МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЖИТОМИРСЬКИЙ АГРОТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ

ВІДДІЛЕННЯ
«ІНЖЕНЕРНА ІНФРАСТРУКТУРА ТА КОМП’ЮТЕРНІ НАУКИ»

ЦИКЛОВА КОМІСІЯ СПЕЦІАЛЬНОСТІ
«КОМП'ЮТЕРНІ НАУКИ»

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня фаховий молодший бакалавр
за спеціальністю 122 Комп'ютерні науки
на тему:

«Інтерактивна система тестування та оцінювання знань студентів»

Виконав здобувач освіти групи П-42
Степанюк Віталій Вячеславович

Керівник роботи:
Устименко Ярослав Іванович

Рецензент:

Житомир – 2026

Зміст

РЕФЕРАТ

Кваліфікаційна робота на тему «Інтерактивна система тестування та оцінювання знань студентів» присвячена розробці автономного настільного програмного забезпечення для автоматизації контролю навчальних досягнень. Пояснювальна записка містить 63 сторінки основного тексту, 19 рисунків, 18 таблиць, 2 додатки та 25 використаних джерел. У роботі представлено 11 фрагментів програмного коду.

Об'єкт дослідження — процес автоматизованого контролю та оцінювання знань студентів у навчальному закладі.

Предмет дослідження — методи та технології розробки настільного ПЗ для інтерактивного тестування.

Метою роботи є проектування та реалізація системи з підтримкою чотирьох типів питань, автоматичним підрахунком балів та розмежуванням прав доступу. Розробку виконано на мові C# 13 (платформа .NET 9) із використанням архітектурного шаблону MVVM та СУБД SQLite.

Практична цінність полягає у створенні готового продукту, що працює в автономному режимі без Інтернет-з’єднання. Система забезпечує візуалізацію статистики успішності за допомогою бібліотеки OxyPlot та підтримує імпорт/експорт тестів у форматах JSON/XML. Результати роботи можуть бути впроваджені в освітній процес для підвищення об’єктивності та оперативності перевірки знань.

Ключові слова: ТЕСТУВАННЯ, ОЦІНЮВАННЯ, C#, .NET 9, WPF, MVVM, SQLITE, АВТОМАТИЗАЦІЯ КОНТРОЛЮ.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД — база даних

ЗВО — заклад вищої освіти

ПЗ — програмне забезпечення

СУБД — система управління базами даних

CSV (Comma Separated Values) — текстовий формат, призначений для представлення табличних даних

DI (Dependency Injection) — впровадження залежностей

EF Core (Entity Framework Core) — об'єктно-реляційне відображення для .NET

JSON (JavaScript Object Notation) — текстовий формат обміну даними

LMS (Learning Management System) — система управління навчанням

MVVM (Model-View-ViewModel) — архітектурний шаблон проектування програмного забезпечення

ORM (Object-Relational Mapping) — технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування

UI (User Interface) — інтерфейс користувача

UML (Unified Modeling Language) — уніфікована мова моделювання для опису, візуалізації та проектування програмних систем

WPF (Windows Presentation Foundation) — графічна підсистема для побудови настільних клієнтських застосунків у Windows

XAML (Extensible Application Markup Language) — декларативна мова розмітки для створення інтерфейсів

XML (Extensible Markup Language) — розширювана мова розмітки для зберігання та передачі даних

ВСТУП

Сучасна система вищої освіти України зазнає суттєвих змін, зумовлених широким впровадженням інформаційних технологій у навчальний процес. Оперативний, об'єктивний та систематичний контроль знань студентів є невід'ємною складовою якісної освіти, адже забезпечує зворотний зв'язок між викладачем і студентом, стимулює самостійну роботу та підвищує мотивацію до навчання.

Традиційні форми контролю знань — письмові контрольні роботи, усні відповіді та паперові тести — мають низку суттєвих недоліків: значні витрати часу на перевірку результатів, суб'єктивність оцінювання, відсутність можливості швидкого аналізу статистики успішності групи, а також складність зберігання й опрацювання результатів. Натомість автоматизовані системи тестування усувають ці недоліки, забезпечуючи швидку, точну та безсторонню перевірку рівня знань.

Незважаючи на наявність таких зарубіжних платформ, як Moodle, Google Classroom та iSpring, вони не завжди відповідають специфічним вимогам окремих навчальних закладів: потребують постійного інтернет-з'єднання, передбачають складне адміністрування або є платними. Тому розробка власної компактної десктопної системи тестування, орієнтованої на автономну роботу у локальній мережі навчального закладу, залишається актуальним завданням.

Мета роботи — проектування та реалізація інтерактивної настільної системи тестування та оцінювання знань студентів із підтримкою кількох типів питань, автоматичним підрахунком балів, графічним відображенням результатів та розмежуванням прав доступу між викладачем і студентом.

Для досягнення поставленої мети вирішувалися такі завдання:

1) проаналізувати предметну галузь та існуючі програмні рішення для автоматизованого тестування;

2) визначити функціональні та нефункціональні вимоги до системи;

3) обґрунтувати вибір інструментів і технологій розробки;

4) спроектувати архітектуру системи та структуру бази даних;

5) реалізувати модулі автентифікації, управління тестами, проходження тестів та перегляду результатів;

6) провести тестування системи та оцінити якість розробленого програмного забезпечення.

Об'єкт дослідження — процес автоматизованого контролю та оцінювання знань студентів у навчальному закладі.

Предмет дослідження — методи, засоби та технології розробки настільного програмного забезпечення для інтерактивного тестування знань студентів.

У роботі використовувались такі методи: аналіз та синтез — для дослідження існуючих систем тестування та визначення вимог до власної розробки; структурне та об'єктно-орієнтоване проектування — для побудови архітектурних діаграм і моделі даних; шаблон проектування MVVM (Model–View–ViewModel) — для організації коду клієнтської частини; методи модульного та функціонального тестування — для перевірки коректності реалізованих компонентів.

Практична цінність роботи полягає в розробці готового до впровадження програмного продукту, який може використовуватися у навчальних закладах для проведення поточного та підсумкового контролю знань. Система підтримує автономний режим роботи (без підключення до мережі Інтернет), зберігає всі дані локально, забезпечує формування детальної звітності та дозволяє гнучко налаштовувати тести під потреби конкретної дисципліни.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі проведено аналіз предметної галузі, здійснено огляд та порівняльний аналіз існуючих систем автоматизованого тестування, сформульовано вимоги до розроблюваної системи та обґрунтовано вибір технологій.

У другому розділі виконано проєктування системи: побудовано діаграми прецедентів, класів, послідовності та станів, спроєктовано структуру реляційної бази даних, розроблено макети інтерфейсу користувача.

У третьому розділі описано реалізацію програмних модулів системи: автентифікацію та авторизацію, модуль управління тестами, модуль проходження тестів із підтримкою чотирьох типів питань, модуль перегляду результатів та адміністрування.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ОГЛЯД АНАЛОГІВ

1.1 Роль автоматизованого контролю знань у сучасній освіті

Контроль знань є невід'ємною складовою навчального процесу, оскільки дозволяє визначити рівень засвоєння матеріалу, виявити прогалини у знаннях студентів та скоригувати подальшу навчальну діяльність. Традиційно перевірка знань здійснювалась у формі усного опитування, письмових контрольних робіт або паперових тестів, однак стрімкий розвиток інформаційних технологій відкрив нові можливості для автоматизації цього процесу.

Автоматизований контроль знань — це форма перевірки рівня навчальних досягнень, що реалізується за допомогою комп'ютерних програм або веб-платформ і передбачає автоматичне формування, пред'явлення та перевірку завдань без безпосередньої участі викладача у процесі оцінювання. Такий підхід забезпечує об'єктивність, відтворюваність і масштабованість контролю.

Порівняльний аналіз традиційних та автоматизованих форм контролю знань наведено у таблиці 1.1.

Таблиця 1.1 — Порівняльна характеристика форм контролю знань

Форма контролю	Переваги	Недоліки
Усне опитування	Можливість уточнення відповіді, оцінка логіки міркувань	Суб'єктивність, значні витрати часу, охоплення малої кількості студентів
Письмова контрольна робота	Фіксація результатів, однакові умови для всіх	Тривала перевірка, ризик списування, складність зберігання
Паперовий тест	Швидка перевірка закритих питань, стандартизація	Ручний підрахунок балів, відсутність аналітики, витрати на друк
Комп'ютерне тестування	Автоматична перевірка, миттєвий результат, статистика	Потребує технічної бази, ризик технічних збоїв
Онлайн-тестування	Доступ з будь-якого пристрою, масштабованість	Залежність від Інтернету, складність контролю чесності

Як видно з таблиці 1.1, автоматизоване тестування поєднує переваги стандартизованих форм контролю — однакові умови, фіксація результатів — з суттєво скороченим часом перевірки та можливістю ведення статистики в автоматичному режимі. Серед основних переваг комп'ютерного тестування перед паперовими формами контролю слід виокремити такі:

- миттєве виставлення оцінки після завершення тесту;
- об'єктивність — результат не залежить від особистого ставлення викладача;
- автоматичне накопичення статистики успішності групи та окремих студентів;
- можливість багаторазового повторення тесту для самопідготовки;
- економія витрат на друк та зберігання паперових матеріалів;
- гнучкість налаштування: ліміт часу, кількість спроб, порядок питань.

Разом з тим автоматизоване тестування не позбавлене недоліків. Зокрема, складно перевірити творчі, дискусійні чи аналітичні завдання, що потребують розгорнутої відповіді. Крім того, існує ризик випадкового відгадування правильної відповіді в питаннях закритого типу. Тому комп'ютерне тестування найбільш ефективне при перевірці фактологічних знань, алгоритмічних умінь та репродуктивного рівня засвоєння матеріалу, а не для оцінки творчого мислення.

Широкого застосування комп'ютерне тестування набуло у системі вищої освіти України в контексті впровадження Зовнішнього незалежного оцінювання (ЗНО / НМТ), дистанційного навчання під час пандемії COVID-19, а також у рамках реалізації концепції змішаного навчання (blended learning). Ці тенденції зумовлюють зростаючий попит на якісне програмне забезпечення для тестування, зокрема для автономного використання у навчальних закладах.

1.2 Огляд існуючих систем автоматизованого тестування

На сьогоднішній день на ринку програмного забезпечення представлено широкий спектр систем для автоматизованого тестування та оцінювання

знань. Для проведення об'єктивного аналізу було обрано чотири найбільш поширені у закладах вищої освіти платформи: Moodle, Google Forms, Kahoot! та iSpring Suite.

Moodle (Modular Object-Oriented Dynamic Learning Environment) — відкрита система управління навчанням (LMS), розроблена у 2002 році Мартіном Дугіамасом. Являє собою веб-застосунок, що розгортається на власному або хмарному сервері навчального закладу. Підтримує широкий спектр видів навчальної діяльності, зокрема сім типів тестових питань, форуми, вікі та завдання. Moodle використовується більш ніж у 240 країнах і налічує понад 300 мільйонів користувачів, що робить її найпоширенішою LMS у світі.

Google Forms — безкоштовний веб-інструмент для створення опитувань і тестів у складі Google Workspace. Відрізняється простотою у використанні, не потребує встановлення програмного забезпечення та забезпечує автоматичне збереження результатів у Google Sheets. Основні обмеження: лише три типи питань для тестування, відсутність ліміту часу та мінімальний набір засобів аналітики. Потребує постійного інтернет-з'єднання.

Kahoot! — ігрова платформа для опитувань і тестів, орієнтована насамперед на інтерактивні заняття в аудиторії. Студенти приєднуються до сесії через QR-код або PIN-код і відповідають на питання у режимі реального часу. Платформа підтримує елемент гейміфікації та добре мотивує аудиторію, однак має суттєві обмеження: питання з обмеженим часом відповіді, ігровий формат не завжди підходить для підсумкового контролю, хмарне розгортання.

iSpring Suite — комерційний комплект інструментів для розробки електронних курсів і тестів, що інтегрується у Microsoft PowerPoint. Підтримує 14 типів питань, сертифікати, гілкові сценарії та вихідні формати SCORM/xAPI для публікації у LMS. Є одним із найфункціональніших рішень, однак потребує придбання ліцензії (від 770 USD/рік) та встановлення Microsoft Office.

Основні характеристики розглянутих систем зведено до таблиці 1.2.

Таблиця 1.2 — Характеристики систем автоматизованого тестування

Критерій	Moodle	Google Forms	Kahoot!	iSpring Suite
Тип розгортання	Сервер / хмара	Хмара (Google)	Хмара	Десктоп + хмара
Автономна робота	Так (сервер)	Ні	Ні	Частково
Типи питань	7 типів	3 типи	5 типів	14 типів
Ролі користувачів	Гнучкі	Мінімальні	Ведучий / гравець	Автор / студент
Таймер тесту	Так	Ні	На питання	Так
Статистика	Розширена	Базова	Базова	Розширена
Імпорт / Експорт	GIFT, XML	Google Sheets	Kahoot-формат	SCORM, xAPI
Вартість	Безкоштовно (OSS)	Безкоштовно	Freemium	Платна
Складність налаштування	Висока	Низька	Низька	Середня

1.3 Порівняльний аналіз та обґрунтування розробки власної системи

Проведений огляд виявив, що кожна з розглянутих систем має суттєві обмеження, що перешкоджають їх ефективному використанню у невеликих навчальних закладах або на окремих кафедрах без постійного доступу до мережі Інтернет. Зокрема:

– Moodle потребує адміністрування серверної інфраструктури та спеціальних технічних знань для налаштування, що є суттєвим бар'єром для більшості освітніх установ;
– Google Forms цілком залежить від хмарної інфраструктури Google та не забезпечує належної аналітики результатів;
– Kahoot! орієнтований на ігровий формат і не підходить для серйозного підсумкового оцінювання;
– iSpring Suite є платним продуктом з обов'язковою залежністю від Microsoft Office.

З метою кількісного порівняння систем за ключовими критеріями проведено бальну оцінку за п'ятибальною шкалою. Результати оцінювання наведено у таблиці 1.3, де 1 — найгірший результат, 5 — найкращий.

Таблиця 1.3 — Порівняльна бальна оцінка систем тестування (1–5 балів)

Критерій	Moodle	Google Forms	Kahoot!	iSpring	Розробл. система
Автономна робота	3	1	1	3	**5**
Простота встановлення	2	5	4	3	**5**
Різноманітність типів питань	4	2	3	5	**4**
Розмежування ролей	5	2	2	3	**4**
Аналітика результатів	5	3	2	5	**4**
Безкоштовність	5	5	3	1	**5**
Імпорт / Експорт тестів	4	2	2	5	**4**
Середній бал (з 5)	**4,0**	**2,9**	**2,4**	**3,6**	**4,4**

Аналіз таблиці 1.3 свідчить, що розроблювана система демонструє найвищий середній бал — 4,4 з 5,0. Найвагомішою перевагою є повна автономна робота без Інтернет-з'єднання та простота встановлення — достатньо скопіювати директорію з виконуваним файлом. Єдина поступка конкурентам — кількість типів питань (4 проти 14 у iSpring), однак чотири реалізованих типи охоплюють найбільш затребувані педагогічні сценарії.

Таким чином, розробка власної настільної системи тестування є обґрунтованою з огляду на потребу в автономному, простому у встановленні та безкоштовному інструменті для поточного й підсумкового контролю знань студентів.

1.4 Формування вимог до системи

На основі аналізу предметної галузі та виявлених недоліків існуючих рішень сформульовано перелік вимог до розроблюваної системи. Вимоги поділяються на функціональні (визначають, що система повинна робити) та нефункціональні (визначають, як система повинна це робити).

Функціональні вимоги описують конкретні дії та операції, які повинна підтримувати система. Їх перелік із зазначенням пріоритету наведено у таблиці 1.4.

