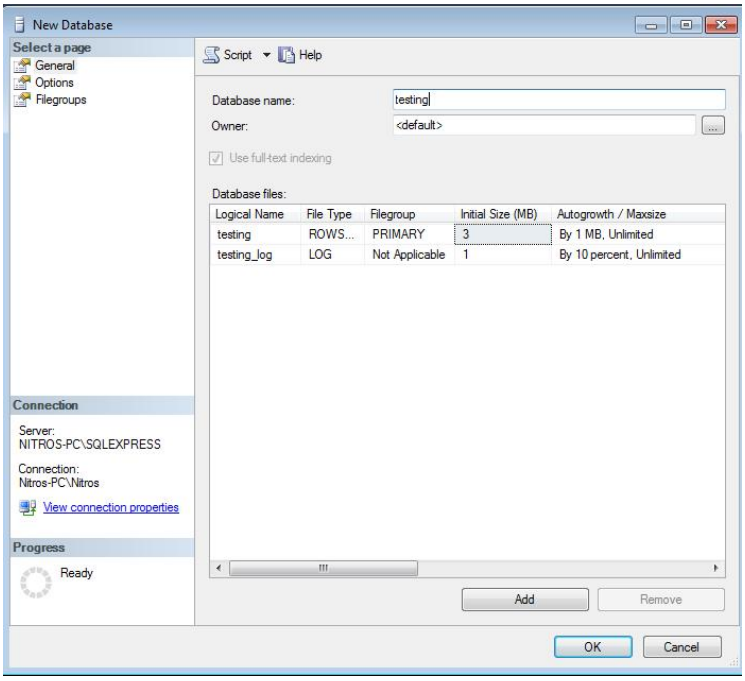

Рисунок 3.2 – Вікно налаштування нового об’єкту бази даних

В результаті буде отримано та виконано такий SQL-скрипт:

```
USE [master]
GO
CREATE DATABASE [testing]
CONTAINMENT = NONE
ON  PRIMARY
GO
USE [testing]
GO
```

Виконання цього скрипта призведе до створення нової бази даних Testing та дасть змогу використовувати наступні скрипти уже в її масштабах.

Наступним кроком буде створення таблиці Group. Для цього викличем редактор скриптів за допомогою комбінацій клавіш Ctrl+N. На рисунку 3.3 зображено вікно редактора скриптів.

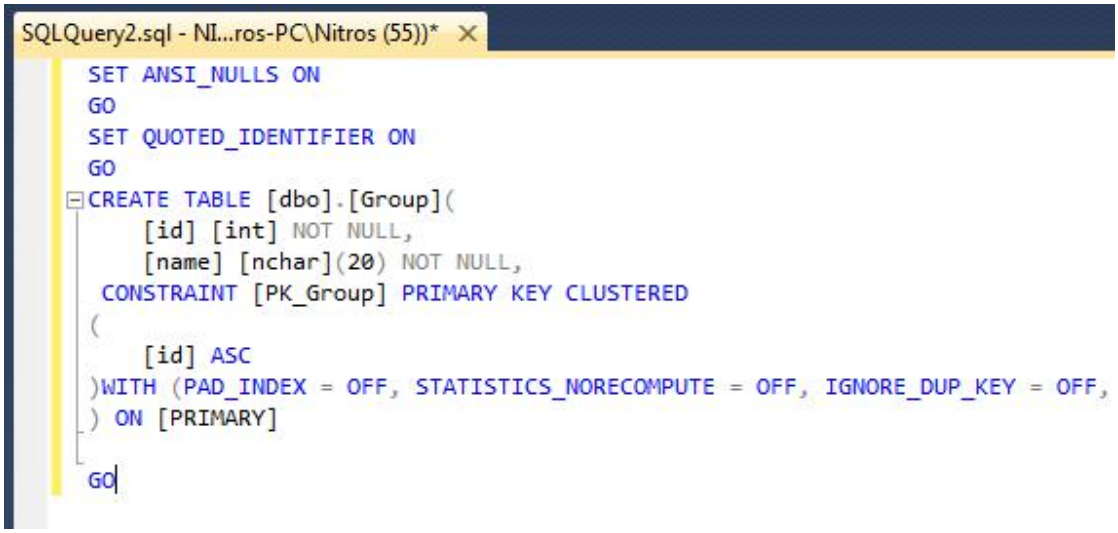

Рисунок 3.3 – Вікно редактора скриптів

Введемо та виконаєм наступний SQL-скрипт:

```
SET ANSI_NULLS ON
GO
```

```
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Group](
	[id] [int] NOT NULL,
	[name] [nchar](20) NOT NULL,
 CONSTRAINT [PK_Group] PRIMARY KEY CLUSTERED
([id] ASC) WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Виконання цього скрипта призведе до створення таблиці Group з первинним ключем id та з полями і типами цієї сутності згідно з таблицею ідентифікаторів та ERD.

Наступним буде створення таблиці Student. Буде використано такий SQL-запит:

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Student](
	[id] [int] NOT NULL,
	[name] [nchar](20) NOT NULL,
	[password] [nchar](16) NOT NULL,
	[group_id] [int] NOT NULL,
 CONSTRAINT [PK_Student] PRIMARY KEY CLUSTERED
([id] ASC)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Виконання цього скрипта призведе до створення таблиці Student з первинним ключем id та з полями і типами цієї сутності згідно з таблицею ідентифікаторів та ERD.

Тепер буде створено таблицю Teacher. Ця таблиця буде використовуватись в іншому модулі програмної системи тестування знань студентів, однак оскільки база даних є спільною створення такої таблиці є обов’язковим. Буде використано такий SQL-скрипт:

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Teacher](
	[id] [int] NOT NULL,
	[name] [nchar](20) NOT NULL,
	[password] [nchar](16) NOT NULL,
 CONSTRAINT [PK_Teacher] PRIMARY KEY CLUSTERED
([id] ASC)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Виконання цього скрипта призведе до створення таблиці Teacher з первинним ключем id та з полями і типами цієї сутності згідно з таблицею ідентифікаторів та ERD.

Далі створим таблицю Record. Буде використано такий SQL-скрипт:

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Record](
	[id] [int] NOT NULL,
	[asnwer] [nchar](50) NULL,
	[time] [nchar](10) NULL,
	[student_id] [int] NOT NULL,
	[task_id] [int] NOT NULL,
 CONSTRAINT [PK_Record] PRIMARY KEY CLUSTERED
```

```
([id] ASC)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Виконання цього скрипта призведе до створення таблиці Record з первинним ключем id та з полями і типами цієї сутності згідно з таблицею ідентифікаторів та ERD.

Далі створим таблицю Task_Group. Вона буде виступати в якості проміжної в зв’язку many-to-many між таблицями Task та Group. Буде використано такий SQL-скрипт:

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Task_Group](
        [id] [int] NOT NULL,
        [group_id] [int] NOT NULL,
        [task_id] [int] NOT NULL,
 CONSTRAINT [PK_Group_Task] PRIMARY KEY CLUSTERED
([id] ASC)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Виконання цього скрипта призведе до створення таблиці Task_Group з первинним ключем id та з полями і типами цієї сутності згідно з таблицею ідентифікаторів та ERD.

Наступним кроком буде створення зв’язків між таблицями. Виконаєм таку послідовність SQL команд:

