**ЗМІСТ**

1. ВСТУП
2. МЕТА ТА ПОСТАНОВКА ЗАВДАННЯ
3. ЗАГАЛЬНА АРХІТЕКТУРА ПРОЕКТУ
4. ДЕТАЛЬНИЙ ОПИС АРХІТЕКТУРИ ПРОЕКТУ
5. UML ДІГРАМИ ПРОЕКТУ
6. ІНСТРУКЦІЯ З ВИКОРИСТАННЯ ЗАСТОСУВАННЯ
7. ВИСНОВКИ
8. СПИСОК ВИКОРИСТАННОЇ ЛІТЕРАТУРИ

**1. ВСТУП**

Без перебільшення можна стверджувати, що сьогодні системи тестування зробили справжню революцію в навчанні. Вони допомогли вдосконалити оцінювання знань студентів. Тепер викладач набагато легше і зручніше може проводити іспити та тести, відстежувати результати і прогрес своїх учнів. Час, коли вчитель мав створювати кожен тест власноруч, записувати результати та вираховувати середній бал, давно минули. Сучасні системи тестування допомагають перевірити компетентність кожного учня без зайвих підрахунків.

Саме для цього і створюється дана курсова робота. Курсова робота полягає у створенні тестування для створення тестів та тестування знань студентів.

**2. МЕТА ТА ПОСТАНОВКА ЗАВДАННЯ**

**Мета** – навчитися основним процесам проектування та створення програмного забезпечення з використанням кращих практик написання коду і виконати розробку програмної системи з консольним додатком на платформі .NET у відповідності з принципами багатошарової архітектури програмних систем та скласти проектну документацію.

**Завдання** - спроектувати та реалізувати на мові С# програмне забезпечення відповідно заданого варіанту:



**Опис завдання**

*Функціональні вимоги до програмного забезпечення*

1. Керування питаннями тестів

1.1. Можливість додавати питання

1.2. Можливість видаляти питання

1.3. Можливість змінити питання

1.4. Можливість переглянути перелік всіх питань

2. Керування відповідями

2.1. Можливість додавати відповідь

2.2. Можливість видаляти відповідь

2.3. Можливість змінювати відповідь

2.4. Можливість переглянути перелік всіх відповідей до конкретного питання

2.4.1. Помітити правильну відповідь позначкою «+»

2.5. Варіанти відповідей до питань генеруються автоматично

2.5.1. Передбачити наявність правильної відповіді

3. Керування тестами

3.1. Можливість створити тест

3.2. Можливість змінювати тест

3.2.1. змінити кількість питань у тесті

3.2.2. змінити час відведений на одне питання

3.3. Можливість порахувати процент правильних відповідей

3.4. Можливість передчасно вийти з тесту

4. Пошук

4.1. Пошук тестів

4.2. Перегляд статистики тестів

**3. ЗАГАЛЬНА АРХІТЕКТУРА ПРОЕКТУ**

Багаторівнева архітектура – клієнт-серверна архітектура, в якій розділяються функції представлення, обробки і зберігання даних. Найбільш поширеним різновидом багаторівневої архітектури є трирівнева архітектура.

N-рівнева архітектура додатка надає модель, по якій розробники можуть створювати гнучкі і повторно-використовуваних програм. Поділяючи додаток на рівні абстракції, розробники набувають можливість внесення змін в якийсь певний шар, замість того, щоб переробляти все додаток цілком. Трирівнева архітектура зазвичай складається з рівня уявлення, рівня бізнес-логіки і рівня зберігання даних.

Робота складається з трьох окремих проектів відповідно до багатошарової архітектури системи.

Загальна архітектура проекту складається з наступних рівнів:

* Рівень **DAL** (Data Access Layer) – рівень доступу до даних, на якому розміщені класи сутностей, конструктори, та методи доступу. Являє собою базу даних додатку.