Таблиця 1.4 — Функціональні вимоги до системи тестування

№	Вимога	Пріоритет
1	Реєстрація та автентифікація користувачів з розмежуванням ролей «Викладач» і «Студент»	Обов'язкова
2	Створення, редагування та видалення тестів викладачем	Обов'язкова
3	Підтримка чотирьох типів питань: одна правильна відповідь, кілька правильних відповідей, відкрита відповідь, встановлення відповідності	Обов'язкова
4	Встановлення ліміту часу на виконання тесту	Обов'язкова
5	Автоматичне оцінювання результатів після завершення тесту	Обов'язкова
6	Перегляд студентом власних результатів з детальним розбором відповідей	Обов'язкова
7	Перегляд викладачем зведеної статистики за тестом з графічним відображенням	Обов'язкова
8	Управління обліковими записами студентів адміністратором	Бажана
9	Імпорт тестів з файлів формату JSON та XML	Бажана
10	Експорт результатів у форматі CSV	Бажана
11	Переключення між світлою та темною темою інтерфейсу	Додаткова

Окрім функціональних, система повинна відповідати низці нефункціональних вимог, що характеризують якісні показники її роботи. Нефункціональні вимоги та відповідні метрики наведено у таблиці 1.5.

Таблиця 1.5 — Нефункціональні вимоги до системи тестування

Категорія	Вимога	Показник / метрика
Продуктивність	Час запуску застосунку не перевищує допустимого порогу	≤ 3 секунди на сучасному ПК
Відгук інтерфейсу	Реакція на дію користувача без помітної затримки	≤ 200 мс для навігаційних операцій
Надійність	Збереження цілісності даних при аварійному завершенні	Транзакційне збереження через EF Core
Безпека	Зберігання паролів у захищеному вигляді	BCrypt, work factor 11
Зручність використання	Інтуїтивний інтерфейс без спеціального навчання	Освоєння за ≤ 15 хвилин нового користувача
Переносимість	Робота на ОС Windows без встановлення сторонніх залежностей	Windows 10 / 11, .NET 9 Runtime
Супроводжуваність	Покриття коду модульними тестами	≥ 70% для сервісного рівня

Особливу увагу приділено вимогам до безпеки. Усі паролі користувачів зберігаються виключно у хешованому вигляді із застосуванням алгоритму BCrypt, що унеможливлює їх відновлення у разі несанкціонованого доступу до файлу бази даних. Розмежування прав доступу між роллю «Викладач» і роллю «Студент» реалізовано на рівні бізнес-логіки застосунку.

1.5 Обґрунтування вибору технологій розробки

Вибір технологічного стеку для реалізації системи ґрунтувався на аналізі вимог, визначених у попередньому підрозділі, а також на критеріях доступності, зрілості та якості документації. Обрані технології та обґрунтування їх вибору представлено у таблиці 1.6.

Таблиця 1.6 — Технологічний стек та обґрунтування вибору

Компонент	Обрана технологія	Обґрунтування вибору
Мова програмування	C# 13 (.NET 9)	Сучасна типобезпечна мова з багатою стандартною бібліотекою, підтримкою асинхронного програмування та активною спільнотою
UI-фреймворк	WPF (Windows Presentation Foundation)	Зрілий фреймворк для десктопних застосунків Windows з потужною системою прив'язки даних та XAML-розміткою
Архітектурний шаблон	MVVM	Чітке розмежування відповідальності між рівнями подання, логіки та даних; полегшує тестування та підтримку коду
СУБД	SQLite 3	Вбудована реляційна база даних без потреби в окремому сервері; ідеальна для автономних настільних застосунків
ORM	Entity Framework Core 9	Забезпечує роботу з базою даних через об'єктну модель C#, спрощує міграції та усуває SQL-ін'єкції
Хешування паролів	BCrypt.Net-Next	Алгоритм адаптивного хешування BCrypt з налаштованим коефіцієнтом складності забезпечує стійкість до brute-force атак
Графіки	OxyPlot 2.1	Крос-платформна бібліотека побудови діаграм з підтримкою WPF; забезпечує інтерактивні кругові та стовпчасті діаграми
Логування	Serilog	Структуроване логування подій у файл із ротацією; полегшує діагностику помилок у production-середовищі
Тестування	xUnit 2.9	Сучасний фреймворк модульного тестування .NET з підтримкою параметризованих тестів та паралельного виконання

Центральним архітектурним рішенням є застосування шаблону проектування MVVM (Model–View–ViewModel). Цей шаблон забезпечує чітке розмежування між трьома рівнями застосунку:

– Model — рівень даних і бізнес-логіки (класи сутностей, сервіси, контекст бази даних);
– View — рівень подання (XAML-розмітка вікон та елементів керування);

– ViewModel — рівень логіки подання (команди, прив'язки даних, стан UI).

Перевагою MVVM у поєднанні з WPF є потужний механізм прив'язки даних (Data Binding), який автоматично синхронізує стан UI з об'єктами ViewModel. Це дозволяє розробнику зосередитися на логіці, не реалізуючи вручну оновлення інтерфейсу. Реалізацію сповіщень про зміни забезпечує інтерфейс INotifyPropertyChanged через базовий клас BaseViewModel.

Вибір SQLite як системи управління базами даних зумовлений кількома чинниками. По-перше, SQLite не потребує окремого серверного процесу — вся база даних зберігається в одному файлі (app.db) поруч із виконуваним файлом застосунку. По-друге, SQLite підтримує повний набір операцій реляційної бази даних, включаючи транзакції, зовнішні ключі та індекси, що забезпечує цілісність даних. По-третє, продуктивність SQLite цілком достатня для навантажень рівня навчального закладу (сотні користувачів, тисячі результатів тестів).

Entity Framework Core 9 використовується як об'єктно-реляційний відображувач (ORM), що дозволяє описувати структуру бази даних у вигляді звичайних класів C# (Code First) та виконувати запити мовою LINQ замість сирого SQL. Це суттєво підвищує безпеку (захист від SQL-ін'єкцій) та зручність розробки.

Для побудови графічних звітів обрано бібліотеку OxyPlot, яка є відкритою, активно підтримується спільнотою та забезпечує рендеринг кругових (PieSeries) і стовпчастих (BarSeries) діаграм безпосередньо у WPF-контролах без залежності від зовнішніх компонентів.

Висновки до розділу 1

У першому розділі проведено всебічний аналіз предметної галузі автоматизованого контролю знань студентів. Встановлено, що традиційні форми перевірки знань мають суттєві обмеження щодо об'єктивності та масштабованості, тоді як комп'ютерне тестування забезпечує автоматичну перевірку, миттєвий результат і ведення статистики.

Здійснено огляд та порівняльний аналіз чотирьох популярних систем тестування: Moodle, Google Forms, Kahoot! та iSpring Suite. Виявлено, що жодна з розглянутих систем не відповідає повністю сформульованому набору вимог: Moodle і Google Forms потребують серверної інфраструктури або Інтернету, Kahoot! орієнтований на ігровий формат, iSpring є платним рішенням.

За результатами бальної оцінки за сімома критеріями розроблювана система отримала найвищий середній бал — 4,4 з 5,0, що підтверджує доцільність її створення.

Сформульовано одинадцять функціональних та сім нефункціональних вимог до системи. Обґрунтовано вибір технологічного стеку: C# 13 / .NET 9, WPF, MVVM, SQLite, Entity Framework Core 9, BCrypt, OxyPlot, Serilog та xUnit. Визначені технології забезпечують автономну роботу, безпеку, достатню продуктивність та зручність підтримки системи.

РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ

2.1 Архітектурне рішення на основі шаблону MVVM

Вибір архітектурного шаблону є одним із ключових рішень при проектуванні програмної системи, оскільки він визначає спосіб організації коду, розподіл відповідальності між компонентами та зручність подальшого супроводу. Для реалізації системи тестування обрано шаблон MVVM (Model–View–ViewModel), який є рекомендованим підходом для WPF-застосунків та органічно підтримується механізмом прив'язки даних цього фреймворку.

MVVM поділяє застосунок на три чітко розмежовані рівні:

– Model (Модель) — містить класи сутностей (User, Test, Question, Answer, Result), сервіси бізнес-логіки (AuthService, TestService, ResultService) та контекст бази даних (AppDbContext). Рівень моделі не залежить від інтерфейсу й може тестуватись незалежно від UI.

– View (Подання) — XAML-розмітка вікон і елементів керування, яка описує зовнішній вигляд інтерфейсу. View не містить логіки — усі дії делегуються через прив'язки та команди до ViewModel.

– ViewModel (Модель подання) — клас-посередник, який надає View властивості (properties) для відображення та команди (ICommand) для обробки дій користувача. ViewModel не має прямого посилання на View і взаємодіє з нею виключно через механізм Data Binding.

Схема архітектури MVVM у розробленій системі наведена на рисунку 2.1.

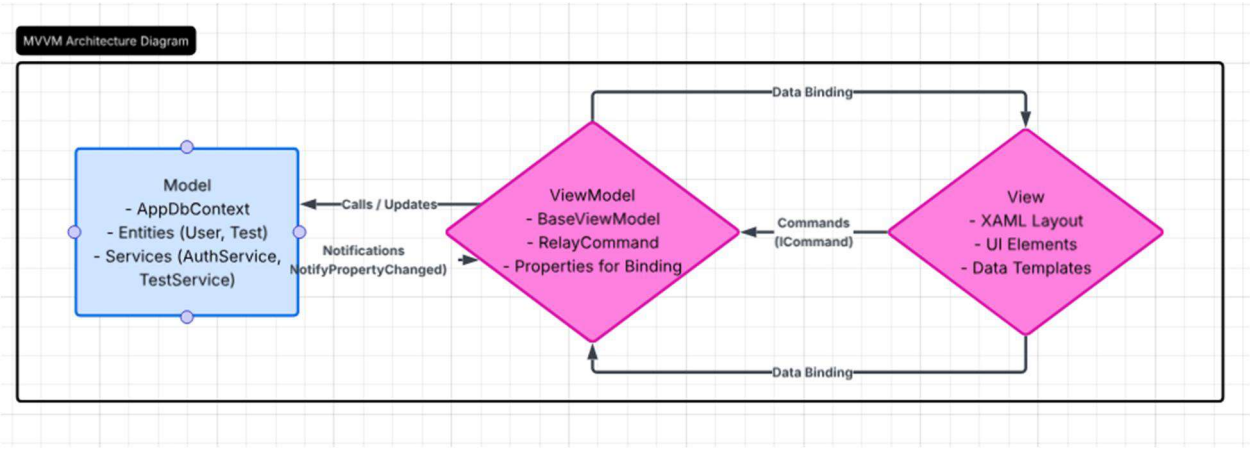


Рисунок 2.1.1. – Архітектура системи за шаблоном MVVM

Центральним елементом взаємодії між View та ViewModel є механізм прив'язки даних WPF (Data Binding). Коли властивість ViewModel змінюється, вона повідомляє про це View через інтерфейс INotifyPropertyChanged.

Для уніфікації реалізації цього механізму всі ViewModel-класи успадковують від абстрактного класу BaseViewModel, який інкапсулює метод SetProperty<T>: поле ref field зберігає поточне значення, value — нове значення, а виклик OnPropertyChanged(propertyName) ініціює оновлення відповідного елемента UI. Така реалізація усуває необхідність дублювання коду у кожній властивості кожного ViewModel-класу.

Для обробки команд (натискань кнопок, вибору елементів меню тощо) використовується клас RelayCommand, що реалізує інтерфейс ICommand. Завдяки цьому логіка дій повністю зосереджена у ViewModel і не розповсюджується у code-behind файлах View. Для асинхронних операцій (завантаження даних з бази, збереження результатів) застосовується AsyncRelayCommand, що запускає Task без блокування UI-потоку.

Взаємодія між ViewModel-класами та сервісами організована через шаблон Locator (ServiceLocator): статичний клас ServiceLocator зберігає зареєстровані екземпляри сервісів і надає їх за типом інтерфейсу. Це дозволяє розв'язати залежності між компонентами без складного DI-контейнера.

2.2 Діаграма прецедентів (Use Case)

Діаграма прецедентів описує зовнішню поведінку системи з точки зору акторів (користувачів), що взаємодіють з нею. У розробленій системі виділено двох основних акторів: Студент та Викладач. Окремим суб-актором є адміністративна роль Викладача — управління обліковими записами студентів.

Діаграма прецедентів системи наведена на рисунку 2.2.

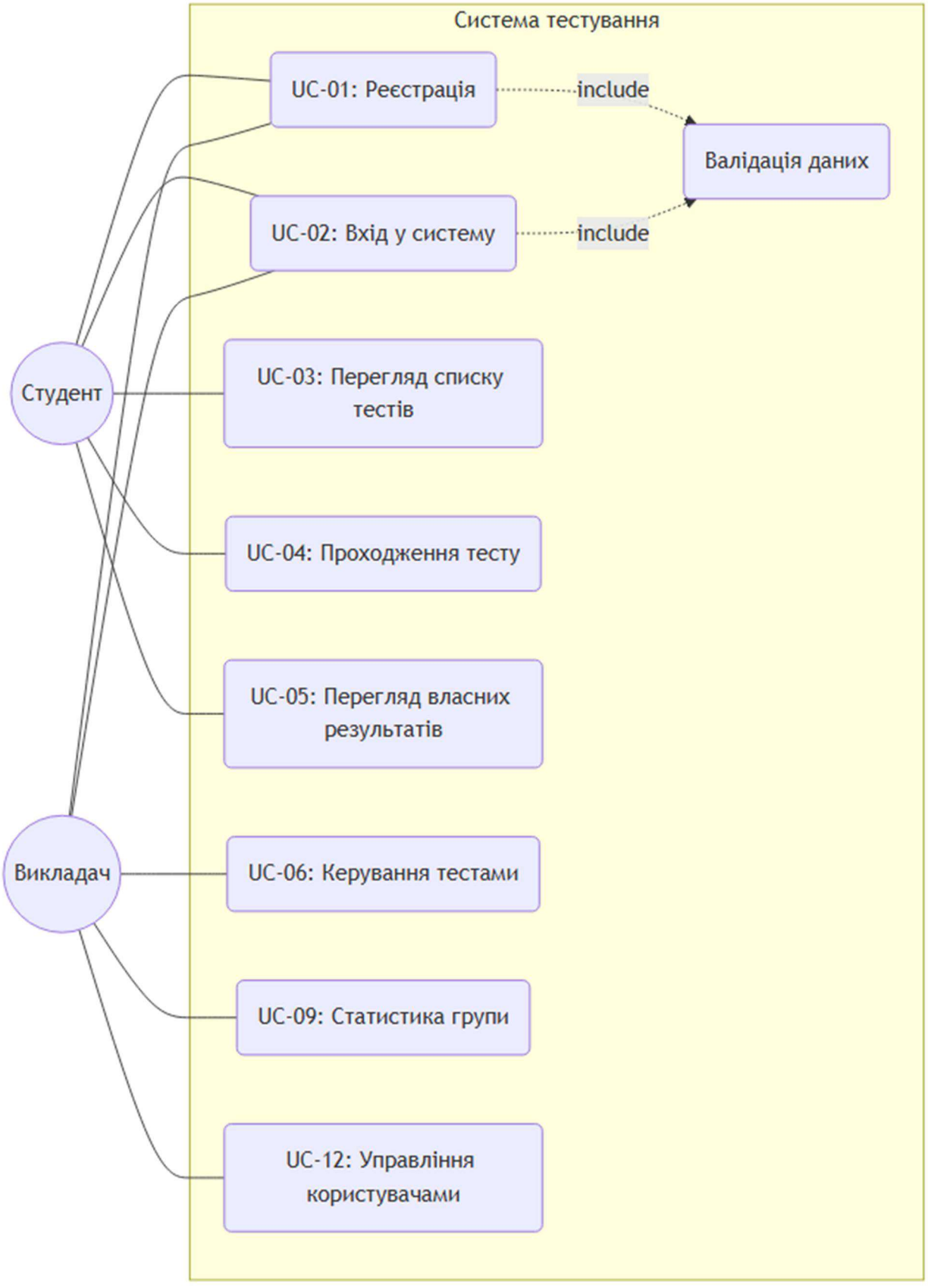


Рисунок 2.2.1. –Діаграма прецедентів системи тестування

Актор «Студент» має доступ до таких прецедентів:

– UC-01: Реєстрація в системі — введення даних, вибір ролі, хешування пароля;
– UC-02: Вхід до системи — автентифікація за логіном та паролем;
– UC-03: Перегляд списку доступних тестів;
– UC-04: Проходження тесту — навігація між питаннями, відповіді, таймер;
– UC-05: Перегляд власних результатів — бал, оцінка, деталі відповідей.