```
ALTER TABLE [dbo].[Record] WITH CHECK ADD CONSTRAINT [FK_Record_Student] FOREIGN
KEY([student_id]) REFERENCES [dbo].[Student] ([id]) GO
ALTER TABLE [dbo].[Record] CHECK CONSTRAINT [FK_Record_Student] GO
```

Цей скрипт виконується відносно таблиці Record та створює зв'язок зовнішнього ключа student_id з первинним ключем id таблиці Student. Це дозволить зв’язати ці дві таблиці між собою в відношенні one-to-many.

```
ALTER TABLE [dbo].[Record] WITH CHECK ADD CONSTRAINT [FK_Record_Task] FOREIGN KEY([task_id]) REFERENCES [dbo].[Task] ([id]) GO
ALTER TABLE [dbo].[Record] CHECK CONSTRAINT [FK_Record_Task] GO
```

Цей скрипт виконується відносно таблиці Record та створює зв'язок зовнішнього ключа task_id з первинним ключем id таблиці Task. Це дозволить зв’язати ці дві таблиці між собою в відношенні one-to-many.

```
ALTER TABLE [dbo].[Student] WITH CHECK ADD CONSTRAINT [FK_Student_Group] FOREIGN KEY([group_id]) REFERENCES [dbo].[Group] ([id]) GO
ALTER TABLE [dbo].[Student] CHECK CONSTRAINT [FK_Student_Group] GO
```

Цей скрипт виконується відносно таблиці Student та створює зв'язок зовнішнього ключа group_id з первинним ключем id таблиці Group. Це дозволить зв’язати ці дві таблиці між собою в відношенні one-to-many.