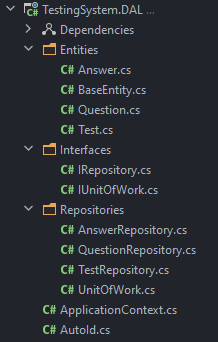


Рис.1. Структура Data Access Layer

* Рівень **BLL** (Business Logic Layer) – шар бізнес-логіки, який посилається на рівень DAL і вміщає в себе всю обробку даних, методи та основний функціонал програми. Шар BLL відокремлює рівень представлення від рівня доступу до даних та накладає бізнес-правила.

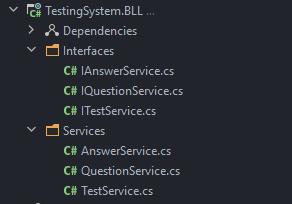


Рис.2. Структура Business Logic Layer

* Рівень **PL** (Presentation Layer) – інтерфейс програмної системи, який посилається на рівень BLL і виконує функцію інтерфейсу для взаємодії програми з користувачем.

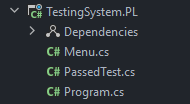


Рис.3. Структура Presentation Layer

* Також в додатку є окремий проект - модульні тести (**TestingSystem.Tests**). Він складається з певної кількості тестових класів, в яких і відбуваються всі перевірки коректності роботи самої програми.

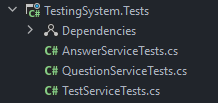


Рис.4. TestingSystem.Tests

Використання багаторівневої архітектури забезпечує наступні важливі переваги:

* *Ізоляція*. Розробка та оновлення ПЗ можуть бути ізольовані рамками одного рівня.
* *Продуктивність*. Розподіл рівнів на окремі фізичні комп'ютери підвищує продуктивність і відмовостійкість.
* *Тестованість*. Рівні допускають незалежне тестування.

Перевагами застосування такої архітектури є простота розробки та тестування. Серед недоліків можна виділити можливі складнощі з продуктивністю і масштабуванням.

**4. ДЕТАЛЬНИЙ ОПИС АРХІТЕКТУРИ ПРОЕКТУ**

**Рівень доступу до даних**

Було створено бібліотеку класів для рівня доступу до даних.

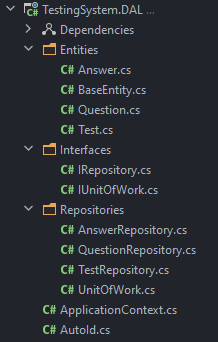


Рис. 6. Бібліотека класів рівня доступу даних

Рис. 7. Зразок деяких особливостей реалізації

Завдяки такій реалізації методу ми отримуємо код, який легше читається. Він може виконувати однаковий функціонал, але використовує різні змінні. Щоб не писати однакові методи, які займають зайву пам’ять і ускладнюють читання коду. За допомогою конструкції “switch” я спростив код та скоротив використання пам’яті.

**Рівень бізнес-логіки**

Було створено бібліотеку класів для бізнес-логіки.

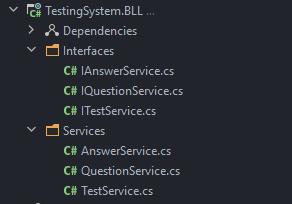


Рис. 8. Бібліотека класів рівня бізнес-логіки



Рис. 9. Зразок вихідного коду

**Рівень представлення**

Відповідно до розподілу завдань мною було реалізовано рівень представлення та розроблено загальну архітектуру програмної системи.

Було створено клас Menu з методами для взаємодії з тестами, питаннями та відповідями.



Рис. 12. Вихідний код PL

**Тести**

Для створення Unit-тестів вибираються невеликі ділянки коду, які треба протестувати. Тестована ділянка, як правило, повинна бути менше класу. У більшості випадків тестується окремий метод класу або навіть частина функціоналу методу. Упор на невеликі ділянки дозволяє досить швидко писати простенькі тести.

Одного разу написаний код нерідко читають багато разів, тому важливо писати зрозумілий код. Особливо це важливо в Unit-тестах, де в разі невдачі при тестуванні розробник повинен швидко прочитати вихідний код і зрозуміти в чому проблема і як її виправити. А використання невеликих ділянок коду значно спрощує подібну роботу.