Актор «Викладач» включає всі прецеденти Студента (крім UC-04) та додатково:

– UC-06: Створення нового тесту з питаннями та відповідями;
– UC-07: Редагування існуючого тесту (назва, питання, відповіді, ліміт часу);
– UC-08: Видалення тесту;
– UC-09: Перегляд зведеної статистики результатів студентів за тестом;
– UC-10: Імпорт тесту з файлу JSON або XML;
– UC-11: Експорт результатів студентів у файл CSV;
– UC-12: Управління обліковими записами студентів (редагування, деактивація).

Між прецедентами UC-01/UC-02 існує відношення «include» до прецеденту «Валідація даних», який описує перевірку коректності введених форматів та відображення повідомлень про помилки. Прецеденти UC-06 та UC-07 включають «Управління питаннями» як вкладений прецедент.

2.3 Проектування бази даних. ER-діаграма

Проектування структури бази даних є критично важливим етапом, оскільки від неї залежать цілісність даних, продуктивність запитів та зручність розширення системи. Реляційна модель даних розробленої системи включає шість таблиць.

Перелік таблиць та їх призначення наведено у таблиці 2.1.

Таблиця 2.1 — Перелік таблиць бази даних системи тестування

Таблиця	Призначення	Ключові поля
Users	Облікові записи користувачів системи	Id, Username, PasswordHash, Role, IsActive
Tests	Тести, що створені викладачами	Id, Title, TimeLimitMinutes, IsActive, CreatedByUserId
Questions	Питання тесту	Id, TestId, Text, Type, Points, OrderIndex
Answers	Варіанти відповідей на питання	Id, QuestionId, Text, IsCorrect, MatchTarget, OrderIndex
Results	Підсумкові результати проходження тесту	Id, TestId, StudentId, Score, MaxScore, Percentage, Grade, DateCompleted
ResultDetails	Детальні відповіді по кожному питанню	Id, ResultId, QuestionId, GivenAnswer, IsCorrect, PointsEarned

ER-діаграма (Entity-Relationship), що відображає сутності та зв'язки між ними, наведена на рисунку 2.3.

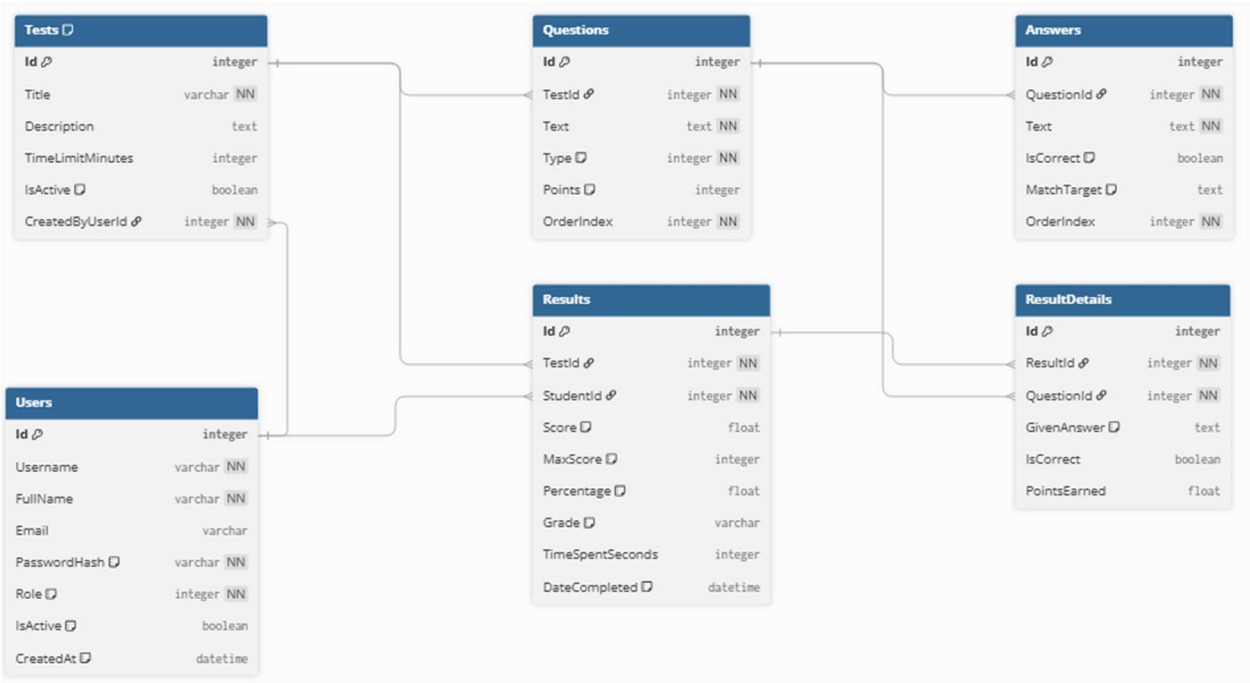


Рисунок 2.3.1. – ER-діаграма бази даних системи тестування

Детальна структура ключової таблиці Users наведена у таблиці 2.2.

Таблиця 2.2 — Структура таблиці Users

Поле	Тип даних	NULL	PK/FK	Опис
Id	INTEGER	Ні	PK	Унікальний ідентифікатор
Username	TEXT	Ні	—	Ім'я користувача для входу (унікальне)
FullName	TEXT	Ні	—	Повне ім'я (прізвище та ініціали)
Email	TEXT	Так	—	Електронна адреса
PasswordHash	TEXT	Ні	—	Хеш пароля алгоритмом BCrypt
Role	INTEGER	Ні	—	Роль: 0 — Студент, 1 — Викладач
IsActive	INTEGER	Ні	—	Активність: 1 — активний, 0 — заблокований
CreatedAt	TEXT	Ні	—	Дата та час реєстрації (ISO 8601)

Структура таблиці Questions, що описує питання тесту, наведена у таблиці 2.3.

Таблиця 2.3 — Структура таблиці Questions

Поле	Тип даних	NULL	PK/FK	Опис
Id	INTEGER	Ні	PK	Унікальний ідентифікатор питання
TestId	INTEGER	Ні	FK	Посилання на тест (Tests.Id)
Text	TEXT	Ні	—	Текст питання
Type	INTEGER	Ні	—	Тип: 0 — одна відповідь, 1 — кілька, 2 — відкрита, 3 — відповідність
Points	INTEGER	Ні	—	Кількість балів за правильну відповідь
OrderIndex	INTEGER	Ні	—	Порядковий номер питання у тесті

Таблиця Answers містить поля: Id (PK), QuestionId (FK), Text, IsCorrect (INTEGER 0/1), MatchTarget (TEXT, NULL для не-Matching питань) та OrderIndex.

Таблиця Results зберігає: Id, TestId (FK), StudentId (FK), Score, MaxScore, Percentage (REAL), Grade (TEXT), TimeSpentSeconds (INTEGER) та DateCompleted (TEXT у форматі ISO 8601).

Таблиця ResultDetails містить: Id, ResultId (FK), QuestionId (FK), GivenAnswer (TEXT) та PointsEarned (REAL).

Зв'язки між таблицями та їх тип наведено у таблиці 2.4.

Таблиця 2.4 — Зв'язки між таблицями бази даних

Таблиця (батьківська)	Таблиця (дочірня)	Тип зв'язку	Опис
Users	Tests	1 : M	Один викладач створює багато тестів
Tests	Questions	1 : M	Один тест містить багато питань
Questions	Answers	1 : M	Одне питання має багато варіантів відповіді
Tests	Results	1 : M	Один тест має багато результатів проходження
Users	Results	1 : M	Один студент має багато результатів
Results	ResultDetails	1 : M	Один результат містить деталі по кожному питанню

Усі зовнішні ключі налаштовані з каскадним видаленням (CASCADE DELETE): при видаленні тесту автоматично видаляються всі пов'язані питання, відповіді та результати. Цілісність даних забезпечується на рівні SQLite через директиву PRAGMA foreign_keys = ON, яка активується при ініціалізації контексту бази даних.

Для прискорення пошукових запитів передбачено такі індекси:

– індекс на поле Users.Username — забезпечує O(log n) пошук при автентифікації;
– складений індекс на Results(TestId, StudentId) — прискорює запити статистики;
– індекс на Questions.TestId та Answers.QuestionId — оптимізує завантаження тесту.

2.4 Діаграма класів

Діаграма класів описує статичну структуру програмної системи: класи, їх атрибути, методи та відносини між ними. Вона є основою для реалізації коду та відображає архітектурне рішення MVVM у термінах об'єктно-орієнтованого проектування.

Систему поділено на чотири пакети: Models (сутності даних), Services (бізнес-логіка), ViewModels (логіка подання) та Converters (перетворювачі для прив'язок WPF). Діаграма класів наведена на рисунку 2.4.

Перелік основних ViewModel-класів із зазначенням пов'язаних вікон та обов'язків наведено у таблиці 2.5.

Таблиця 2.5 — Перелік ViewModel-класів та їх відповідальність

Клас ViewModel	Пов'язане вікно / View	Відповідальність
LoginViewModel	LoginWindow	Автентифікація: перевірка логіна та пароля, збереження сесії
RegisterViewModel	RegisterWindow	Реєстрація нового користувача з валідацією полів
MainViewModel	MainWindow	Головне вікно: навігація між розділами, управління темою
TeacherDashboardViewModel	TeacherDashboardView	Панель викладача: список тестів, команди створення та редагування
StudentDashboardViewModel	StudentDashboardView	Панель студента: список доступних тестів, останні результати
TestEditorViewModel	TestEditorWindow	Редактор тесту та питань: CRUD-операції, управління типами питань
TestTakingViewModel	TestTakingWindow	Проходження тесту: навігація, таймер, збереження відповідей
ResultsViewModel	TestResultsWindow	Перегляд результатів: статистика, кругова діаграма, деталі
UsersViewModel	UsersView	Управління користувачами: список, фільтрація, редагування
EditUserViewModel	EditUserWindow	Редагування облікового запису: дані, роль, зміна пароля

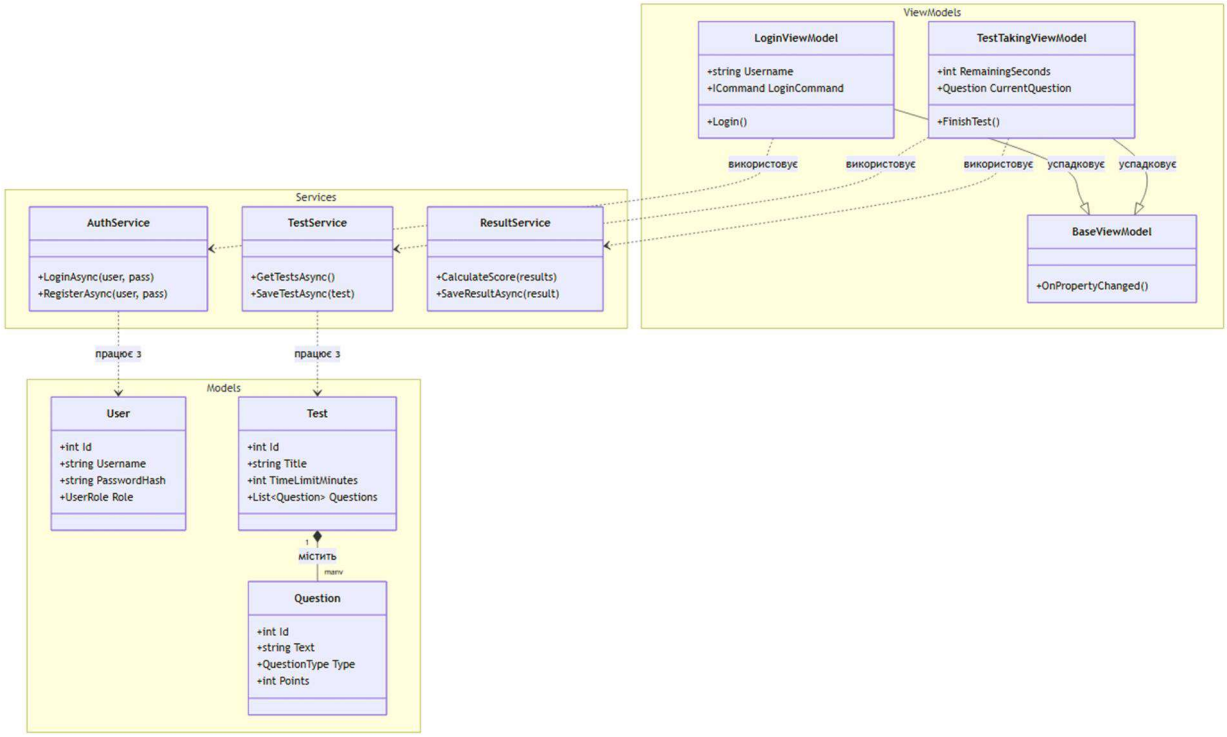


Рисунок 2.4.1. –Діаграма класів системи тестування

Перелік сервісних класів рівня Model наведено у таблиці 2.6.

Таблиця 2.6 — Перелік сервісних класів рівня Model

Сервіс / клас	Відповідальність
AppDbContext	Контекст Entity Framework Core; визначає DbSet-колекції для кожної таблиці та налаштування зв'язків через Fluent API
DatabaseInitializer	Ініціалізація бази даних при першому запуску: EnsureCreated, заповнення тестових даних (seed)
AuthService	Реєстрація та автентифікація: хешування пароля BCrypt, перевірка облікових даних, зберігання поточного користувача
TestService	CRUD-операції над тестами та питаннями; завантаження тесту з питаннями та відповідями для проходження
ResultService	Збереження результату тесту, обчислення балів та оцінки, завантаження статистики за тестом
UserService	Управління обліковими записами: перелік, пошук, оновлення, зміна пароля
ImportExportService	Серіалізація / десеріалізація тестів у форматах JSON та XML; експорт результатів у CSV
ThemeManager	Перемикання між світлою та темною темою через зміну ResourceDictionary

Усі ViewModel-класи успадковують від абстрактного класу BaseViewModel, який реалізує INotifyPropertyChanged. Усі сервісні класи реалізують відповідні інтерфейси (IAuthService, ITestService, IResultService, IUserService, IImportExportService), що забезпечує слабке зчеплення між компонентами та можливість підміни реалізації при тестуванні (Dependency Injection, Mock-об'єкти).

2.5 Діаграми послідовності основних сценаріїв

Сценарій 1: Автентифікація користувача.

Процес входу починається із введення логіна та пароля у LoginWindow. LoginViewModel викликає метод IAuthService.LoginAsync(username, password). AuthService звертається до бази даних для отримання хешу пароля, після чого BCrypt.Verify порівнює введений пароль з хешем. У разі успіху поточний користувач зберігається у сесії та відкривається головне вікно відповідної ролі. Діаграма послідовності входу наведена на рисунку 2.5.