```
ALTER TABLE [dbo].[Task_Group] WITH CHECK ADD CONSTRAINT [FK_Task_Group_Group] FOREIGN KEY([group_id]) REFERENCES [dbo].[Group] ([id]) GO
ALTER TABLE [dbo].[Task_Group] CHECK CONSTRAINT [FK_Task_Group_Group] GO
ALTER TABLE [dbo].[Task_Group] WITH CHECK ADD CONSTRAINT [FK_Task_Group_Task] FOREIGN KEY([task_id]) REFERENCES [dbo].[Task] ([id]) GO
ALTER TABLE [dbo].[Task_Group] CHECK CONSTRAINT [FK_Task_Group_Task] GO
```

Цей скрипт виконується відносно таблиці Task_Group, яка є проміжною в звязку many-to-many між таблицями Task та Group. Цей зв'язок забезпечується створенням двох зв’язків one-to-one за допомогою зовнішніх ключів task_id та group_id з первинними ключами id таблиць Task та Group відповідно.

Висновок до третього розділу

Обгрунтовано вибір мови програмування та додаткових програмних засобів, описно вимоги до програмного продукту, описана програмна реалізація системи з приведеними скриптами.

РОЗДІЛ 4. ТЕСТУВАННЯ ТА ДОСЛІДНА ЕКСПЛУАТАЦІЯ

4.1. Тестування

З метою перевірки програмний продукт на відповідність вимогам та відсутність помилок було проведено тестування. Врахувавши специфіку розроблюваного продукту було складено та проведено наступні види тестування:

- Функціональне тестування;
- GUI тестування;

Функціональне тестування

Функціональне тестування дозволяє провести перевірку на відповідність продукту функціональним вимогам. Тести проводяться над усіма функціями програмної системи та Функціональне тести охоплюють всі розроблені функції, та розроблені з акцентом на тестування найбільш ймовірних типів помилки в програмній системі.

Для проведення функціонального тестування було розроблено тестові випадку у вигляді таблиці, що містить таку інформацію:

- № (номер тестового випадку);
- Description (опис об’єкту тестування);
- Steps to reproduce (кроки відтворення тестового випадку);
- Expected result (очікуваний результат тестування);
- Actual result (реальний результат тестування);
- Passed/Failed (вказується, чи пройдений тест, чи ні.);
- Environment (середовище, в якому здійснювалося тестування).

Для тестування графічного інтерфейсу користувача проводиться GUI тестування. Цей вид тестування має на меті перевірити відповідність роботи інтерфейсу користувача вимогам. Дії потенційного користувача відтворюються для перевірки очікуваного результату з реальним. Для цього було складено чек-ліст та проведено ручне тестування інтерфейсу. Так як програмною системою водночас буде користуватися дуже мала кількість користувачів, автоматизоване тестування в даному випадку є затратним та зайвим.

Під час фази розробки функціональних тестів було спроектовано 9 функціональних тестових випадків. Таблиця 4.1 показує розподіл функціональних тестових випадків і наборів тестових даних для цих випадків за варіантами використання.

Таблиця 4.1

Розподіл за варіантами використання

Варіанти використання	**Тестові випадки**	**Тестові дані**
View result	1	8
Authorization	1	5
Task info view	1	2
Task execution	3	47
Task list view	3	9
Загалом	9	61

Таблиця 4.2 містить деталі функціональних тестових випадків.

Таблиця 4.2

Специфікація тестів для функціонального тестування.

	Тестові випадки	**Тестові дані**

1	Verify the authorization	5
2	Verify task list	2

Продовження таблиці 4.2

3	Verify task selection	4
4	Verify test info view	2
5	Verify query execution	19
6	Verify query error view	11
7	Verify task completion	7
8	Verify grade calculation	8
9	Verify student info	3
	Загалом	61

Було розроблено 12 GUI тестових випадків. Таблиця 4.3 показує розподіл GUI тестових випадків і наборів тестових даних за варіантами використання.

Таблиця 4.3

Розподіл за варіантами використання

Варіанти використання	**Тестові випадки**	**Тестові дані**
View result	1	3
Authorization	2	6
Task info view	3	12
Task execution	7	16

Task list view	6	19
Загалом	19	56

Таблиця 4.4 містить деталі GUI тестових випадків.

Таблиця 4.4

Специфікація тестів для GUI тестування.

	Тестові випадки	**Тестові дані**
1	Check -> “Authorization drop-down”	2
2	Check -> “Task list”	2
3	Check -> “Task row”	3
4	Check -> “View task button”	3
5	Check -> “SQL query field”	5
6	Check -> “Execution button”	3
7	Check -> “Query result field”	3
8	Check -> “Error text”	2
9	Check -> “Grade text”	3
10	Check -> “Student info text”	2
11	Check -> “Navigation buttons”	5
12	Check -> “Window resizing”	4
	Загалом	28

9 тестів з 9 функціональних тестів, наведених в таблиці 4.2 пройшли успішно. Отже, усі програмна система відповідає функціональним вимогам.