При тестуванні важливо ізолювати тестований код від решти програми, з якою він взаємодіє, щоб потім чітко визначити можливість помилок саме в цьому ізольованому коді. Що спрощує і підвищує контроль над окремими компонентами програми.

В курсовому проекті було застосовано фреймворк NUnit.

Тести були написані за принципом AAA.

* **Arrange**: Це перший крок додаткового модульного тестування. Тут ми організуємо тест, іншими словами зробимо необхідну установку тесту.
* **Act**: Це середній крок застосування одиничного кроку. На цьому кроці ми виконаємо тест. Іншими словами, ми зробимо фактичне тестування одиниці, і результат буде отриманий з тестової програми.
* **Assert**: Це останній крок заявки на одиничне тестування. На цьому кроці ми звіряємо повернений результат із очікуваними результатами.



Рис. 10. Зразок вихідного коду тестів

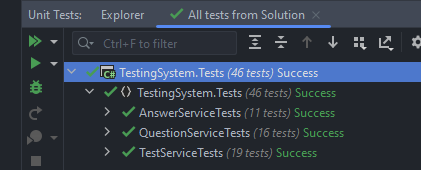


Рис. 11. Список виконаних тестів

**5. UML ДІАГРАМИ ПРОЕКТУ**

Діаграма використання

Рис. 13. Діаграма використання

Діаграма компонентів

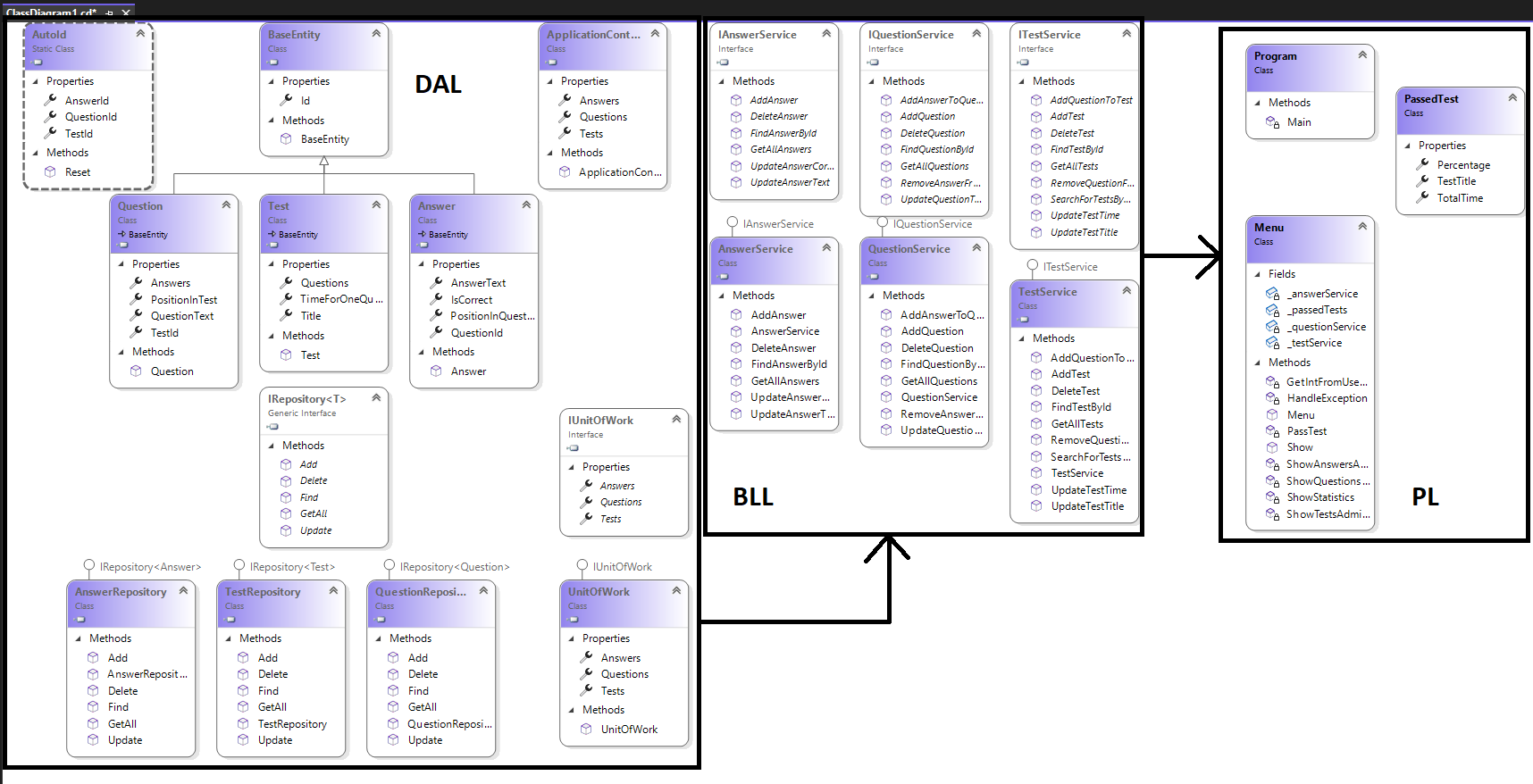


Рис. 14. Діаграма компонентів

На цьому рисунку зображена діаграма компонентів. Як бачите в проекті DAL розміщенні усі класи сутностей. Між ними ніякого зв'язку немає. Їх лише об’єднує клас UnitOfWork. Проект BLL вміщає в себе весь функціонал програми, який приймає запит від PL, обробляє і тільки тоді передає його в DAL. Проект PL вміщає в себе безпосередньо інтерфейс програми, який посилається на проект BLL , щоб обробляти отриману інформацію і за потреби виводити його на консоль.

Діаграми класів рівнів проекту