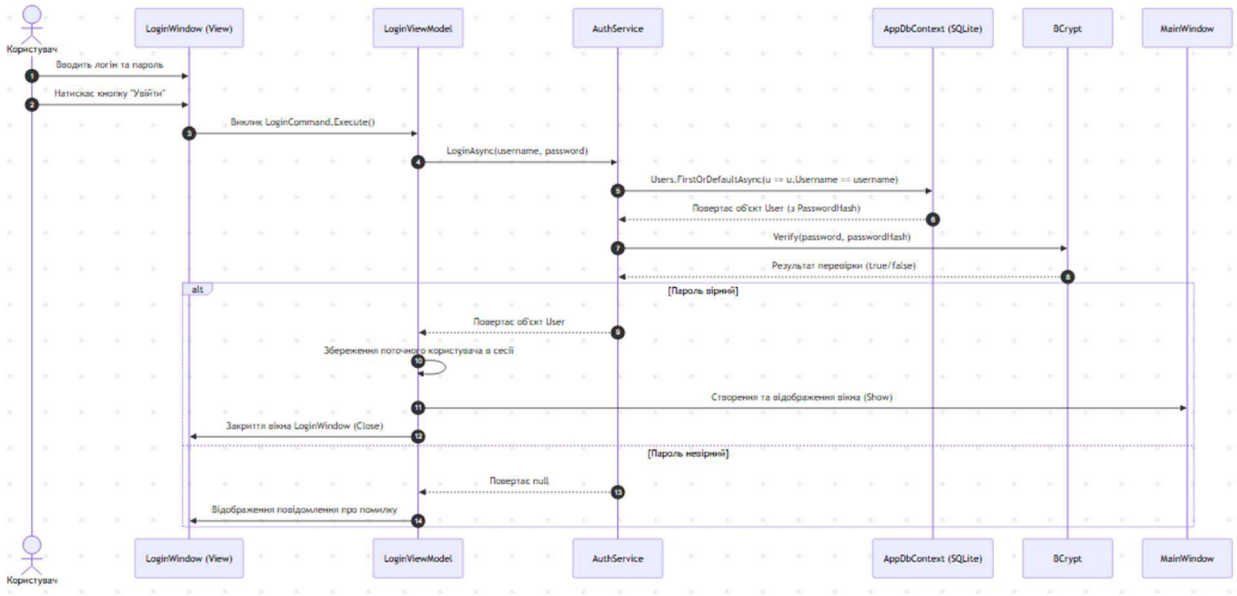


Рисунок 2.5 — Діаграма послідовності сценарію автентифікації

Сценарій 2: Проходження тесту.

Студент обирає тест у StudentDashboardView. ViewModel ініціює завантаження тесту через TestService.GetTestWithQuestionsAsync(testId) — повертається об'єкт Test з колекціями питань та відповідей. Відкривається TestTakingWindow, де TestTakingViewModel керує таймером

(System.Timers.Timer), навігацією між питаннями (властивості CurrentIndex, CurrentQuestion, IsFirst, IsLast) та збором відповідей. Після натискання «Здати тест» виконується FinishAsync. Діаграма послідовності наведена на рисунку 2.6.

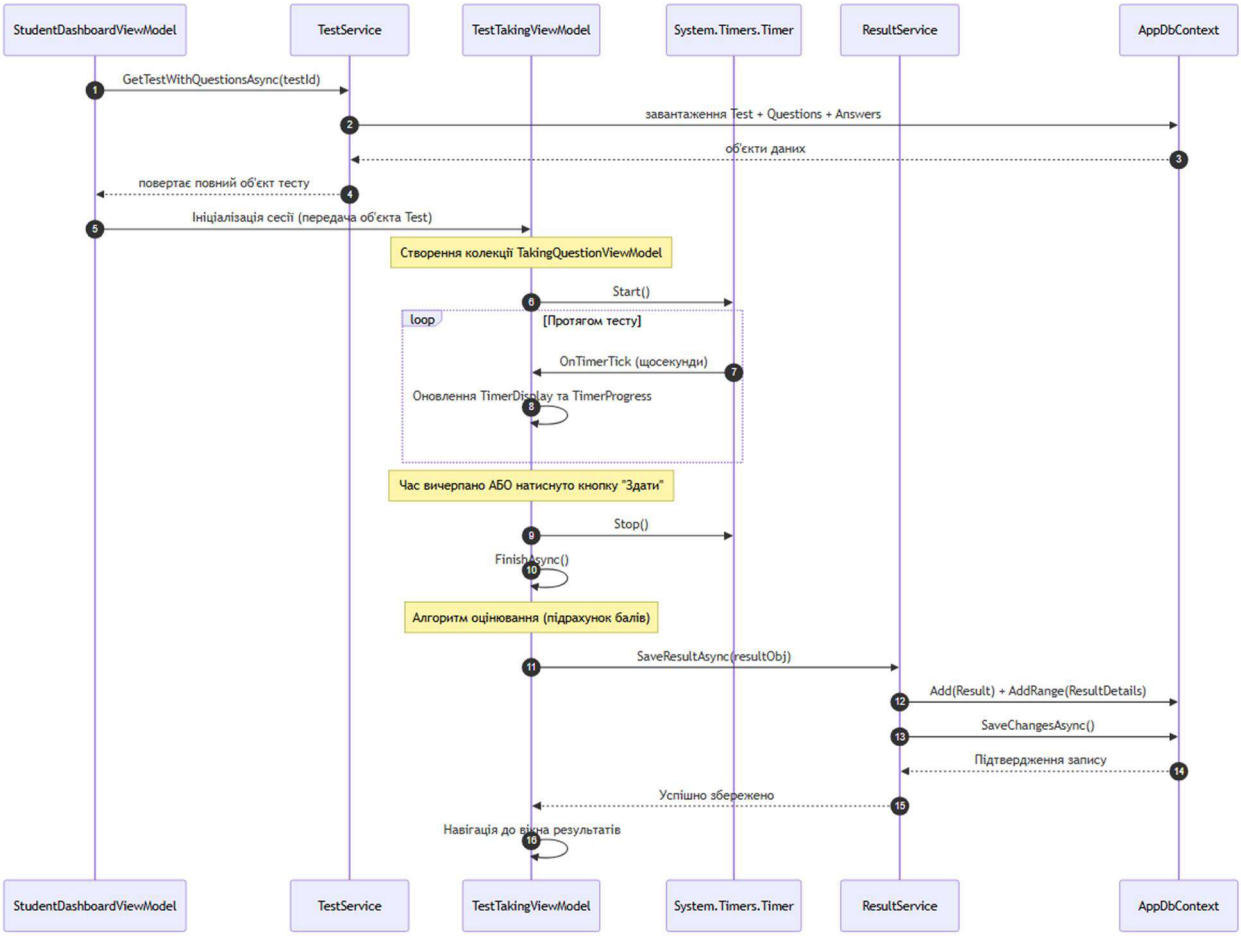


Рисунок 2.6 — Діаграма послідовності сценарію проходження тесту

Сценарій 3: Оцінювання та збереження результату.

Метод FinishAsync у TestTakingViewModel перебирає всі питання та обчислює нараховані бали відповідно до типу питання. Алгоритм підрахунку описано у підрозділі 2.7. Сформований об'єкт Result разом із колекцією ResultDetail передається до ResultService.SaveResultAsync, який зберігає дані у базі. Студент отримує екран з підсумковим результатом. Діаграма послідовності наведена на рисунку 2.7.

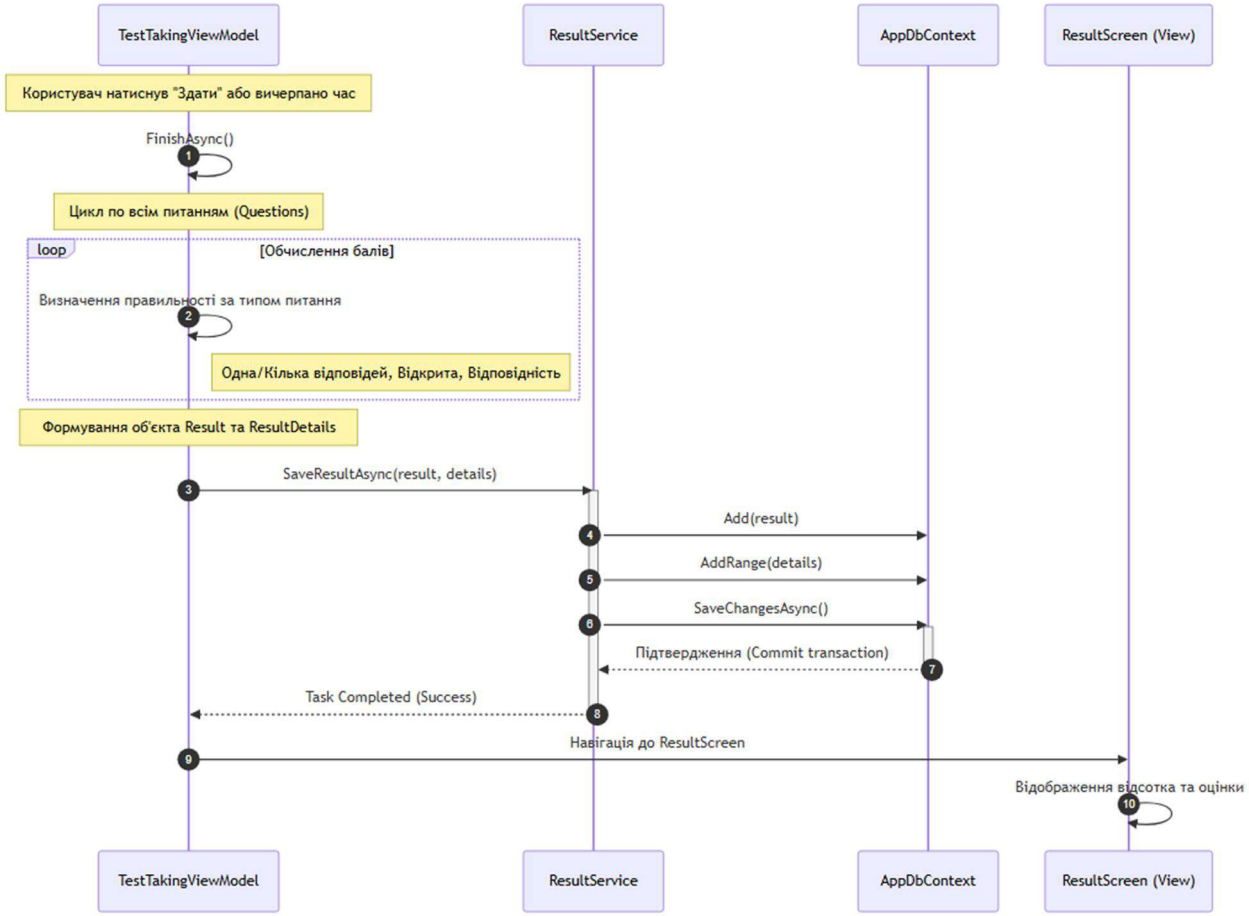


Рисунок 2.7 — Діаграма послідовності оцінювання та збереження результату

2.6 Діаграма станів тесту

Для опису поведінки об'єкта «Тест» протягом його життєвого циклу побудовано діаграму станів (State Machine Diagram). Ця діаграма моделює стан тесту як з точки зору його існування в системі, так і з точки зору поточного сеансу проходження.

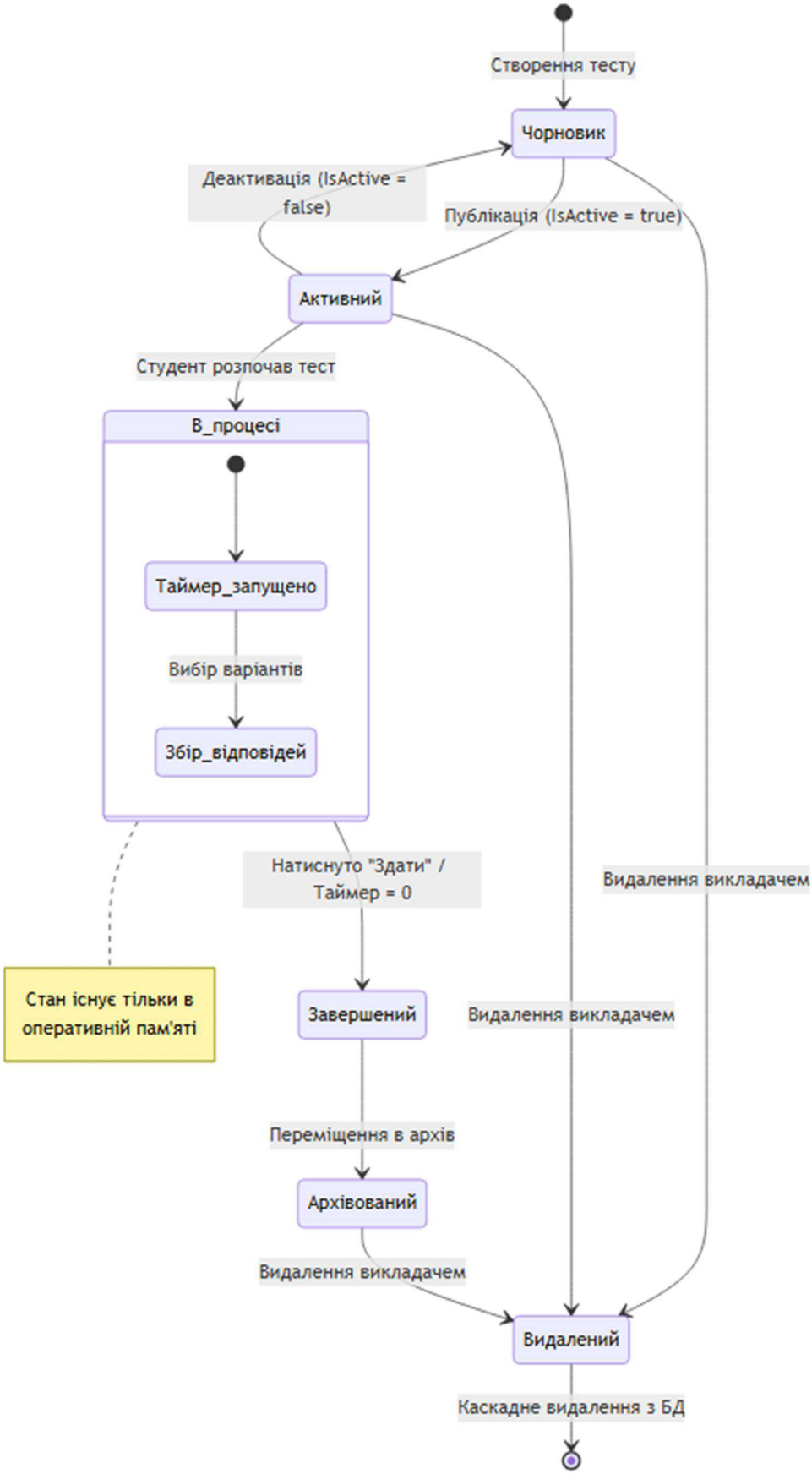


Рисунок 2.8 — Діаграма станів тесту

Тест у системі може перебувати в одному з таких станів:

– Неактивний (IsActive = false) — тест створено або вимкнено викладачем; студенти не бачать його у списку доступних;
– Активний (IsActive = true) — тест опубліковано; відображається у списку тестів для студентів;
– В процесі проходження — студент розпочав тест; таймер запущено; відповіді збираються у пам'яті;
– Завершений — студент натиснув «Здати» або минув час; результат збережено у БД;
– Видалений — тест видалено викладачем; каскадно видаляються питання, відповіді та результати.

Переходи між станами ініціюються діями викладача (зміна IsActive, видалення), студента (початок та завершення сеансу) або системи (спрацювання таймера). Стан «В процесі проходження» є тимчасовим та існує виключно в оперативній пам'яті — він не зберігається у базі даних до явного завершення тесту.

2.7 Проектування алгоритму оцінювання

Оцінювання результатів тестування є ключовою функцією системи, тому алгоритм підрахунку балів спроектовано окремо з урахуванням чотирьох типів питань. Загальна формула обчислення відсотка правильних відповідей:

$$P = (S / M) \times 100\%$$

де P — відсоток виконання (Percentage), S — сума нарахованих балів (Score), M — максимально можлива сума балів (MaxScore).

На основі значення P формується оцінка за національною шкалою:

– $P \geq 90\%$ — «Відмінно» (A);
– $75\% \leq P < 90\%$ — «Добре» (B–C);
– $60\% \leq P < 75\%$ — «Задовільно» (D–E);
– $P < 60\%$ — «Незадовільно» (F).

Нарахування балів залежно від типу питання описано у таблиці 2.7.

Таблиця 2.7 — Алгоритм нарахування балів залежно від типу питання

Тип питання	Максимум балів	Умова нарахування балів
Одна правильна відповідь	Points (ціле)	Points, якщо обрана відповідь = IsCorrect; 0 — інакше
Кілька правильних відповідей	Points (ціле)	Points, якщо множина обраних = множина IsCorrect; 0 — інакше
Відкрита відповідь	Points (ціле)	Points, якщо введений текст збігається з еталоном (без урахування регістру); 0 — інакше
Відповідність	Points (дробове)	Points × (кількість правильних пар / загальна кількість пар) — пропорційне нарахування

Для питань типу «Відповідність» застосовується пропорційне нарахування балів, що дозволяє оцінити часткове знання: якщо студент правильно встановив 3 пари з 4, він отримує 75% від максимального балу за це питання. Для питань з однією та кількома правильними відповідями нарахування балів є бінарним: або повний бал, або нуль.