11 тестів із 12 GUI тестових випадків, наведених в таблиці 4.4 пройшли успішно. Отже, GUI тестування можна розшлядати як частково успішно – 0,92% тестових випадків пройшли успішно.

Відомі дефекти

Дефект №1

Під час зміни розміру вікна, поле результату виконання запиту не міняє своїх розмірів.

Подолання дефекту:

Розмір поля відображення результатів запиту було змінено з врахуванням зміни розміру вікна і тепер також змінює свій розмір

4.2. Розгортання програмного продукту

Для роботи програмної системи необхідне таке апаратне та програмне забезпечення:

Мінімальні вимоги до сервера :

Комп'ютер з процесором x86 з тактовою частотою 1,0 ГГц і вище , 1 Гб оперативної пам'яті, 15 Гб жорсткої пам'яті.

Рекомендовані вимоги до сервера :

2 комп'ютера з процесором з тактовою частотою 2,0 ГГц і вище , 4 Гб оперативної пам'яті, 30 Гб жорсткої пам'яті.

Мінімальні вимоги до клієнта :

Комп'ютер з процесором x86 з тактовою частотою 1,0 ГГц і вище , 512 Мб оперативної пам'яті, 3 Гб жорсткої пам'яті.

Рекомендовані вимоги до клієнта:

Комп'ютер з процесором x86 з тактовою частотою 2,0 ГГц і вище , 1 Гб оперативної пам'яті, 5 Гб жорсткої пам'яті.

Для роботи системи необхідне наступне встановлене програмне забезпечення: Microsoft .Net Framework 4, MS SQL Server 2014, Microsoft Visual C++ Redistributable x86. Всі ці продукти можна завантажити на офіційному сайті Microsoft. Для роботи з клієнтською частиною системи потрібний комп'ютер , підключений по протоколу TCP / IP до мережі , в якій знаходиться сервер.

4.3. Інструкція користувача

4.3.1. Компоненти ПЗ

Програмна система розроблена на мові програмування C# у середовищі Microsoft Visual Studio 2013 на платформі .NET і експлуатується під управлінням ОС Windows.

Для функціонування системи тестування необхідне встановлення на СУБД сервера двох баз даних, файли яких постачаються разом з програмною системою.

Набір файлів модуля «Студент» системи тестування знань студентів представлений в таблиці 4.5.

Таблиця 4.5

Набір файлів, необхідних для функціонування модуля «Студент»

№	Файл	Призначення	Належить проекту
1	Hash.dll	Бібліотека для хешування паролів та запитів	Бібліотеки програмної системи тестування знань студентів
2	DataLib.dll	Бібліотека для з’єднання з базами даних та діалогу з ними	
3	SQLtesting.exe	Основний виконуваний файл додатку	Модуль «Студент»

		модуля «Студент»	програмної системи тестування знань студентів
4	SQLtesting.config	Конфігураційний файл	

4.3.2. Встановлення ПЗ

1. Встановити Microsoft .NET Framework 4, завантаживши його з офіційного сайту Microsoft.

2. Встановити Microsoft SQL Server, завантаживши його з офіційного сайту Microsoft.

3. На стороні сервера, використовуючи СУБД Microsoft SQL Server 2014 розгорнути 2 бази даних: Testing та AdventureWorksDW2014. Для цього використати середовище Microsoft SQL Server Management Studio 2014, в якому після підключення до СУБД необхідно обрати імпорт бази даних в панелі інструментів. Джерелом даних слід вказати файли баз даних, які надаються разом з програмною системою.

4. Бібліотеки та модуль «Студент» програмної системи тестування знань студентів розмістити в одній папці та використовувати запускаючи виконуваний файл SQLtesting.exe.

4.3.3. Налаштування ПЗ

Відповідно до створених баз даних, користувачів та налаштувань з’єднання, необхідно записати стрічки підключення до двох баз даних в файл конфігурації SQLtesting.config в стрічки параметри ‘Testing’ та ‘AdventureWorks’.

4.3.4. Базові функції ПЗ

Для авторизації та перегляду списку завдань виберіть групу та ім'я студента зі списку, введіть пароль в поле “Password” і нажміть “Login”.

Для вибору конкретного завдання зі списку завдань двічі натисніть на рядок відповідного завдання в таблиці.

Для перегляду умови завдання натисніть кнопку “View task”.

Для повернення до попереднього меню натисніть кнопку “ ↩ ”.

Для виконання введення відповіді на завдання введіть запит в поле “Your SQL query” та натисніть на кнопку “Execute”.

Для завершення роботи з програмою натисніть стандартну кнопку вікна “Закрити”.

4.3.5. Аналіз помилок

При виникненні помилок “Server connection timed out” та “Cannot connect to server” необхідно провірити з’єднання клієнта та сервера з мережею, активність СУБД сервера.

4.3.6. Організація інтерфейсу з користувачем

Організація взаємодії з користувачем базується на графічному інтерфейсі, представленому формами додатку та передбачає собою прямий діалог користувача з системою.