Діаграма класів - структурна діаграма мови моделювання UML, що демонструє загальну структуру ієрархії класів системи, їх кооперацій, атрибутів (полів), методів, інтерфейсів і взаємозв'язків між ними. Широко застосовується не тільки для документування та візуалізації, але також для конструювання за допомогою прямого або зворотного проектування.

Метою створення діаграми класів є графічне представлення статичної структури декларативних елементів системи (класів, типів і т. П.).

Діаграма класів є ключовим елементом в об'єктно-орієнтованому моделюванні. На діаграмі класи представлені в рамках, що містять три компоненти:

У верхній частині написано ім'я класу. Ім'я класу вирівнюється по центру і пишеться напівжирним шрифтом. Імена класів починаються з великої літери. Якщо клас абстрактний - то його ім'я пишеться напівжирним курсивом.

Посередині розташовуються поля (атрибути) класу. Вони вирівняні по лівому краю і починаються з маленької літери.

Нижня частина містить методи класу. Вони також вирівняні по лівому краю і пишуться з малої літери.

**Діаграма послідовності**

**КЛІЄНТ**

**ЗАСТОСУНОК**

**МЕНЕДЖЕР ВІДДІЛУ КАДРІВ**

**Запустити застосунок**

**Показати весь функціонал програми**

**Внести зміни**

**Обробити і зберегти зміни**

Рис. 15. Діаграма послідовності

**Діаграма розгортання**

БаАІВавіа

**База даних**

**Веб-сайт університету**

**Веб-сайт зворотнього зв'язку**

**Інтерфейс бази даних**

**Користувацький веб-інтерфейс**

**Комп'ютер менеджера**

**WEB-сервер**

**Сервер бази даних університету**

**Комп'ютер користувача**

Рис. 16. Діаграма розгортання

**6. ІНСТРУКЦІЯ З ВИКОРИСТАННЯ ЗАСТОСУВАННЯ**

Після включення застосунку потрапляємо у головне меню.

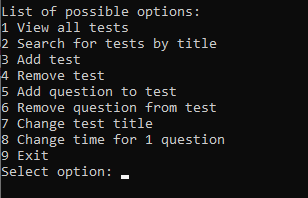
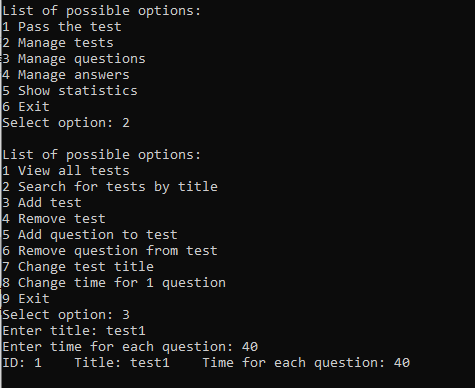


Рис. 17. Головне меню.