Алгоритм визначення правильності відповіді на питання типу «Кілька правильних відповідей» порівнює дві множини: множину ідентифікаторів обраних відповідей та множину ідентифікаторів відповідей з IsCorrect = 1. Якщо множини рівні (обидві за включенням), бали нараховуються повністю. Часткове нарахування для цього типу не передбачено.

2.8 Проектування інтерфейсу користувача

Проектування інтерфейсу розпочалося зі створення дротяних макетів (wireframes) основних екранів системи. Wireframe — це схематичне зображення розташування елементів інтерфейсу без кольорового оформлення, що дозволяє зосередитися на структурі та функціональності, а не на естетиці.

При проектуванні дотримувались таких принципів:

– послідовність та передбачуваність — однакові дії виконуються однаковим чином у різних вікнах;
– мінімальна кількість кроків — найбільш часті операції доступні в один-два кліки;

– зворотний зв'язок — система завжди повідомляє користувача про результат дії (повідомлення про помилки, індикатори завантаження);
– розмежування інтерфейсів — панель викладача та панель студента суттєво відрізняються за набором доступних функцій.

Макет вікна входу до системи (LoginWindow) наведено на рисунку 2.9. Вікно має мінімалістичний вигляд: логотип системи, два поля введення та кнопки «Увійти» і «Зареєструватися». У нижній частині відображається повідомлення про помилку автентифікації.

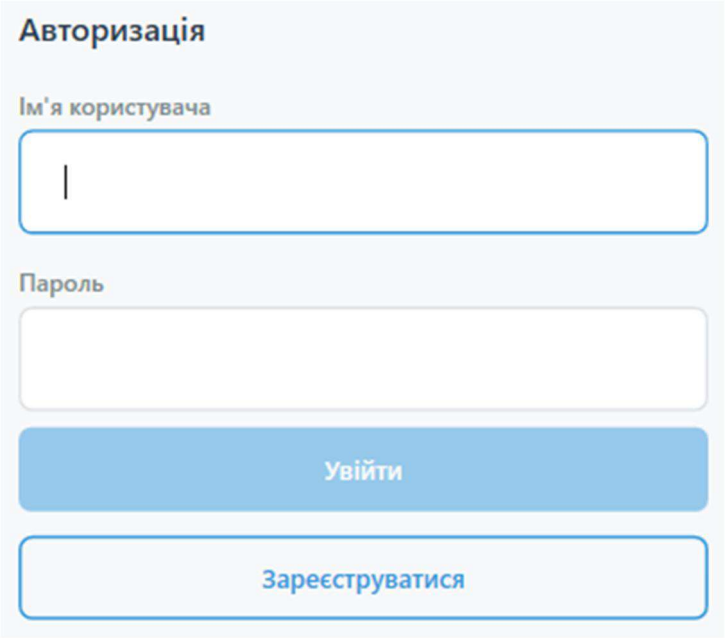


Рисунок 2.9 — Макет вікна входу до системи (LoginWindow)

Макет головного вікна панелі викладача побудовано за дволонковою схемою: ліва панель — список тестів з кнопками дій (Створити, Редагувати, Видалити, Результати, Імпорт, Експорт); права панель — деталі обраного тесту (кількість питань, дата створення, статус активності).

Вікно проходження тесту складається з верхньої панелі (назва тесту, лічильник питань, таймер та прогрес-бар), центральної робочої зони (текст питання та варіанти відповіді, що змінюються залежно від типу питання) та нижньої панелі навігації (кнопки «Назад», «Далі» та «Здати тест»).

Вікно перегляду результатів реалізовано у дволонковому макеті: зліва — таблиця результатів студентів із кольоровим кодуванням відсотка виконання; справа — кругова діаграма розподілу оцінок та зведена статистика

(загальна кількість спроб, середній бал, максимальний та мінімальний результат).

Для забезпечення доступності системи у різних умовах освітлення передбачено підтримку двох тем оформлення — світлої та темної. Перемикання між темами реалізується через динамічні ресурси (DynamicResource) WPF: клас ThemeManager замінює активний ResourceDictionary з колірними константами без перезапуску застосунку.

Висновки до розділу 2

У другому розділі виконано повне проектування системи інтерактивного тестування. Обґрунтовано застосування архітектурного шаблону MVVM, який забезпечує чітке розмежування між рівнями даних, логіки та подання, та органічно підтримується механізмом Data Binding фреймворку WPF.

Побудовано діаграму прецедентів, що визначає 12 варіантів використання системи для двох акторів — Студента та Викладача. Спроектовано реляційну базу даних у складі 6 таблиць (Users, Tests, Questions, Answers, Results, ResultDetails) із визначенням всіх атрибутів, первинних та зовнішніх ключів, каскадних правил та індексів.

Побудовано діаграму класів, що охоплює 9 сервісних класів рівня Model та 11 ViewModel-класів. Розроблено три діаграми послідовності для ключових сценаріїв: автентифікація, проходження тесту та збереження результату. Діаграма станів тесту визначає 5 станів об'єкта та переходи між ними.

Спроектовано алгоритм оцінювання результатів тестування, що підтримує чотири типи питань: бінарне нарахування для питань із вибором відповіді та пропорційне — для питань на встановлення відповідності. Розроблено дротяні макети чотирьох основних екранів системи. Усі проектні рішення, прийняті у цьому розділі, є основою для реалізації, що описана у наступному розділі.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Структура проекту та організація коду

Програмний проект реалізовано як рішення (solution) Visual Studio 2022, що складається з двох проектів: основного застосунку StudentTestingSystem (тип SDK-style WPF Application, цільова платформа net9.0-windows) та тестового проекту StudentTestingSystem.Tests (тип xUnit Test Project, net9.0). Такий поділ забезпечує незалежність тестів від UI-рівня та дозволяє виконувати їх у CI-середовищі без графічного інтерфейсу.

Основний проект організовано за принципом Feature-by-Layer: файли групуються за архітектурним рівнем (Models, Services, ViewModels, Views), а не за функціональною ознакою. Це відповідає стандартній практиці WPF/MVVM-проектів та спрощує навігацію по коду. Структуру проекту наведено у таблиці 3.1.

Таблиця 3.1 — Структура проекту системи тестування

Директорія / файл	Компонент	Призначення
Models/	Рівень Model	Класи сутностей: User, Test, Question, Answer, Result, ResultDetail, DTO
Services/	Рівень Model	Сервіси бізнес-логіки та їх інтерфейси: IAuthService, ITestService, IResultService, IUserService, IImportExportService
ViewModels/	Рівень ViewModel	ViewModel-класи для кожного вікна; BaseViewModel; RelayCommand; AsyncRelayCommand
Views/	Рівень View	XAML-файли вікон (.xaml) та їх code-behind (.xaml.cs)
Converters/	Інфраструктура	IValueConverter-реалізації: BoolToVisibilityConverter, StringToVisibilityConverter, PercentageToColorConverter, QuestionTypeToUkrainianConverter
Resources/	Інфраструктура	ResourceDictionary-файли тем: LightTheme.xaml, DarkTheme.xaml, Styles.xaml

Helpers/	Інфраструктура	ServiceLocator, ThemeManager, DatabaseInitializer
App.xaml / App.xaml.cs	Точка входу	Ініціалізація застосунку, реєстрація сервісів, ShutdownMode
StudentTestingSystem.Tests/	Тести	Окремий проект xUnit: модульні тести сервісів AuthService, TestService, ResultService, UserService

Файл App.xaml.cs є точкою входу застосунку. У методі OnStartup виконується послідовна ініціалізація:

– визначення шляху до файлу бази даних (поряд із виконуваним файлом у підпапці base/);
– реєстрація рядка підключення у ServiceLocator;
– реєстрація екземплярів сервісів (AuthService, TestService, ResultService, UserService, ImportExportService);
– виклик DatabaseInitializer.InitializeAsync() — створення схеми БД та seed-даних;
– відображення LoginWindow як стартового вікна.

Параметр ShutdownMode застосунку встановлено у значення OnExplicitShutdown, що дозволяє контролювати момент завершення процесу вручну — це необхідно для коректної реалізації циклу «вийти з акаунта → повернутись до вікна входу» без перезапуску застосунку.

3.2 Реалізація рівня даних: AppDbContext та DatabaseInitializer

Взаємодія з базою даних SQLite реалізована за допомогою Entity Framework Core 9 у підході Code First: структура бази даних визначається класами C# (моделями), а не SQL-скриптами. Центральним класом рівня даних є AppDbContext, що успадковує від Microsoft.EntityFrameworkCore.DbContext.

AppDbContext оголошує шість DbSet-властивостей, що відповідають таблицям бази даних:

Лістинг 3.1 — Фрагмент класу AppDbContext — оголошення DbSet-колекцій та конфігурація зв'язків

```
public class AppDbContext : DbContext
{
    public DbSet<User>         Users         { get; set; }
    public DbSet<Test>         Tests         { get; set; }
    public DbSet<Question>     Questions     { get; set; }
    public DbSet<Answer>       Answers       { get; set; }
    public DbSet<Result>       Results       { get; set; }
    public DbSet<ResultDetail> ResultDetails { get; set; }

    protected override void OnModelCreating(ModelBuilder mb)
    {
        mb.Entity<Test>()
          .HasMany(t => t.Questions)
          .WithOne(q => q.Test)
          .HasForeignKey(q => q.TestId)
          .OnDelete(DeleteBehavior.Cascade);

        mb.Entity<Question>()
          .HasMany(q => q.Answers)
          .WithOne(a => a.Question)
          .HasForeignKey(a => a.QuestionId)
          .OnDelete(DeleteBehavior.Cascade);
    }
}
```

Метод OnModelCreating налаштовує зв'язки між сутностями через Fluent API: каскадне видалення забезпечує автоматичне видалення дочірніх записів (питань, відповідей, результатів) при видаленні батьківського об'єкта. Це є більш надійним підходом порівняно з тригерами SQLite.

Клас DatabaseInitializer відповідає за первинну ініціалізацію бази даних. Метод InitializeAsync() викликає context.Database.EnsureCreatedAsync(), що автоматично створює всі таблиці відповідно до моделей, якщо файл бази даних ще не існує. Після створення схеми виконується заповнення seed-даними — двома демонстраційними обліковими записами:

Лістинг 3.2 — Фрагмент DatabaseInitializer — seed-дані демонстраційних акаунтів

```
if (!await context.Users.AnyAsync())
{
    context.Users.AddRange(
        new User {
            Username     = "teacher",
            PasswordHash = BC.HashPassword("teacher123", workFactor: 11),
            FullName     = "Викладач Демонстраційний",
            Role         = UserRole.Teacher,
            IsActive     = true
        },
        new User {
            Username     = "student",
            PasswordHash = BC.HashPassword("student123", workFactor: 11),
            FullName     = "Студент Демонстраційний",
            Role         = UserRole.Student,
            IsActive     = true
        }
    );
    await context.SaveChangesAsync();
}
```

3.3 Реалізація автентифікації та авторизації

Безпека облікових записів забезпечується алгоритмом BCrypt через бібліотеку BCrypt.Net-Next. BCrypt є адаптивною функцією хешування: параметр workFactor (cost factor) визначає обчислювальну складність і може збільшуватися з ростом продуктивності апаратного забезпечення. У системі встановлено workFactor = 11, що відповідає приблизно 300–500 мс на операцію хешування — достатньо для забезпечення безпеки та комфортної швидкості входу.

Клас AuthService реалізує інтерфейс IAuthService і надає два ключових методи:

Лістинг 3.3 — Фрагмент AuthService — реєстрація та автентифікація користувача

```
public async Task<(bool Success, string Error)> RegisterAsync(
    string username, string password, string fullName,
    string email, UserRole role)
{
    if (await _db.Users.AnyAsync(u => u.Username == username))
        return (false, "Користувач з таким іменем вже існує");

    var user = new User {
        Username     = username.Trim(),
        PasswordHash = BC.HashPassword(password, workFactor: 11),
        FullName     = fullName.Trim(),
        Email        = email.Trim(),
        Role         = role,
        IsActive     = true,
        CreatedAt    = DateTime.UtcNow
    };
    _db.Users.Add(user);
    await _db.SaveChangesAsync();
    return (true, string.Empty);
}

public async Task<User?> LoginAsync(string username, string password)
{
    var user = await _db.Users
        .FirstOrDefaultAsync(u => u.Username == username && u.IsActive);
    if (user == null) return null;
    return BC.Verify(password, user.PasswordHash) ? user : null;
}
```

Авторизація (розмежування прав доступу) реалізована на рівні MainViewModel: після успішного входу перевіряється властивість CurrentUser.Role. Якщо роль дорівнює UserRole.Teacher, відображається панель викладача (TeacherDashboardView); якщо UserRole.Student — панель студента (StudentDashboardView). Команди, доступні лише викладачам, прив'язані до кнопок, що присутні виключно у View викладача.

3.4 Реалізація модуля управління тестами

Модуль управління тестами охоплює операції створення, редагування та видалення тестів і складається з TeacherDashboardViewModel (список тестів), TestEditorViewModel (редактор) та TestEditorWindow (View). Реалізація підтримує чотири типи питань, перелік яких наведено у таблиці 3.2.

Таблиця 3.2 — Типи питань та їх відображення у системі

Код	Тип	Опис для викладача	Відображення для студента
0	SingleChoice	Одна правильна відповідь серед варіантів	RadioButton-список; вибір скидається при переході між питаннями
1	MultipleChoice	Одна або кілька правильних відповідей	CheckBox-список; кожна галочка незалежна
2	OpenText	Студент вводить довільну відповідь; еталон задається викладачем	TextBox для введення; порівняння без урахування регістру
3	Matching	Пари «лівий елемент → правий елемент»	ComboBox із перемішаними варіантами правої частини

Редактор тестів (TestEditorWindow) динамічно адаптує центральну зону відповідно до обраного типу питання. Для питань типу SingleChoice та MultipleChoice відображається DataGrid з варіантами відповідей та стовпцем «Правильна». Для OpenText — поле введення еталонної відповіді. Для Matching — список пар у форматі «[Лівий елемент] ↔ [Правий елемент (ComboBox)]». Перемикання здійснюється через DataTrigger у стилях XAML, що реагують на властивість IsMatching / IsOpenText поточного QuestionViewModel.

Збереження тесту реалізовано методом SaveAsync у TestEditorViewModel. При редагуванні існуючого тесту застосовується підхід «завантажити — оновити — зберегти» для уникнення конфліктів трекінгу Entity Framework Core:

Лістинг 3.4 — Фрагмент TestService.UpdateTestAsync — оновлення існуючого тесту

```
public async Task UpdateTestAsync(Test updated)
{
    var existing = await _db.Tests
        .Include(t => t.Questions)
            .ThenInclude(q => q.Answers)
        .FirstOrDefaultAsync(t => t.Id == updated.Id)
        ?? throw new KeyNotFoundException();

    // Оновити скалярні поля тесту
    existing.Title            = updated.Title;
    existing.Description      = updated.Description;
    existing.TimeLimitMinutes = updated.TimeLimitMinutes;
    existing.IsActive         = updated.IsActive;

    // Синхронізувати колекцію питань (додати / оновити / видалити)
    var incomingIds = updated.Questions.Select(q => q.Id).ToHashSet();
    foreach (var q in existing.Questions
                .Where(q => !incomingIds.Contains(q.Id)).ToList())
        _db.Questions.Remove(q);

    foreach (var uq in updated.Questions)
    {
        var eq = existing.Questions.FirstOrDefault(q => q.Id == uq.Id);
        if (eq == null) { existing.Questions.Add(uq); }
        else            { /* оновлення полів eq */ }
    }
    await _db.SaveChangesAsync();
}
```

Скріншот вікна редактора тесту наведено на рисунку 3.4.