Графічний інтерфейс є одним з типів інтерфейсів, в якому взаємодія з користувачем базується на використанні графічних зображень та елементів. Цей тип інтерфейсу є найпулярнішим серед більшості додатків, оскільки є найбільш зручним та зрозумілим для користувача.

Діалог типу регламентованого діалогу є першим в історії способом взаємодії користувача і комп’ютера. При цьому виді діалогу запити користувача і відповіді системи мають наперед задану структуру і кодуються з використанням лінгвістичного забезпечення IT (IC). Залежно від ситуації ініціатива ведення діалогу може належати комп'ютеру або користувачу.

Загалом форм в модулі “Студент” є чотири: UserLogin для вікна авторизації, TaskMenu для вікна вибору задвань, TaskExec для виконання завдання та TaskView для перегляду умов завдання.

На рисунку 4.1 представлено вікно авторизації користувача, в даному вікні користувач може здійснити вхід у систему, обраши свою групу та ім’я в випадаючих списках “Stundent” та “Group”, ввівши персональний пароль в поле “Password”. Виконання входу здійснюється натисканям кнопки “Login”. Також є можливість завершення роботи програми при натисканні стандартної кнопки вікна “Закрити”.

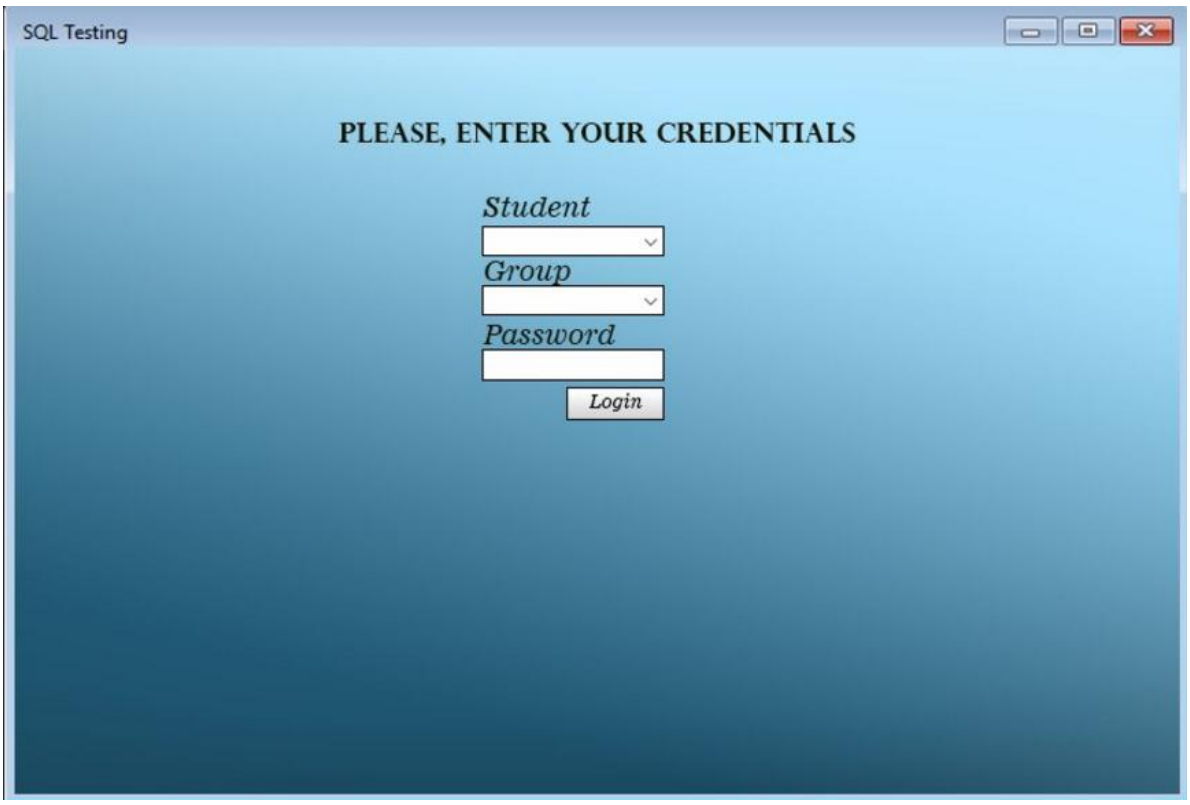

Рисунок 4.1 – Вікно авторизації

На рисунку 4.2 зображено вікно вибору завдань. У цьому вікні користувач бачить список завдань, представлений у таблиці короткою інформацією про завдання. Також вікно містить інформацію про користувача та поточну оцінку. Можливість завершення роботи з програмою передбачена натисканням стандартної кнопки вікна “Закрити”.

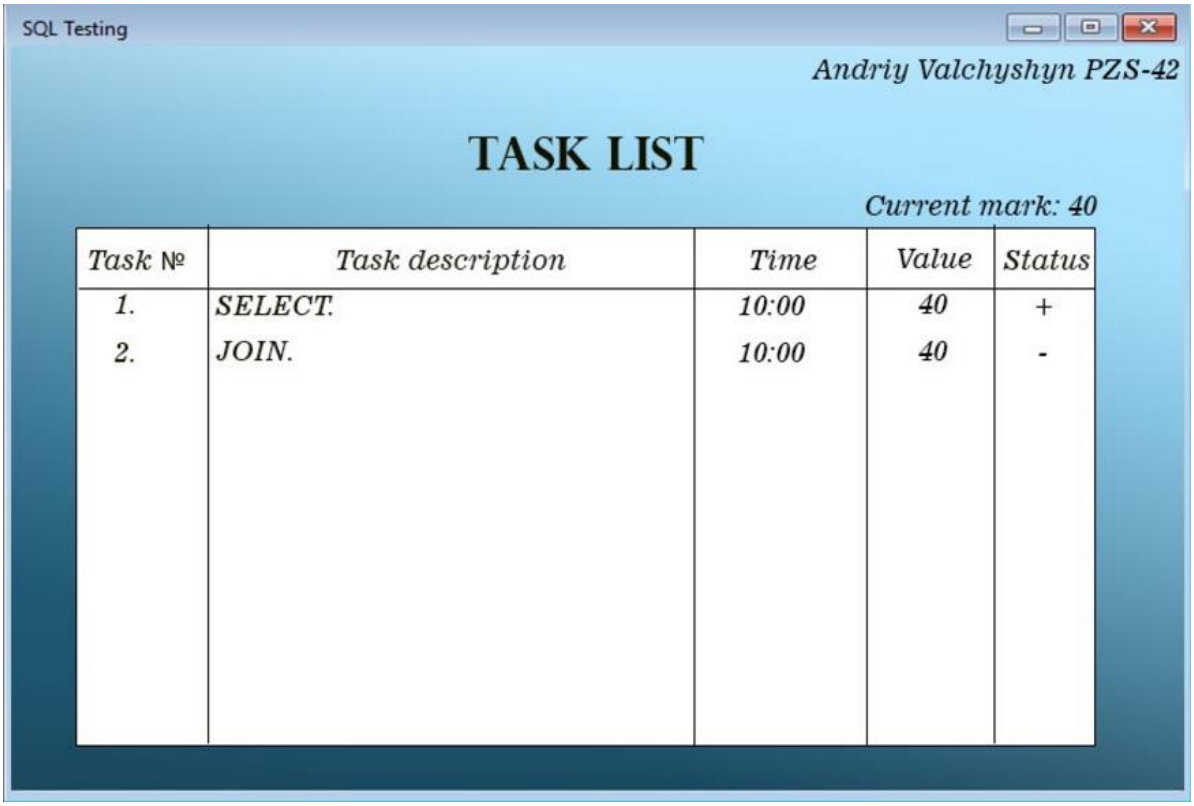

Рисунок 4.2 – Вікно вибору завдань

На рисунку 4.3 зображено вікно виконання завдання. У цьому вікні користувач має змогу ввести запит в поле “Your SQL query” як відповідь на завдання. Підтвердження виконання запиту здійснює кнопка “Execute”. Результат виконання такого запиту відображається в полі “Result”. Кнопка “View task” передбачає відкриття вікна перегляду умови завдання.

Користувач може повернутися до попереднього меню натиснувши кнопку “ ⬑ ”. Можливість завершення роботи з програмою передбачена натисканням стандартної кнопки вікна “Закрити”.

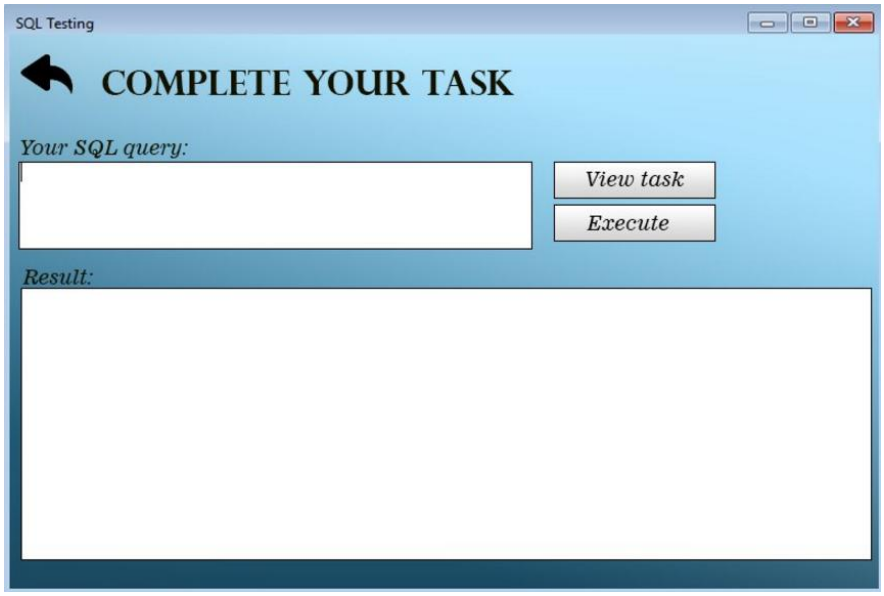

Рисунок 4.3 – Вікно виконання завдання

На рисунку 3.4 зображено вікно перегляду умови завдання. У цьому вікні користувач має змогу побачити опис завдання в полі “Description” та зображення ERD в полі “ERD”. Користувач може повернутися до попереднього меню натиснувши кнопку “ ⬑ ”. Можливість завершення роботи з програмою передбачена натисканням стандартної кнопки вікна “Закрити”.

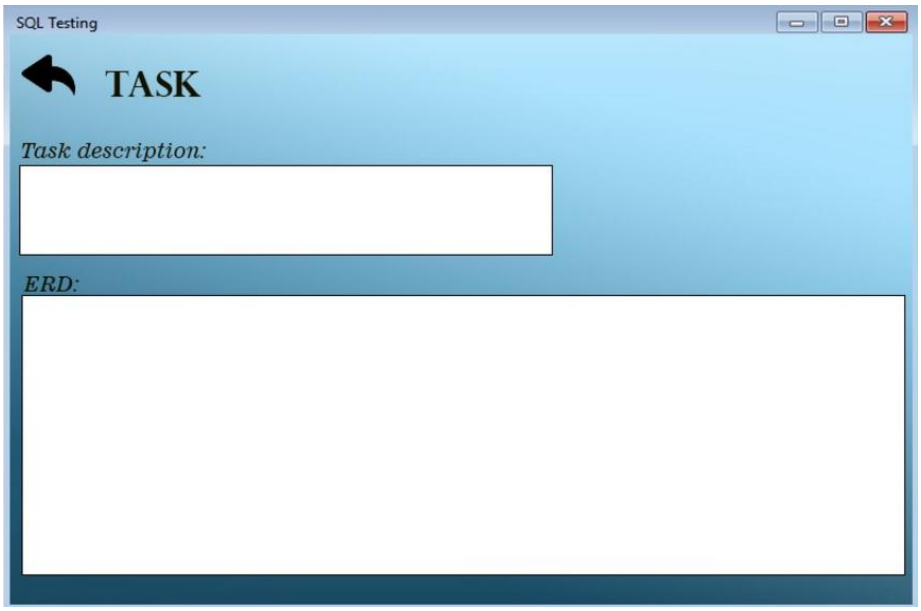

Рисунок 4.4 – Вікно перегляду умови завдання

Висновок до четвертого розділу

Проведено аналіз програмного продукту та вибрано необхідні методи тестування. Розроблено тест-кейси та чек-лісти, згідно з якими проводилось тестування програмної системи. Описано необхідне програмне та апаратне забезпечення, необхідне для розгортання програмного продукту, приведені назви необхідних файлів та їх опис. Створено інструкцію користувача та наведено вигляд інтерфейсу програми з детальним описом усіх функцій.

ВИСНОВОК

Тестування – важливий процес у визначенні рівня знань студентів. Можливість автоматизувати процес створення тестів для викладача має дуже велике значення, адже це дозволить зменшити затрати часу на підготовку тестових завдань, полегшить спостереження за процесом тестування, дасть змогу вести статистику завдань та слідкувати за результатами тестувань.

Мета розробленої системи полягає якраз у полегшенні проведення тестування зі сторони студента.

Система проектувалась на основі досліджених трьох аналогів, програмної системи для тестування знань “Айрен” та “MyTestStudentPRO”. Було враховано усі їх недоліки та переваги та застосовано ці знання для розробки продукту.

Аріхектура програмної системи розроблена із повною відповідністю до поставлених вимог, здійснено аналіз потоків даних для подальшого проектування.

Система створена з використанням мови програмування C# у середовищі розробки Microsoft Visual Studio 2013. База даних базується на реляційній моделі

даних. Ця модель забезпечує найвищу надійність збереження даних, та високу швидкодію при роботі з ними. Бази даних реалізовано за допомогою Microsoft SQL Server та Microsoft Management Studio. Це пояснено тим, що ці програмні продукти добре інтегровані з іншими продуктами від Microsoft, тому це дозволило зробити проектування та програмування дуже простим та призвело до максимального зниження кількості помилок.