Додамо тест.

Рис. 18. Меню додавання тесту.



Спробуємо провести пошук тестів за назвою.

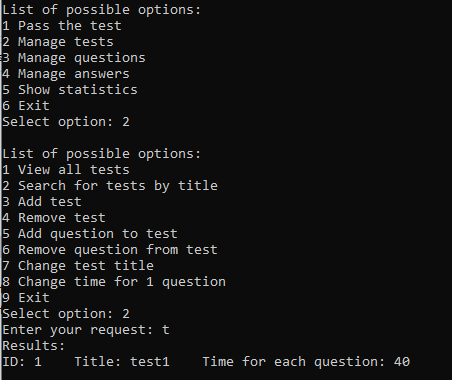


Рис. 19. Меню пошуку тестів.

Додамо питання.

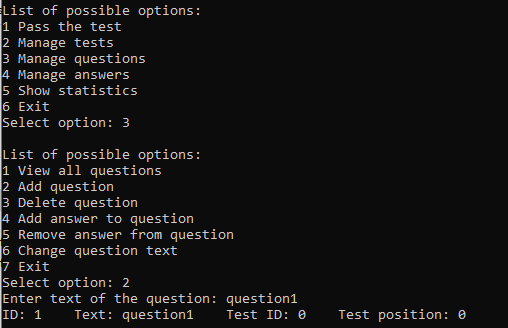


Рис. 20. Меню додавання питання до бази.

Додамо правильний варіант відповіді для питання.

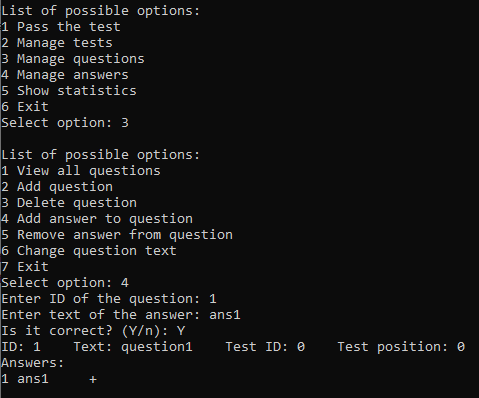


Рис. 21. Меню додавання відповіді до питання.

Додамо неправильний варіант відповіді для питання.

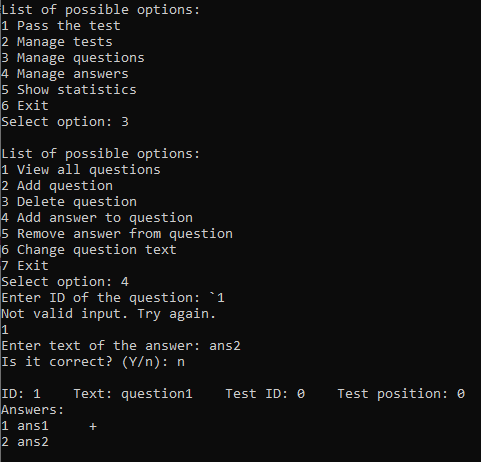


Рис. 22. Меню додавання відповіді до питання.

Переглянемо список питань.

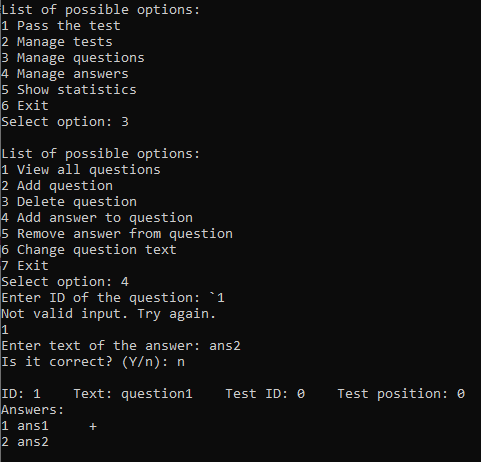


Рис. 23. Перегляд списку питань.