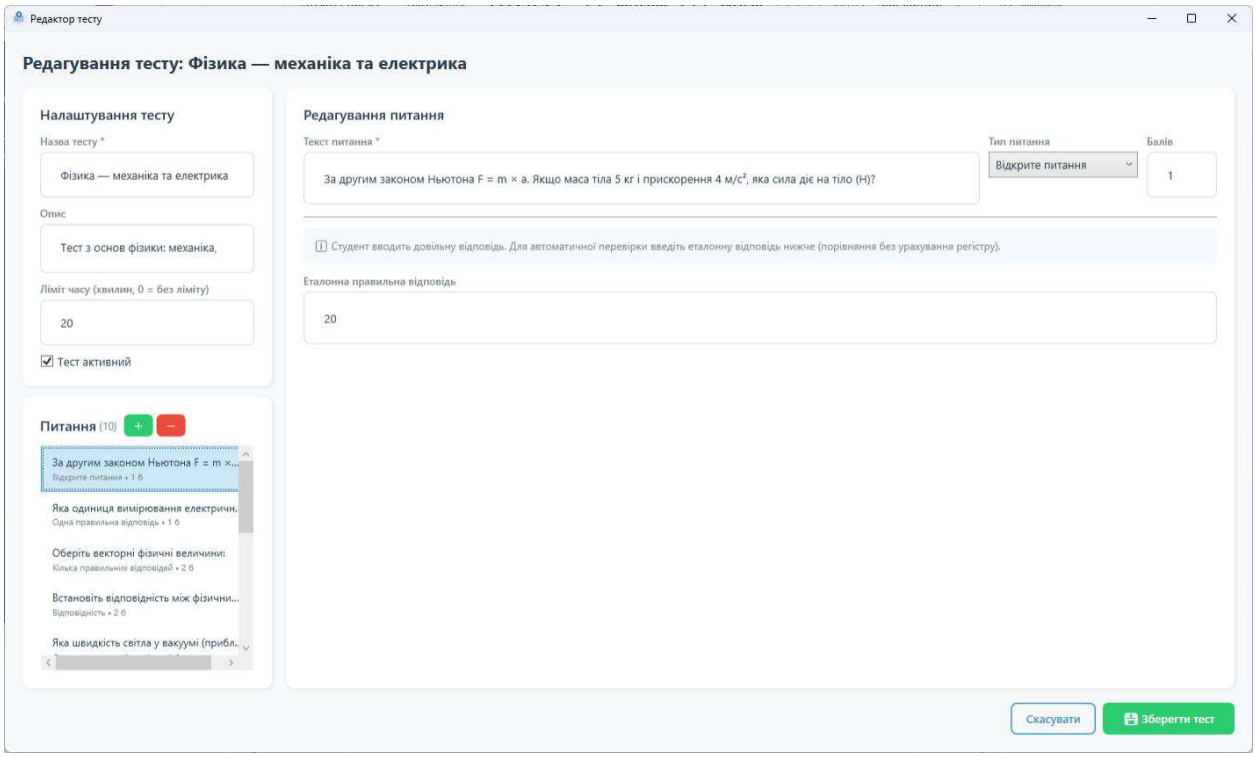


Рисунок 3.4.1. –Вікно редактора тесту (TestEditorWindow)

3.5 Реалізація модуля проходження тесту

Модуль проходження тесту реалізовано у класі TestTakingViewModel. При ініціалізації завантажується об'єкт Test з усіма питаннями та відповідями, після чого будується колекція TakingQuestionViewModel — об'єктів-оберток, що зберігають стан відповідей у пам'яті під час сеансу.

Таймер реалізовано через System.Timers.Timer з інтервалом 1000 мс. При спрацюванні таймера RemainingSeconds декрементується; при досягненні нуля автоматично викликається FinishAsync. Виклик із фонового потоку таймера маршалується в UI-потік через Application.Current.Dispatcher.Invoke. Обчислювані властивості TimerDisplay та TimerProgress сповіщають UI про зміни:

Лістинг 3.5 — Фрагмент TestTakingViewModel — таймер та обчислювані властивості

```
private void OnTimerTick(object? sender, ElapsedEventArgs e)
{
    Application.Current.Dispatcher.Invoke(() =>
    {
        if (--RemainingSeconds <= 0) { _timer.Stop(); _ = FinishAsync(); }
        OnPropertyChanged(nameof(TimerDisplay));
        OnPropertyChanged(nameof(TimerProgress));
    });
}

public string TimerDisplay =>
    HasTimer ? $"{RemainingSeconds / 60:D2}:{RemainingSeconds % 60:D2}" : "∞";

public double TimerProgress =>
    HasTimer
    ? (double)RemainingSeconds / (_test.TimeLimitMinutes!.Value * 60) * 100
    : 100;
```

Для питань типу MultipleChoice реалізовано Command-підхід замість TwoWay-прив'язки, оскільки WPF-механізм прив'язки для CheckBox у DataTemplate виявив нестабільну поведінку при прокручуванні. CheckBox прив'язується до IsSelected у режимі OneWay (тільки відображення), а кліки обробляються через ToggleAnswerCommand:

Лістинг 3.6 — Команди вибору відповідей у TakingQuestionViewModel

```
// MultipleChoice: перемкнути стан конкретної відповіді
ToggleAnswerCommand = new RelayCommand<TakingAnswerViewModel>(
    a => { if (a != null) a.IsSelected = !a.IsSelected; });

// SingleChoice: скинути всі, обрати одну
SelectAnswerCommand = new RelayCommand<TakingAnswerViewModel>(a =>
{
    if (a == null) return;
    foreach (var ans in Answers) ans.IsSelected = false;
    a.IsSelected = true;
});
```

Для питань типу Matching варіанти правої частини перемішуються алгоритмом Фішера–Єйтса (реалізовано через OrderBy(_ => Guid.NewGuid())) і зберігаються у властивості MatchOptions поточного TakingQuestionViewModel. Студент обирає відповідність зі спадного списку (ComboBox), що унеможливлює помилки введення та спрощує перевірку.

Скріншоти вікна проходження тесту для різних типів питань наведено на рисунках 3.5.1 та 3.5.2.

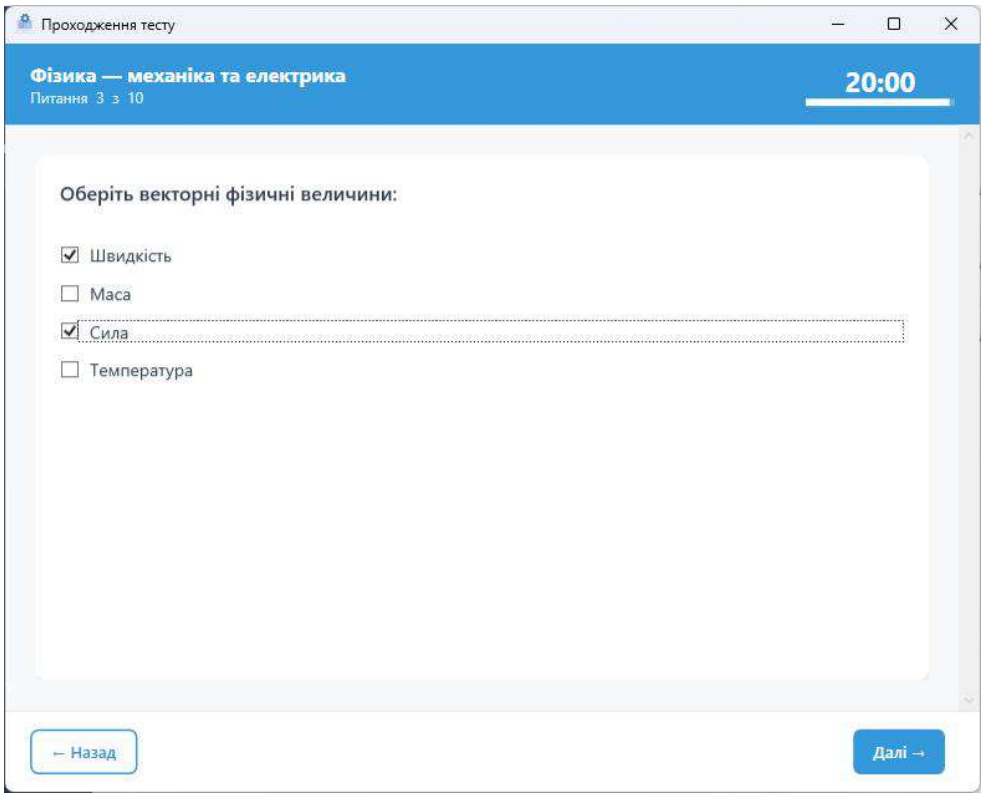


Рисунок 3.5.1. –Вікно проходження тесту — питання типу «Кілька правильних відповідей»

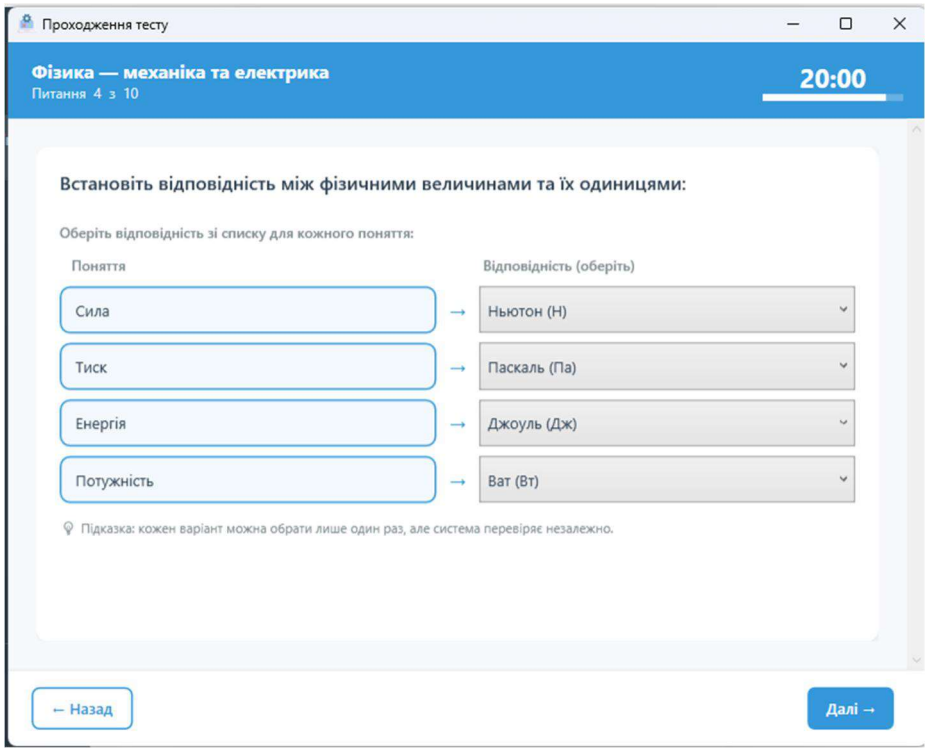


Рисунок 3.5.2. –Вікно проходження тесту — питання типу «Відповідність»

3.6 Реалізація модуля оцінювання та результатів

Після завершення тесту (натискання кнопки «Здати» або спрацювання таймера) викликається асинхронний метод FinishAsync у TestTakingViewModel. Метод зупиняє таймер, перебирає всі питання та обчислює нараховані бали відповідно до типу питання.

Лістинг 3.7 — Фрагмент FinishAsync — алгоритм підрахунку балів

```
foreach (var q in Questions)
{
    double pts = 0; bool isCorrect = false; string given = "";

    if (q.IsSingleChoice)
    {
        var sel = q.Answers.FirstOrDefault(a => a.IsSelected);
        isCorrect = sel?.AnswerId ==
            q.Answers.FirstOrDefault(a => a.IsCorrect_Original)?.AnswerId;
        pts = isCorrect ? q.Points : 0;
    }
    else if (q.IsMultipleChoice)
    {
        var selected = q.Answers.Where(a => a.IsSelected)
                                .Select(a => a.AnswerId).ToHashSet();
        var correct  = q.Answers.Where(a => a.IsCorrect_Original)
                                .Select(a => a.AnswerId).ToHashSet();
        isCorrect = selected.SetEquals(correct);
        pts = isCorrect ? q.Points : 0;
    }
    else if (q.IsOpenText)
    {
        var etalon = q.Answers.FirstOrDefault()?.MatchTarget ?? "";
        isCorrect  = string.Equals(q.OpenTextAnswer.Trim(),
                        etalon.Trim(),
                        StringComparison.OrdinalIgnoreCase);
        pts = isCorrect ? q.Points : 0;
    }
    else if (q.IsMatching)
    {
        int correct = q.Answers.Count(a =>
            string.Equals(a.SelectedMatchOption?.Trim(),
                          a.MatchTarget?.Trim(),
                          StringComparison.OrdinalIgnoreCase));
        pts = Math.Round((double)correct / q.Answers.Count * q.Points, 2);
        isCorrect = correct == q.Answers.Count;
    }
    score += pts;
}
```

Після підрахунку формується об'єкт Result та колекція ResultDetail (по одному запису на питання). Обидва об'єкти передаються до ResultService.SaveResultAsync, де зберігаються в одній транзакції через EF Core. Студент переходить до екрана з підсумком: відсоток виконання, оцінка, кольорове коло з відсотком.

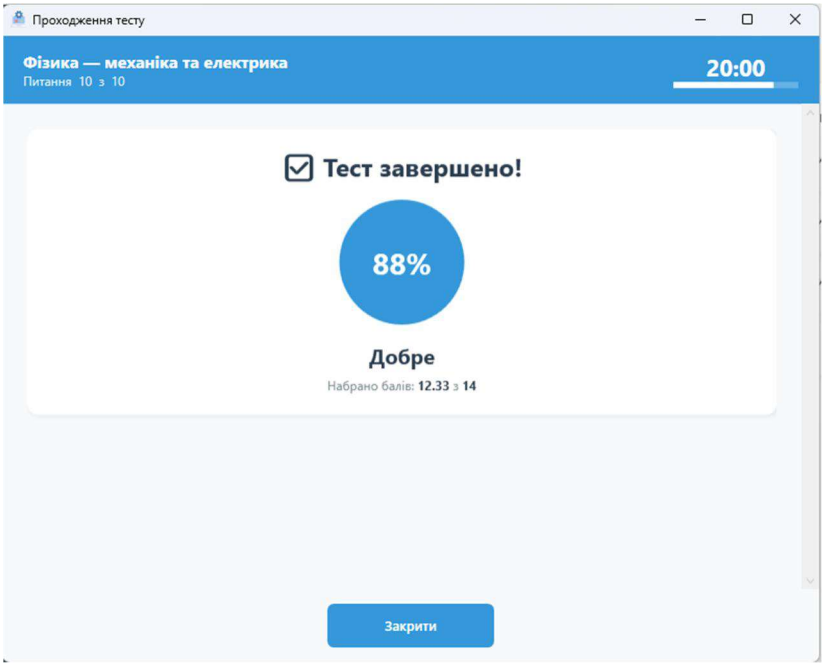


Рисунок 3.6.1. – Вікно результатів

Вікно результатів (TestResultsWindow) для викладача відображає зведену статистику: таблицю результатів усіх студентів із кольоровим кодуванням (зелений — відмінно, синій — добре, жовтий — задовільно, червоний — незадовільно), кругову діаграму OxyPlot у стилі «пончика» (donut chart) та зведені показники: кількість спроб, середній бал, максимальний та мінімальний результат. Скріншот наведено на рисунку 3.7.