Тестування програмної системи не показало ніяких помилок. Функціональне тестування пройшло успішно, вказавши повну відповідність поставленим вимогам. Тестування графічного інтерфейсу не виявило помилок, що означає хорошу організацію та оптимізацію користувацького інтерфейсу підсистеми “Студент”.

Створено документацію для користувача, яка містить опис програмного продукту і його функцій. Розроблено інструкцію щодо розгортання програмного продукту, вказано вимоги до апаратного та програмного забезпечення. Наведено список файлів необхідних для коректної роботи програми та їх опис.

Отже, підсистема “Студент” програмної системи тестування студентів по SQL запитах є надійною та зручною системою, яка дозволяє автоматизувати процес проведення тестування, забезпечує запис результатів та збереження їх у базі даних. Система захищена від несанкціонованого доступу, тому тести та правильні відповіді до них не будуть втрачені чи змінені стороннім користувачем. Робота системи є швидкою, з мінімальною кількістю збоїв.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Джеймс Р. Грофф, Пол Н. Вайнберг, Эндрю Дж. Оппель. SQL: полное руководство, 3-е издание = SQL: The Complete Reference, Third Edition. — М.: «Вильямс», 2014. — 960 с.
2. Крис Фиайли. SQL: Руководство по изучению языка. — М.: Peachpit Press, 2003. — 456 с.
3. К. Дж. Дейт. Введение в системы баз данных / Пер. с англ. — 8-е изд. — М.: Вильямс, 2005. — 1328 с.
4. Codd, Edgar F (June 1970). "A Relational Model of Data for Large Shared Data Banks". Communications of the ACM 13 (6): 377–87.
5. C. J. Date with Hugh Darwen: A Guide to the SQL standard : a users guide to the standard database language SQL, 4th ed., Addison Wesley, USA 1997.
6. Лайза Криспин, Джанет Грегори Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд = Agile Testing: A Practical Guide for Testers and Agile Teams. — М. : «Вильямс», 2010. — 464 с. — (Addison-Wesley Signature Series). — 1000 прим.

7. Канер Кем, Фолк Джек, Нгуен Енг Кек Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений. — Киев : ДиаСофт, 2001. — 544 с. — ISBN 9667393879.
8. Калбертсон Роберт, Браун Крис, Кобб Гэри Быстрое тестирование. — М. : «Вильямс», 2002. — 374 с.
9. Синицын С. В., Налютин Н. Ю. Верификация программного обеспечения. — М. : БИНОМ, 2008. — 368 с.
10. Бейзер Б. Тестирование чёрного ящика. Технологии функционального тестирования программного обеспечения и систем. — СПб. : Питер, 2004. — 320 с.
11. *Beizer, Boris (1990). Software Testing Techniques (Second ed.). New York: Van Nostrand Reinhold. pp. 21,430. ISBN 0-442-20672-0.*
12. IEEE (1990). IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York: IEEE. ISBN 1-55937-079-3.
13. ISO/IEC/IEEE 29119-1:2013 – Software and Systems Engineering – Software Testing – Part 1 – Concepts and Definitions; Section 4.38.
14. Rodríguez, Ismael; Llana, Luis; Rabanal, Pablo (2014). "A General Testability Theory: Classes, properties, complexity, and testing reductions". IEEE Transactions on Software Engineering 40 (9): 862–894. doi:10.1109/TSE.2014.2331690.ISSN 0098-5589.
15. Hershey, William; Easthope, Carol (1972). A set theoretic data structure and retrieval language. Spring Joint Computer Conference, May 1972. ACM SIGIR Forum 7 (4). pp. 45–55. doi:10.1145/1095495.1095500
16. *Childs, David L. (1968). "Description of a set-theoretic data structure". CONCOMP (Research in Conversational Use of Computers) Project. Technical Report 3. University of Michigan.*

17. Дивак М.П. Системний аналіз та проектування КІС /М.П.Дивак// Навчальний посібник – Т.: Економічна думка. – 2004.
18. Дейт К. Введение в системы баз даннях /К.Дейт// – К.; М.; СПб.: Изд.дом “Вильямс”. – 2000. –560 с.
19. Буч Г., Рамбо Дж., Джекобсон А. Язык UML. Руководство пользователя. — Пер. с англ. — М.: ДМК, 2000. — 432 с.
20. Фаулер М., Скотт К. UML. Основы. — Пер. с англ. — СПб: Символ-Плюс, 2002. — 192 с., ил. ISBN 5-93286-032-4
21. Никаноров, С. П. Системний аналіз: етап розвитку методології рішення. – 2001. – Выпуск 12. – С. 62–87. -фрагмент
22. Губанов В.А. і др. Введення в системний аналіз: Навчальний посібник /Под ред. Л.А. Петросяна. - Л.: Изд-во ЛГУ, 1988.
23. Клир Дж. Системологія. Автоматизація рішення системних задач.ер. с англ. – М.:1990.. – С. 62–87.

ДОДАТОК А

Лістинг коду програми