Змінимо текст питання.

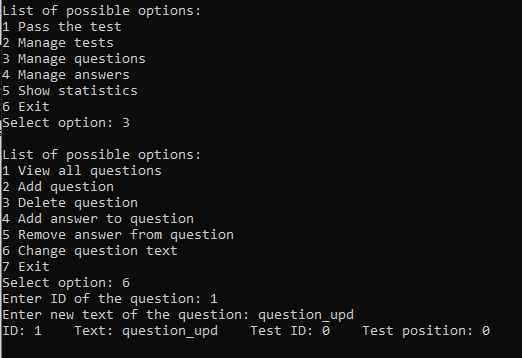


Рис. 24. Меню зміни тексту питання.

Видалимо один варіант відповіді з питання.

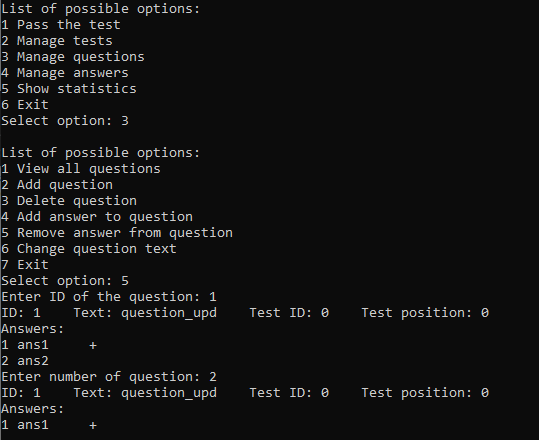


Рис. 25. Видалення варіанту відповіді з питання.

Додамо ще одне питання.

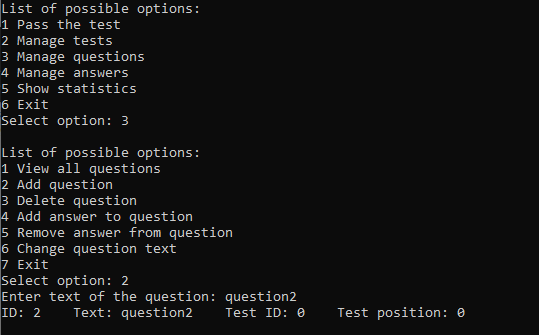


Рис. 26. Меню додавання питання до бази.

Видалимо щойно додане питання.

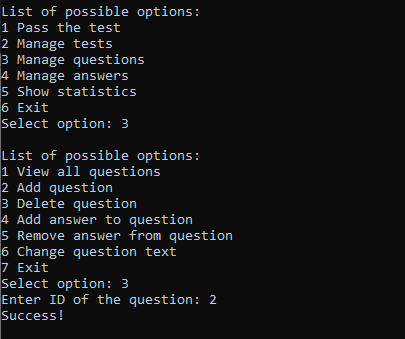


Рис. 27. Меню видалення питання з бази.

Додамо існуюче питання до тесту.

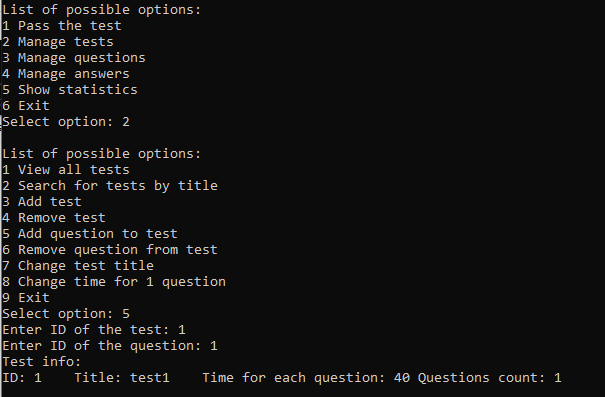


Рис. 28. Меню додавання питання до тесту.