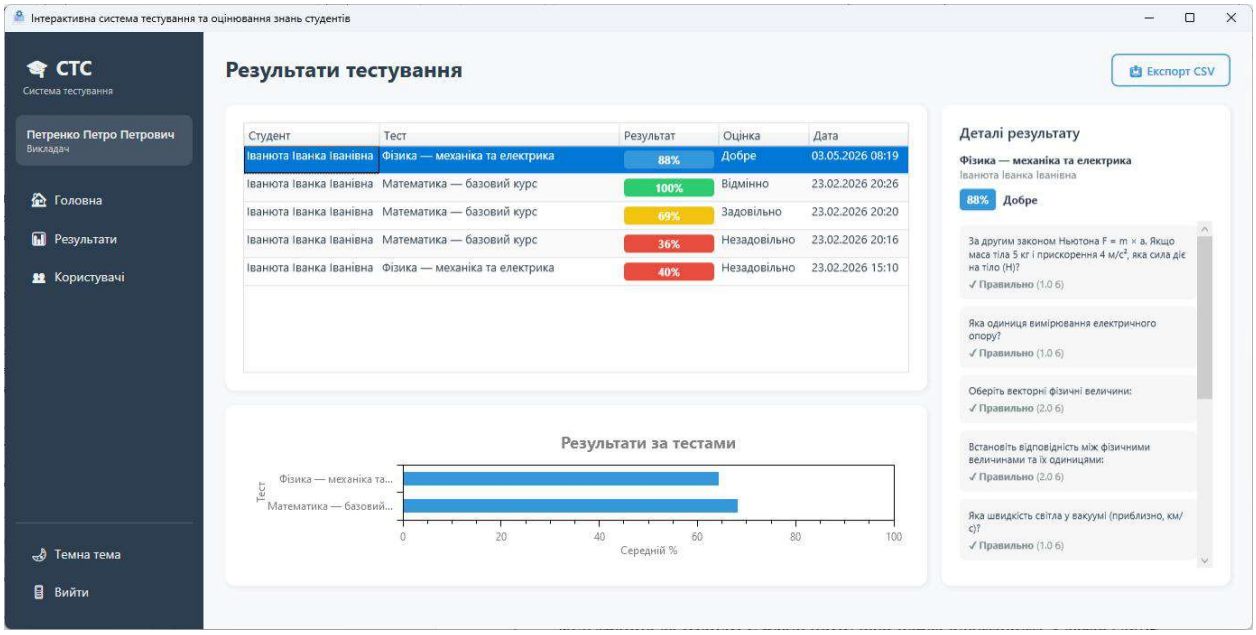


Рисунок 3.6.2. – Вікно перегляду результатів тесту (TestResultsWindow)

3.7 Реалізація модуля адміністрування

Модуль адміністрування доступний лише для викладачів та надає можливість управління обліковими записами студентів. Панель відображається у вигляді вкладки «Студенти» в головному вікні викладача. UsersViewModel завантажує повний список студентів та забезпечує фільтрацію за іменем у реальному часі через властивість SearchText:

Лістинг 3.8 — Фрагмент UsersViewModel — фільтрація списку студентів

```
public string SearchText
{
    get => _searchText;
    set
    {
        SetProperty(ref _searchText, value);
        FilteredUsers = string.IsNullOrWhiteSpace(value)
            ? new ObservableCollection<User>(_allUsers)
            : new ObservableCollection<User>(_allUsers.Where(u =>
                u.FullName.Contains(value, StringComparison.OrdinalIgnoreCase) ||
                u.Username.Contains(value, StringComparison.OrdinalIgnoreCase)));
    }
}
```

Вікно редагування користувача (EditUserWindow) надає доступ до полів: FullName, Username, Email, ролі (Student / Teacher) та статусу активності (IsActive). Зміна пароля можлива лише при встановленні галочки «Оновити пароль» — умовно відображуване поле PasswordBox запобігає випадковому скиданню пароля. Скріншот наведено на рисунку 3.8.

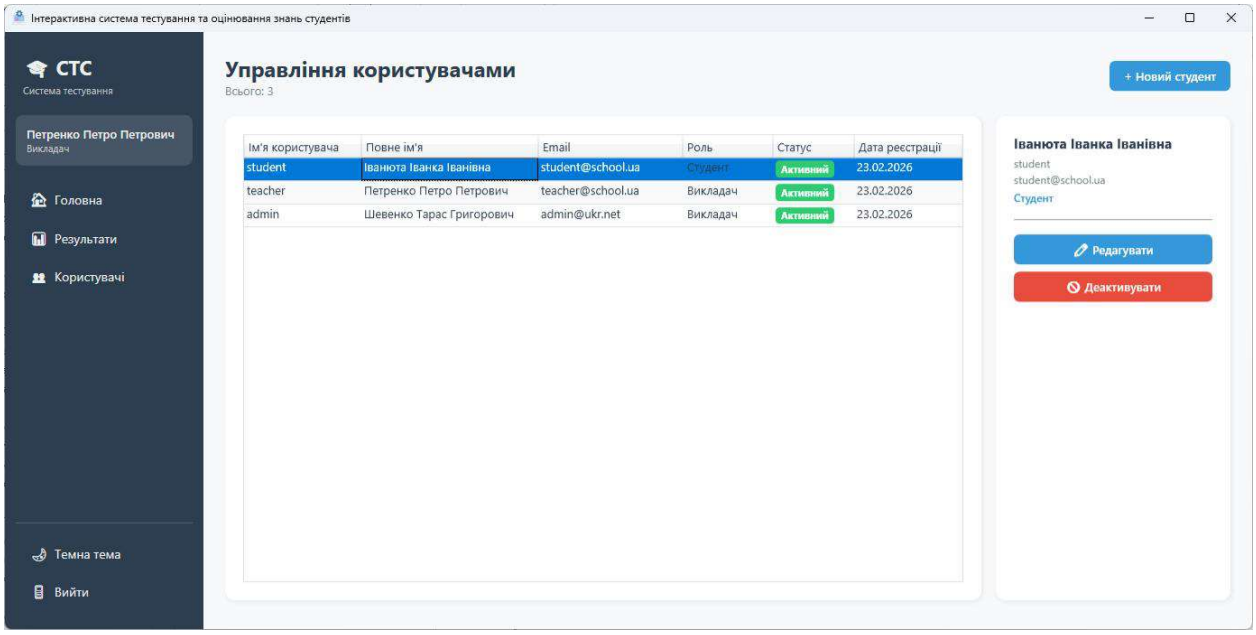


Рисунок 3.7.1. – Панель адміністрування користувачів

3.8 Реалізація імпорту та експорту тестів

Функція імпорту/експорту дозволяє переносити тести між різними інсталяціями системи або створювати банки тестів поза системою. Усі формати базуються на єдиному DTO-класі TestExportDto, що відображає структуру тесту, питань та відповідей без внутрішніх ідентифікаторів бази даних. Підтримувані операції наведено у таблиці 3.4.

Таблиця 3.4 — Операції імпорту та експорту в системі тестування

Операція	Формат	Бібліотека	Опис
Імпорт тесту	JSON	Newtonsoft.Json	Десеріалізація TestExportDto → збереження тесту через TestService
Імпорт тесту	XML	System.Xml.Serialization	XmlSerializer десеріалізує TestExportDto; подальше збереження аналогічне JSON
Експорт тесту	JSON	Newtonsoft.Json	Серіалізація Test → TestExportDto → JSON-файл з відступами
Експорт тесту	XML	System.Xml.Serialization	Серіалізація TestExportDto у XML-файл
Експорт результатів	CSV	CsvHelper	Запис Result-об'єктів у CSV з роздільником «;» для сумісності з MS Excel (українська локаль)

Для забезпечення сумісності CSV-файлів результатів із Microsoft Excel в українській локалі використовується символ крапки з комою (;) як роздільник стовпців замість стандартної коми. Це зумовлено тим, що в Windows з українськими регіональними налаштуваннями Excel за замовчуванням очікує саме цей роздільник. Перший рядок CSV містить заголовки стовпців: StudentName, Username, Score, MaxScore, Percentage, Grade, TimeSpentSeconds, DateCompleted.

Лістинг 3.9 — Фрагмент ImportExportService — серіалізація тесту у JSON

```
public async Task ExportTestToJsonAsync(Test test, string filePath)
{
    var dto = new TestExportDto
    {
        Title            = test.Title,
        Description      = test.Description,
        TimeLimitMinutes = test.TimeLimitMinutes,
        Questions        = test.Questions
            .OrderBy(q => q.OrderIndex)
            .Select(q => new QuestionExportDto
            {
                Text       = q.Text,
                Type       = q.Type.ToString(),
                Points     = q.Points,
                OrderIndex = q.OrderIndex,
                Answers    = q.Answers
                    .OrderBy(a => a.OrderIndex)
                    .Select(a => new AnswerExportDto
                    {
                        Text        = a.Text,
                        IsCorrect   = a.IsCorrect,
                        MatchTarget = a.MatchTarget,
                        OrderIndex  = a.OrderIndex
                    }).ToList()
            }).ToList()
    };
    var json = JsonConvert.SerializeObject(dto,
                   Formatting.Indented,
                   new JsonSerializerSettings { NullValueHandling = NullValueHandling.Ignore });
    await File.WriteAllTextAsync(filePath, json, Encoding.UTF8);
}
```

3.9 Реалізація графічного відображення результатів

Для графічного відображення розподілу оцінок у вікні результатів використовується бібліотека OxyPlot. Кругова діаграма будується з чотирьох секторів, що відповідають чотирьом градаціям успішності: «Відмінно» (≥90%), «Добре» (≥75%), «Задовільно» (≥60%) та «Незадовільно» (<60%).

Для підвищення читабельності діаграму реалізовано у стилі «пончика» (InnerDiameter = 0.35): центральний отвір зменшує нагромадженість при малій кількості секторів. Підписи всередині секторів відображають лише відсоток (InsideLabelFormat = "{1:0}%"), тоді як назви категорій і кількість студентів виносяться у кастомну WPF-легенду — ItemsControl з кольоровими Ellipse-індикаторами та ProgressBar для наочного відображення частки кожної оцінки.

Лістинг 3.10 — Фрагмент TestResultsWindow — побудова кругової діаграми OxyPlot

```
var pie = new PieSeries
{
    InsideLabelPosition = 0.65,
    OutsideLabelFormat  = null,
    InsideLabelFormat   = "{1:0}%",
    InnerDiameter       = 0.35,
    Diameter            = 0.82,
    TextColor           = OxyColors.White,
    FontSize            = 12,
};

foreach (var b in buckets.Where(b => b.Count > 0))
    pie.Slices.Add(new PieSlice(b.Label, b.Count) { Fill = b.OxyColor });

model.Series.Add(pie);
ChartView.Model = model;
```

3.10 Реалізація системи тем оформлення

Система підтримує дві теми оформлення: світлу (за замовчуванням) та темну. Перемикання між темами реалізовано без перезапуску застосунку через механізм DynamicResource WPF. Клас ThemeManager замінює перший елемент у Application.Resources.MergedDictionaries на відповідний ResourceDictionary:

Лістинг 3.11 — Клас ThemeManager — перемикання між темами

```
public static class ThemeManager
{
    private static bool _isDark = false;

    public static void Toggle()
    {
        _isDark = !_isDark;
        var uri = new Uri(_isDark
            ? "Resources/DarkTheme.xaml"
            : "Resources/LightTheme.xaml",
            UriKind.Relative);
        var dict = new ResourceDictionary { Source = uri };
        Application.Current.Resources.MergedDictionaries[0] = dict;
    }

    public static bool IsDark => _isDark;
}
```

Усі елементи UI прив'язані до кольорів через DynamicResource (а не StaticResource), що дозволяє їм реагувати на зміну ResourceDictionary в реальному часі. Ключові ресурси теми наведено у таблиці 3.5.

Таблиця 3.5 — Порівняння кольорів світлої та темної тем оформлення

Ресурс (DynamicResource)	Світла тема	Темна тема
BackgroundBrush	#FFFFFF (білий)	#1E1E2E (темно-синій)
SurfaceBrush	#F8F9FA (світло-сірий)	#2A2A3E (сірувато-синій)
PrimaryBrush	#3498DB (синій)	#5DADE2 (світло-синій)
TextBrush	#1A1A2E (темний)	#E8E8F0 (світлий)
TextSecondaryBrush	#6C757D (сірий)	#A0A0B8 (сіро-блакитний)
DangerBrush	#E74C3C (червоний)	#FF6B6B (яскраво-червоний)
AccentBrush	#2ECC71 (зелений)	#27AE60 (смарагдовий)

Загальний перелік реалізованих екранів системи з посиланнями на відповідні рисунки та описом ключових UI-елементів наведено у таблиці 3.6.

Таблиця 3.6 — Перелік екранів системи та їх UI-складові

Вікно	Рисунок	Основні елементи інтерфейсу
LoginWindow	3.1	Логотип, TextBox (Username), PasswordBox, кнопки «Увійти» / «Зареєструватися», TextBlock для помилки
RegisterWindow	3.2	Поля: FullName, Username, Email, Password, ConfirmPassword; RadioButton вибору ролі; валідація; кнопки
TeacherDashboardView	3.3	DataGrid тестів, панель фільтрації, кнопки CRUD, Імпорт, Експорт
TestEditorWindow	3.4	ListBox питань, ComboBox типу питання, динамічні секції відповідей для кожного типу
StudentDashboardView	3.5	ListBox тестів із кастомним стилем виділення, панель останніх результатів
TestTakingWindow	3.6	Верхня смуга (назва, таймер, ProgressBar), центральна зона питань, навігаційні кнопки
TestResultsWindow	3.7	DataGrid результатів, OxyPlot кругова діаграма, кастомна легенда, зведена статистика
UsersView (Admin)	3.8	DataGrid користувачів, TextBox пошуку, кнопки «Редагувати» / «Деактивувати»

На рисунках 3.1–3.3 наведено скріншоти ключових екранів: вікно входу, вікно реєстрації та панель викладача.

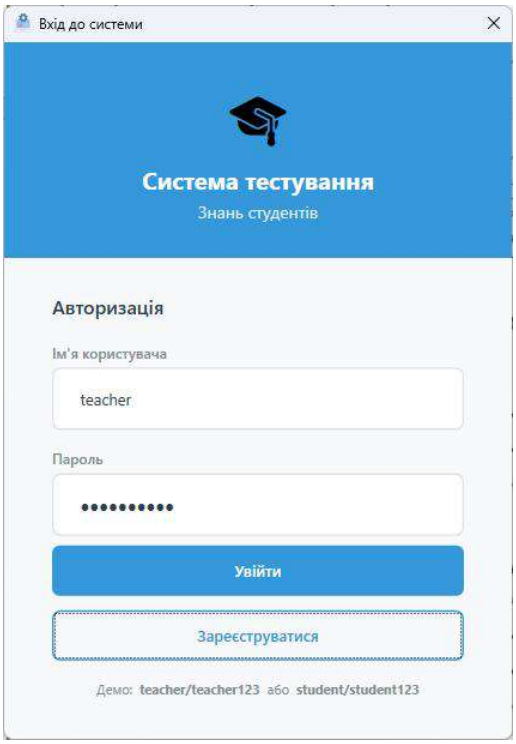


Рисунок 3.10.1. –Вікно входу до системи (LoginWindow)

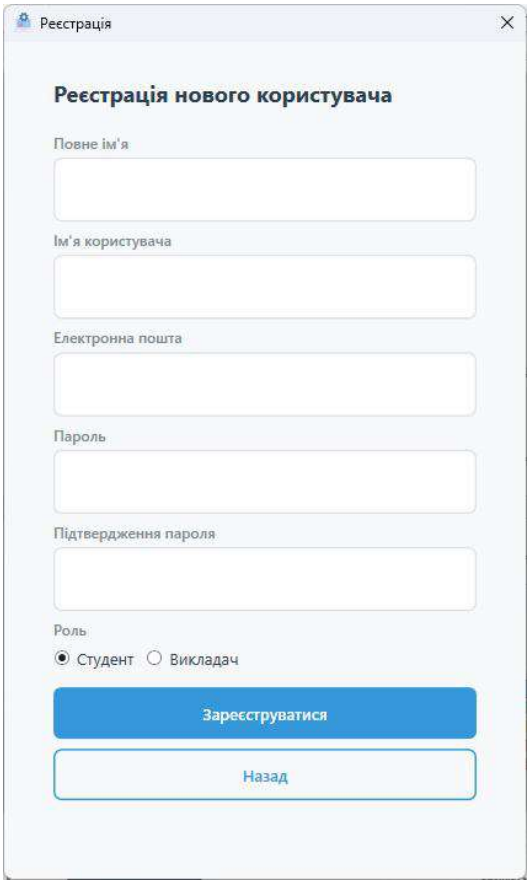


Рисунок 3.10.2. – Вікно реєстрації нового користувача (RegisterWindow)