```
using DataLib;
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Configuration;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Task_Generation
{
   public partial class AdminMainForm : Form
   {
      public AdminMainForm()
```

```
{
    InitializeComponent();
    try
    {
        using (StreamReader sr = new StreamReader(ConfigurationManager.ConnectionStrings["DefaultTask"].ConnectionString))
        {
            String line = sr.ReadToEnd();
            var listTC = JsonConvert.DeserializeObject<List<TaskControl>>(line);
            foreach (var item in listTC)
            {
                Tasklist.Items.Add(item.description);
            }

        }
    }
    catch (Exception exept)
    {
        MessageBox.Show(exept.Message);
    };
}

private void AdminMainForm_Load(object sender, EventArgs e)
{

}

private void AddTaskBtnClick(object sender, EventArgs e)
{
    Task_Gen TaskForm = new Task_Gen();
    if (TaskForm.ShowDialog() == DialogResult.OK)
    {
        Tasklist.Items.Clear();
        try
        {
```

```
            using (StreamReader sr = new StreamReader(ConfigurationManager.ConnectionStrings["DefaultTask"].ConnectionString))
            {
                String line = sr.ReadToEnd();
                var listTC = JsonConvert.DeserializeObject<List<TaskControl>>(line);
                foreach (var item in listTC)
                {
                    Tasklist.Items.Add(item.description);
                }
            }
        }
        catch (Exception exept)
        {
            MessageBox.Show(exept.Message);
        };
    }
}

private void DeleteBtn_Click(object sender, EventArgs e)
{
    List<TaskControl> task = new List<TaskControl>();
    using (StreamReader sr = new StreamReader(ConfigurationManager.ConnectionStrings["DefaultTask"].ConnectionString))
    {
        String line = sr.ReadToEnd();
        var listTC = JsonConvert.DeserializeObject<List<TaskControl>>(line);
        foreach (var item in listTC)
        {
            task.Add(item);
        }
    }

    while (Tasklist.SelectedItems.Count > 0)
    {
        int index = Tasklist.SelectedIndex;
```

```
            Tasklist.Items.Remove(Tasklist.SelectedItems[0]);
            task.RemoveAt(index);
            string jstr = JsonConvert.SerializeObject(task, Formatting.Indented);
            System.IO.File.WriteAllText(@"D:\task.json", jstr);
        }
    }

    private void TaskListLabel_Click(object sender, EventArgs e)
    {

    }
  }
}


namespace SQLDev
{
  public partial class UserLogin : Form
  {
    public UserLogin()
    {
      InitializeComponent();
      try
      {
        using (StreamReader sr = new StreamReader(ConfigurationManager.ConnectionStrings["Students"].ConnectionString))
        {
          String line = sr.ReadToEnd();
          var groupList = JsonConvert.DeserializeObject<List<GroupControl>>(line);
          foreach (var group in groupList)
          {
            GroupCombo.Items.Add(group.groupName);
          }

        }
```

```
        }
        catch (Exception exept)
        {
            MessageBox.Show(exept.Message);
        };
    }

    private void LogInBtn_Click(object sender, EventArgs e)
    {
        if (String.IsNullOrWhiteSpace(GroupCombo.Text))
        {
            GroupStat.ForeColor = Color.Red;
            GroupStat.Text = "Choose group!";
        }
        else
        {
            if (String.IsNullOrWhiteSpace(StudCombo.Text))
            {
                StudStat.ForeColor = Color.Red;
                StudStat.Text = "Choose student!";
            }
            else
            {
                TaskMenu menuForm = new TaskMenu(StudCombo.SelectedItem.ToString() + ", " +
GroupCombo.SelectedItem.ToString());
                menuForm.FormClosed += new FormClosedEventHandler(menuForm_FormClosed);
                menuForm.Show();
                this.Hide();
            }
        }
    }

    private void menuForm_FormClosed(object sender, FormClosedEventArgs e)
    {
        this.Close();
```

```
        }

        private void GroupCombo_SelectedIndexChanged(object sender, EventArgs e)
        {
            GroupStat.Text = "";
            StudCombo.Items.Clear();
            try
            {
                using (StreamReader sr = new StreamReader(ConfigurationManager.ConnectionStrings["Students"].ConnectionString))
                {
                    String line = sr.ReadToEnd();
                    List<GroupControl> groupList = JsonConvert.DeserializeObject<List<GroupControl>>(line);
                    foreach (var stud in groupList[GroupCombo.SelectedIndex].students)
                    {
                        StudCombo.Items.Add(stud.name);
                    }
                }
            }
            catch (Exception exept)
            {
                MessageBox.Show(exept.Message);
            }
        }

        private void StudCombo_SelectedIndexChanged(object sender, EventArgs e)
        {
            StudStat.Text = "";
        }
    }
}


namespace DataLib
{
    public class GroupControl
```

```
    {
        public string groupName;
        public List<StudentControl> students;
    }
}

namespace DataLib
{
    public class TaskControl
    {
        public string description { get; set; }
        public string imgPath { get; set; }
        public string correctSQL { get; set; }
    }
}

namespace DataLib
{
    public class StudentControl
    {
        public string name { get; set; }
    }
}


namespace SQLDev
{
    public partial class TaskView : Form
    {

        public TaskView(int taskNum)
        {
            InitializeComponent();
            try
            {
                using (StreamReader sr = new StreamReader(ConfigurationManager.ConnectionStrings["DefaultTask"].ConnectionString))
```