Змінимо назву тесту.

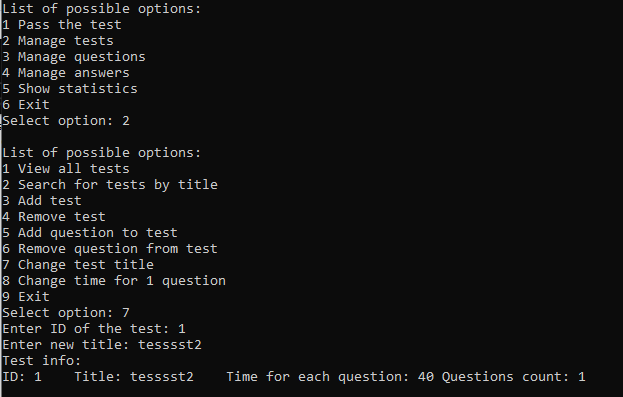


Рис. 29. Меню зміни назви тесту.

Змінимо час на 1 питання тесту.

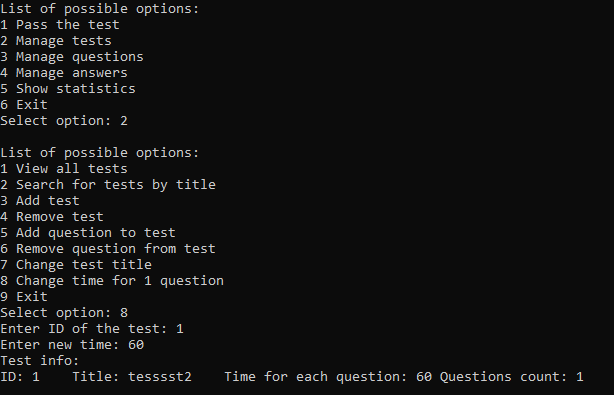


Рис. 30. Меню зміни часу на 1 питання тесту.

Спробуємо пройти тест.

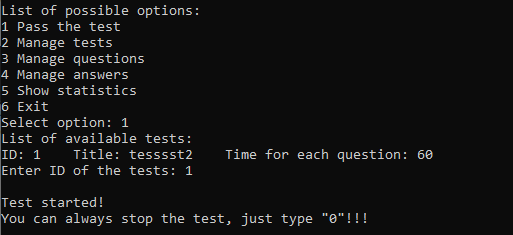


Рис. 31. Меню запуску тесту.

Після успішного проходження тесту отримаємо свою статистику.

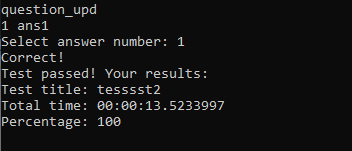


Рис. 32. Статистика після проходження тесту.

Спробуємо перервати тест під час виконання.

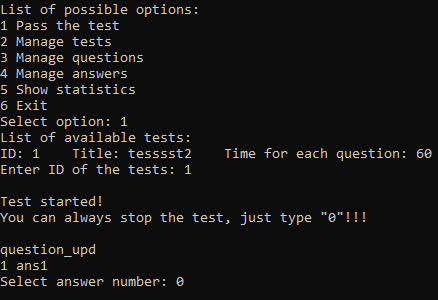


Рис. 32. Переривання тесту під час виконання.

Переглянемо статистику пройдених тестів.

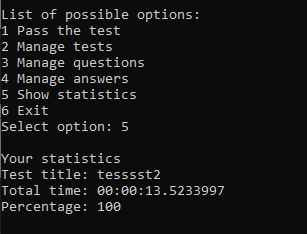


Рис. 33. Статистика пройдених тестів.

Переглянемо список всіх відповідей на всі питання.