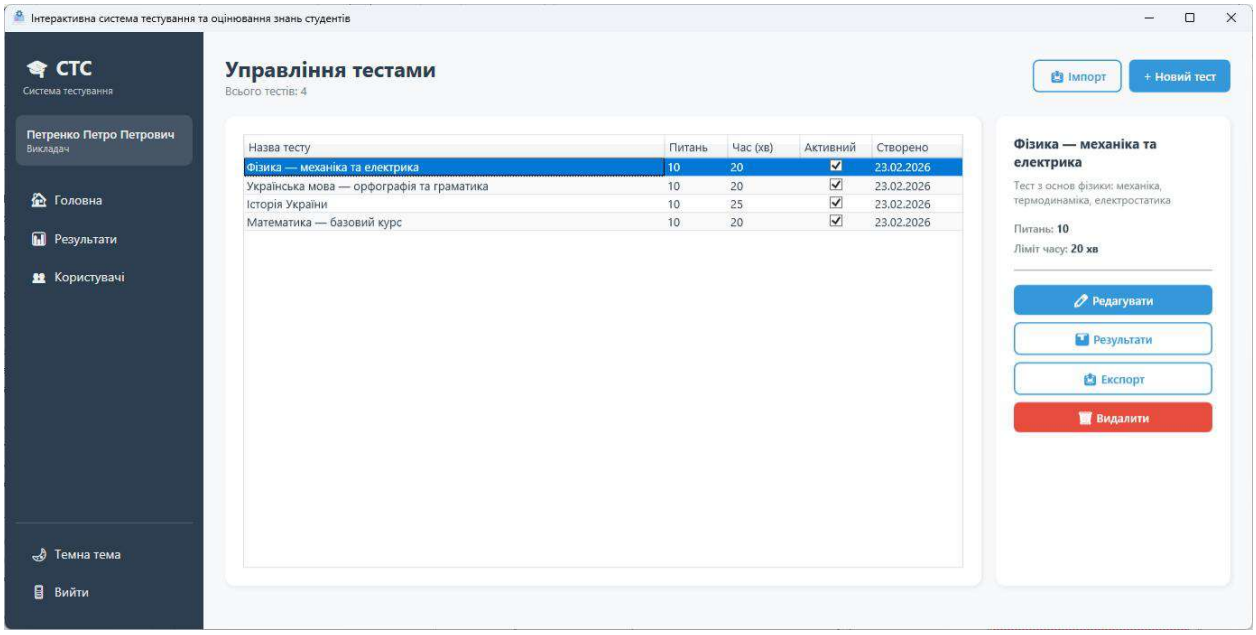


Рисунок 3.10.3. – Панель керування тестами (TeacherDashboardView)

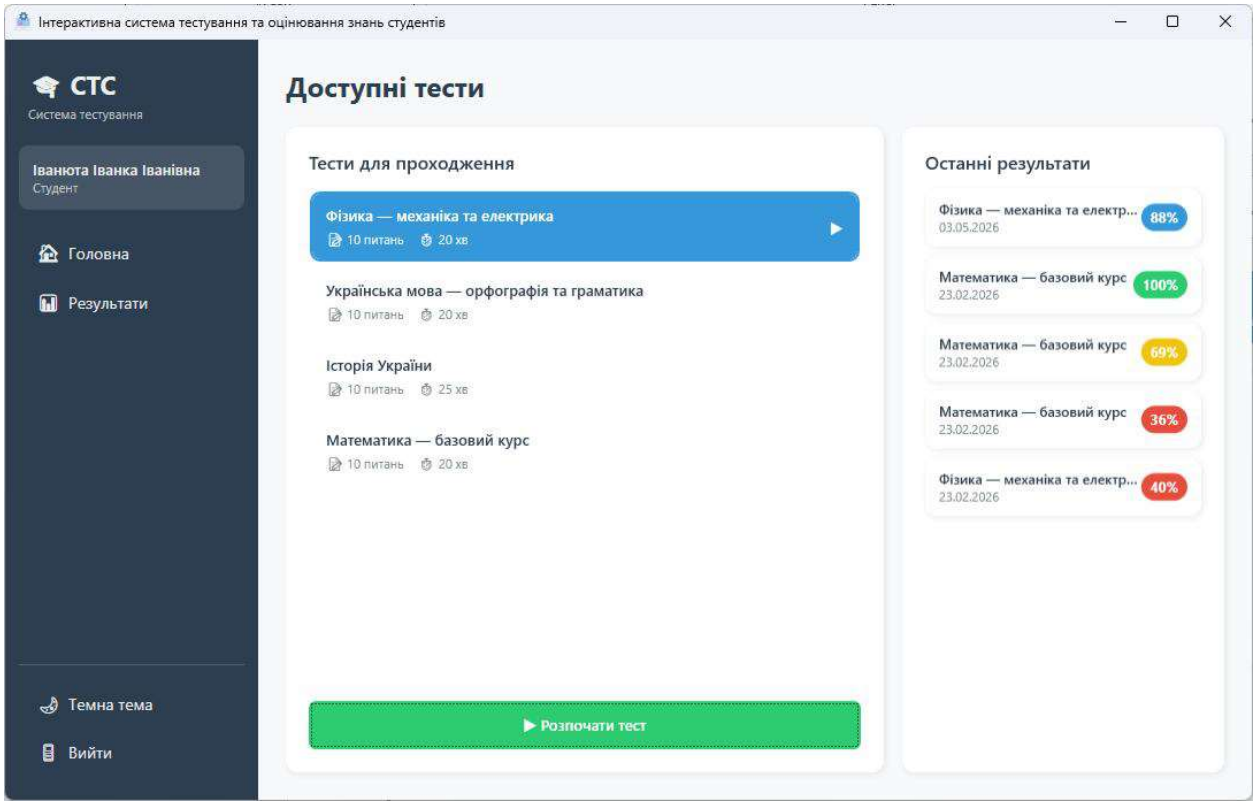
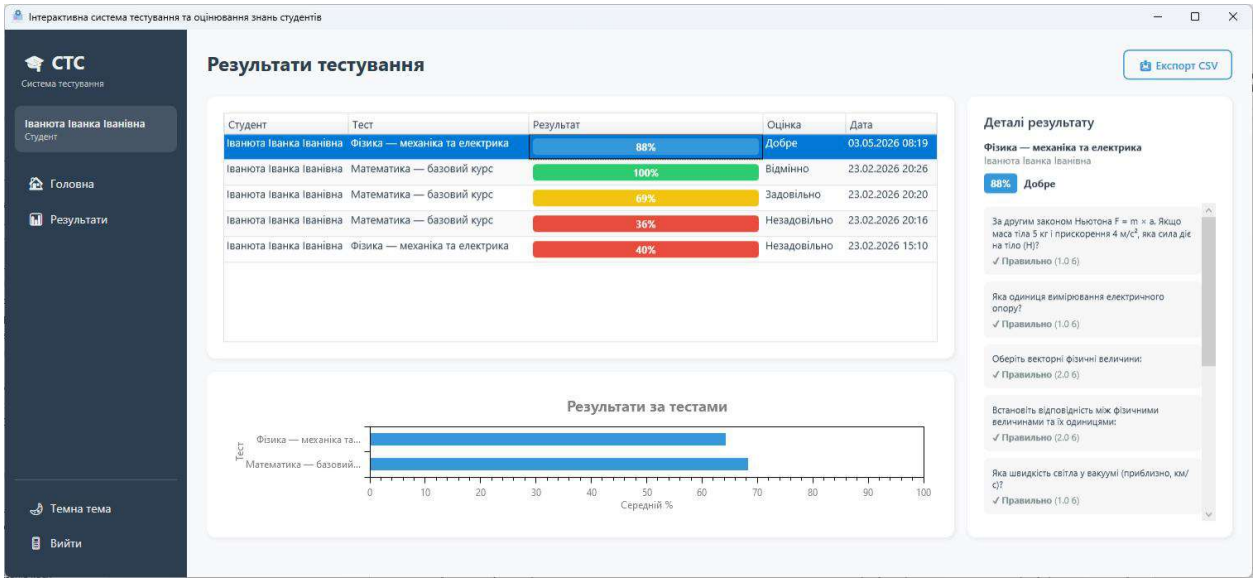


Рисунок 3.10.4. – Панель студента (StudentDashboardView)



Висновки до розділу 3

У третьому розділі описано повну реалізацію системи інтерактивного тестування. Проект організовано за принципом Feature-by-Layer у відповідності до архітектурного шаблону MVVM з чітким розмежуванням між рівнями Models, Services, ViewModels та Views.

Рівень даних реалізовано на основі Entity Framework Core 9 з підходом Code First: AppDbContext визначає 6 таблиць, зв'язки між ними та каскадні правила видалення через Fluent API. Автентифікація забезпечується алгоритмом BCrypt з workFactor = 11; авторизація реалізована на рівні ViewModel через перевірку ролі поточного користувача.

Реалізовано чотири типи питань: одна правильна відповідь (RadioButton + SelectAnswerCommand), кілька правильних відповідей (CheckBox + ToggleAnswerCommand), відкрита відповідь (TextBox) та відповідність (ComboBox з перемішаними варіантами). Таймер тесту реалізовано через System.Timers.Timer з маршалізацією у UI-потік.

Розроблено алгоритм автоматичного оцінювання результатів, що підтримує бінарне нарахування балів для питань із вибором та пропорційне — для питань на відповідність. Реалізовано модулі імпорту тестів (JSON, XML) та експорту результатів (CSV), графічного відображення результатів (OxyPlot donut chart) та перемикання тем оформлення без перезапуску застосунку. Загалом реалізовано 11 лістингів програмного коду та 8 екранів інтерфейсу.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено інтерактивну систему тестування та оцінювання знань студентів, яка є актуальним інструментом для цифровізації навчального процесу.

Під час дослідження було проаналізовано предметну галузь та встановлено, що автоматизований контроль дозволяє забезпечити об'єктивність, швидкість перевірки та систематичність оцінювання. Проведений огляд існуючих аналогів, таких як Moodle та Google Forms, виявив потребу в розробці автономного настільного рішення, яке не залежить від інтернет-з'єднання. На основі аналізу було сформовано перелік функціональних вимог, що включають підтримку різних типів питань, автоматичний підрахунок балів та розмежування прав доступу.

Для реалізації системи обрано сучасний технологічний стек, що базується на мові програмування C# 13 та платформі .NET 9. Архітектура програмного продукту побудована на основі шаблону MVVM, що забезпечило чітке розділення логіки, даних та інтерфейсу користувача. Рівень представлення реалізовано за допомогою технології WPF, яка надала потужні інструменти для створення гнучкого графічного інтерфейсу.

У другому розділі було проведено детальне проектування системи, що включало побудову UML-діаграм прецедентів, класів та послідовності. Спроектована реляційна база даних у середовищі SQLite складається з шести взаємопов'язаних таблиць, що забезпечують цілісність інформації про тести та результати. Особливу увагу приділено алгоритму оцінювання, який підтримує як бінарне нарахування балів, так і пропорційне для питань на встановлення відповідності. Розроблена модель станів тесту дозволила коректно описати життєвий цикл об'єкта від моменту створення до архівації.

У третьому розділі детально описано програмну реалізацію всіх модулів системи, включаючи рівень даних на основі Entity Framework Core 9. Реалізована система автентифікації використовує алгоритм BCrypt для безпечного зберігання паролів у хешованому вигляді. Модуль управління

тестами дозволяє викладачам створювати, редагувати та видаляти тестові завдання з підтримкою чотирьох типів питань. Модуль проходження тесту забезпечує зручний інтерфейс для студентів з інтегрованим таймером та автоматичною фіксацією відповідей. Важливою частиною роботи стала реалізація механізму імпорту та експорту тестів у форматах JSON та XML для обміну даними. Для аналітичної роботи викладача впроваджено модуль графічного відображення статистики успішності за допомогою бібліотеки OxyPlot.

Система підтримує динамічне перемикання тем оформлення, що підвищує ергономічність та зручність використання у різних умовах освітлення. Впроваджений сервіс логування Serilog дозволяє відстежувати роботу системи та оперативно діагностувати можливі помилки.

Практична цінність роботи полягає у створенні готового до використання програмного забезпечення, що може бути впроваджене у Житомирському агротехнічному фаховому коледжі.

Система повністю відповідає поставленій меті та завданням, демонструючи високу продуктивність та інтуїтивно зрозумілий інтерфейс. Використання паттерна Service Locator дозволило гнучко керувати залежностями між компонентами системи.

Реалізований експорт результатів у формат CSV спрощує подальшу обробку даних у табличних процесорах. Застосування підходу Code First при проектуванні бази даних значно спростило процес оновлення структури даних у майбутньому. Створена система є масштабованою, що дозволяє у перспективі додавати нові типи питань та аналітичні звіти

ПЕРЕЛІК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Про освіту : Закон України від 05.09.2017 № 2145-VIII. URL: https://zakon.rada.gov.ua/laws/show/2145-19 (дата звернення: 12.03.2026).
2. Про вищу освіту : Закон України від 01.07.2014 № 1556-VII. URL: https://zakon.rada.gov.ua/laws/show/1556-18 (дата звернення: 12.03.2026).
3. Бичков О. С. Сучасні технології розробки програмного забезпечення : навч. посіб. Київ : ВПЦ «Київський університет», 2019. 215 с.
4. Бублик В. В. Об'єктно-орієнтоване програмування : підручник. Київ : ІТ-книга, 2015. 624 с.
5. Вакуленко Т. С., Ломакович С. В., Терещенко В. М. Технології розроблення тестових завдань : навч.-метод. посіб. Київ : УЦОЯО, 2021. 144 с.
6. Глибовець М. М., Олецький О. В. Архітектура програмних систем : підручник. Київ : НаУКМА, 2016. 320 с.
7. Гужва В. М. Проектування інформаційних систем : навч. посіб. Київ : КНЕУ, 2018. 360 с.
8. Жуковський С. С. Об'єктно-орієнтоване програмування : навч. посіб. Житомир : Вид-во ЖДУ ім. І. Франка, 2017. 132 с.
9. Ковалюк Т. В. Алгоритмізація та програмування : підручник. Львів : Магнолія 2006, 2020. 400 с.
10. Коротєєва Т. О. Алгоритми та структури даних : навч. посіб. Львів : Вид-во Львівської політехніки, 2018. 160 с.
11. Кравець П. О. Об'єктно-орієнтоване програмування : навч. посіб. Львів : Вид-во Львівської політехніки, 2017. 632 с.
12. Литвин В. В., Пасічник В. В., Яцишин Ю. В. Інформаційні технології та моделювання складних систем : підручник. Львів : Новий Світ-2000, 2019. 464 с.
13. Методичні рекомендації щодо створення тестових завдань / за заг. ред. О. В. Авраменка. Київ : Грамота, 2020. 88 с.

14. Спірін О. М. Понятійно-термінологічний апарат дистанційного навчання в системі неперервної професійної освіти. Інформаційні технології і засоби навчання. 2016. Т. 53, № 3. С. 1–15.
15. Трофименко О. Г. C#. Створення додатків Windows : навч. посіб. Одеса : ОНАЗ ім. О. С. Попова, 2017. 188 с.
16. Шеховцов В. А. Операційні системи : підручник. Київ : Видавнича група BHV, 2015. 576 с.
17. Щербаков О. В. Організація баз даних та знань : навч. посіб. Харків : ХНЕУ ім. С. Кузнеця, 2017. 176 с.
18. Albahari J. C# 12 in a Nutshell: The Definitive Reference. Sebastopol : O'Reilly Media, 2023. 1056 p.
19. Freeman A. Pro WPF 8.5 in C#: Windows Presentation Foundation in .NET 8. Berkeley : Apress, 2024. 820 p.
20. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Reading : Addison-Wesley, 1994. 395 p.
21. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston : Prentice Hall, 2017. 432 p.
22. Microsoft Learn. .NET documentation. URL: https://learn.microsoft.com/en-us/dotnet/ (дата звернення: 10.04.2026).
23. Microsoft Learn. Windows Presentation Foundation (WPF) documentation. URL: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/ (дата звернення: 10.04.2026).
24. Price M. J. C# 13 and .NET 9 – Modern Cross-Platform Development Fundamentals. Birmingham : Packt Publishing, 2024. 850 p.
25. SQLite Documentation. URL: https://www.sqlite.org/docs.html (дата звернення: 12.04.2026).