```
            {
                String line = sr.ReadToEnd();
                var listTC = JsonConvert.DeserializeObject<List<TaskControl>>(line);
                var currTask = listTC[taskNum];
                DescLabel.Text = currTask.description;
                ERDpictureBox.Image = new Bitmap(currTask.imgPath);
                ERDpictureBox.SizeMode = PictureBoxSizeMode.Zoom;
            }
        }
        catch (Exception exept)
        {
            MessageBox.Show(exept.Message);
        };
    }

    private void TaskView_Load(object sender, EventArgs e)
    {

    }

    private void TaskView_SizeChanged(object sender, EventArgs e)
    {

    }
  }
}


namespace SQLDev
{
  public partial class TaskMenu : Form
  {
    private Boolean[] Marks;
    private Int32 TaskCount = 0;
    Button[] buttonArray;
    public TaskMenu(string student)
```

```
{
    InitializeComponent();
    StudLab.Text += student;
}

private void TaskMenu_Load(object sender, EventArgs e)
{
    this.TaskCount = getTaskCount();
    CreatingNewButtons(this.TaskCount);
    this.Marks = new Boolean[this.TaskCount];
    for (Int32 i = 0; i < Marks.Length; i++)
    {
        this.Marks[i] = false;
    }
}

private void CreatingNewButtons(int taskCount)
{
    int horizotal = 13;
    int vertical = 65;
    buttonArray = new Button[taskCount];

    for (int i = 0; i < buttonArray.Length; i++)
    {
        buttonArray[i] = new Button();
        buttonArray[i].Name = "taskBtn_" + i;
        buttonArray[i].Size = new Size(65, 65);
        buttonArray[i].Location = new Point(horizotal, vertical);
        buttonArray[i].Text = "Task " + (i+1).ToString();
        buttonArray[i].Click += TaskBtn_Click;
        if ((i+1)%5 == 0)
        {
            horizotal = 13;
            vertical = vertical + 70;
        }
```

```
            else
            {
                horizotal = horizotal + 70;
            }
            this.Controls.Add(buttonArray[i]);
        }
    }

    private int getTaskCount()
    {
        try
        {
            using (StreamReader sr = new
StreamReader(ConfigurationManager.ConnectionStrings["DefaultTask"].ConnectionString))
            {
                String line = sr.ReadToEnd();
                var taskList = JsonConvert.DeserializeObject<List<TaskControl>>(line);
                return taskList.Count;
            }
        }
        catch (Exception exept)
        {
            MessageBox.Show(exept.Message);
            return 0;
        };
    }

    private void  TaskBtn_Click(object sender, EventArgs e)
    {
        string[] words = (((Button)sender).Name).Split('_');

        MainForm taskForm = new MainForm(Convert.ToInt32(words[1]));
        taskForm.Text += (Convert.ToInt32(words[1])+1).ToString();
        switch (taskForm.ShowDialog())
        {
```

```
            case DialogResult.Abort:
                break;
            case DialogResult.Cancel:
                break;
            case DialogResult.Ignore:
                break;
            case DialogResult.No:
                {
                    if (!this.Marks[Convert.ToInt32(words[1])])
                        this.buttonArray[Convert.ToInt32(words[1])].BackColor = System.Drawing.Color.Salmon;
                }
                break;
            case DialogResult.None:
                break;
            case DialogResult.OK:
                {
                    this.Marks[Convert.ToInt32(words[1])] = true;
                    this.StudentMark.Text = "Mark: " + GetStudentMark().ToString();
                    this.buttonArray[Convert.ToInt32(words[1])].BackColor = System.Drawing.Color.LightGreen;
                }
                break;
            case DialogResult.Retry:
                break;
            case DialogResult.Yes:
                break;
            default:
                break;
        }
    }

    private void LogOutBtn_Click(object sender, EventArgs e)
    {
        DialogResult result = MessageBox.Show("Are you sure to log out?", "Log Out", MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation);
```

```
    if (result == DialogResult.Yes)
    {
        UserLogin loginForm = new UserLogin();
        loginForm.FormClosed += new FormClosedEventHandler(loginForm_FormClosed);
        loginForm.Show();
        this.Hide();
    }
}

private void loginForm_FormClosed(object sender, FormClosedEventArgs e)
{
    this.Close();
}

private void TaskMenu_SizeChanged(object sender, EventArgs e)
{

}
private Int32 CountCorrectTasks()
{
    Int32 counter = 0;
    for (Int32 i = 0; i < this.Marks.Length; i++)
    {
        if (this.Marks[i])
        {
            counter++;
        }
    }
    return counter;
}
private Int32 GetStudentMark()
{
    Double correct = CountCorrectTasks();
    Double result = correct / this.TaskCount;
    result *= 100;
```

```
        return Convert.ToInt32(result);
    }
}
```

1. №	Action	Date(s)	Status	Found bug(s)

```
}
```

ДОДАТОК Б

1	Check -> “Authorization”	01.05.2016	P	-
2	Check -> “Task list”	01.05.2016	P	-
3	Check -> “Student info”	01.05.2016	P	-
4	Check -> “Results text”	01.05.2016	P	-
5	Check -> “Select task”	01.05.2016	P	-
6	Check -> “Query input”	01.05.2016	P	-
7	Check -> “Execute button”	01.05.2016	P	-
8	Check -> “Result field”	01.05.2016	P	-
9	Check -> “View task info”	01.05.2016	P	-
10	Check -> “Task info text and image”	01.05.2016	P	-
11	Check -> “Error text”	01.05.2016	P	-
12	Check -> “Return button”	01.05.2016	P	-

Чек-ліст