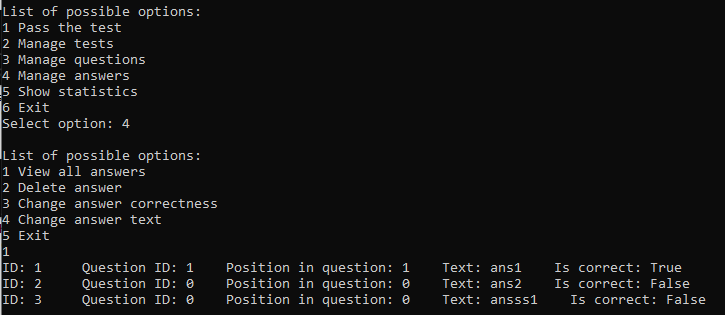


Рис. 34. Меню перегляду відповідей.

Видалимо одну з відповідей.

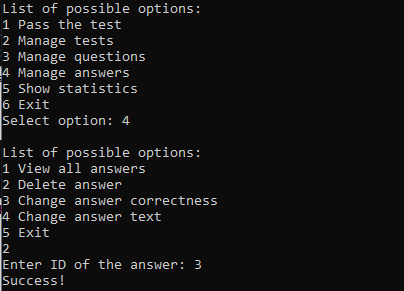


Рис. 35. Меню видалення відповіді.

Змінимо правильність відповіді.

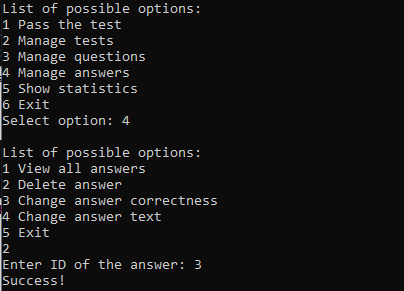


Рис. 36. Меню зміни правильності відповіді.

Змінимо текст відповіді.

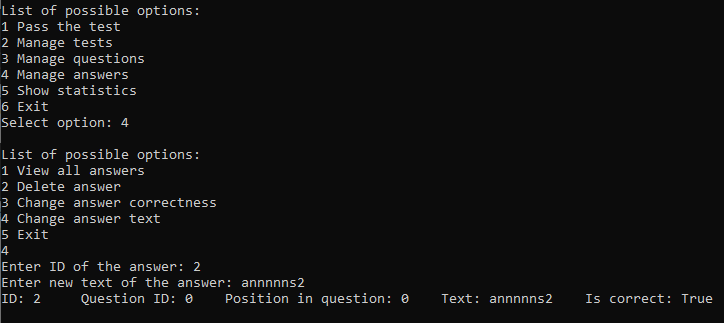


Рис. 37. Меню зміни тексту відповіді.

Видалимо тест.

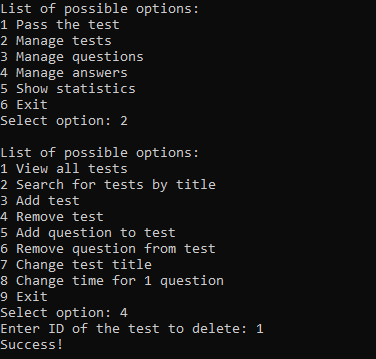


Рис. 38. Меню видалення тесту.

**7. ВИСНОВКИ**

В даній курсовій роботі було спроектовано та розроблено програмну систему з багаторівневою системною архітектурою, яка виконує функцію системи для тестування студентів. Було реалізовано можливість роботи з базою даних тестів.

Для проектування і розробки даної системи було використано IDE Jetbrains Rider. Для створення діаграм було використано програмний засіб Microsoft Visio 2019.

Було узагальнено та використано свої знання з об’єктно-орієнтованого програмування на мові С# при використанні методів проектування та розробки багаторівневої архітектури та побудовано діаграму класів.

Було реалізовано багаторівневу архітектуру програми та вивчено правила «високого» стилю програмування та застосовано їх під час написання програми.

**8. СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ**

1. Цикл статей С#:

[https://stackoverflow.com](https://stackoverflow.com/)

1. Розділ присвячений мові програмування C# і платформі .NET:

<https://metanit.com/sharp/>

1. Стаття “C# Corner - Community of Software and Data Developers”:

<https://www.c-sharpcorner.com/>

1. Посібник “Уроки з С# і платформи .NET Framework”

[https://professorweb.ru](https://professorweb.ru/)

1. Цикл статей “wiki for .NET developers”

<http://pinvoke.net/>