Полтавський університет економіки і торгівлі
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп’ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____Олена ОЛЬХОВСЬКА
(підпис)

«____»_____2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему
«ІНФОРМАЦІЙНА СИСТЕМА ДЛЯ ГЕНЕРАЦІЇ ТЕСТІВ ТА ОЦІНЮВАННЯ ЗНАНЬ ЗДОБУВАЧІВ З МАТЕМАТИЧНИХ ДИСЦИПЛІН»

зі спеціальності 122 «Комп’ютерні науки»
освітня програма «Комп’ютерні науки»
ступеня бакалавр

Виконавець роботи Лісненко Максим Романович

_____«__»_____202_р.
(підпис)

Науковий керівник к.ф.-м.н., доц., Парфьонова Тетяна Олександрівна

_____«__»_____202_р.
(підпис)

Рецензент _____

ПОЛТАВА 2026

РЕФЕРАТ

Записка: 84 с., 16 рис., 2 додатки, 38 джерел.

ІНФОРМАЦІЙНА СИСТЕМА, ГЕНЕРАЦІЯ ТЕСТІВ, ОЦІНЮВАННЯ ЗНАНЬ, МАТЕМАТИЧНІ ДИСЦИПЛІНИ, БІБЛІОТЕКА ФОРМУЛ, АВТОМАТИЗАЦІЯ КОНТРОЛЮ, .NET CORE.

Об’єкт розробки – процеси створення тестових завдань та автоматизованого оцінювання знань здобувачів у навчальному процесі.

Мета роботи – проектування та програмна реалізація інформаційної системи для генерації тестів та оцінювання знань здобувачів з математичних дисциплін.

Методи дослідження – методи математичного моделювання тестових завдань, об’єктно-орієнтоване програмування (C#), платформа .NET Core, система керування базами даних SQLite, Entity Framework.

Отримані результати. Виконано огляд існуючих систем контролю знань з математичних дисциплін, виявлено їх переваги та недоліки. Розроблено архітектуру інформаційної системи, побудовано UML-діаграми та ER-діаграму бази даних. Програмно реалізовано модулі генерації тестів, автоматичного оцінювання відповідей, а також бібліотеку математичних формул, яка дозволяє викладачеві обирати формули під час створення тесту. Система підтримує ролі адміністратора, викладача та студента. Проведено тестування, яке підтвердило коректність роботи системи.

Рекомендації щодо використання. Розроблена система може бути впроваджена в навчальний процес закладів вищої освіти для контролю знань з математичних дисциплін (алгебра, геометрія, математичний аналіз, дискретна математика тощо).

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	Пояснення умовних позначень, скорочень, символів
БД	база даних
ПЗ	програмне забезпечення
СУБД	система управління базами даних
API	програмний інтерфейс додатку
EF Core	Entity Framework Core (об'єктно-реляційне відображення для C#)
ER-діаграма	діаграма сутність-зв'язок (логічна модель бази даних)
HTTP	протокол передачі гіпертексту
IDE	інтегроване середовище розробки
JSON	текстовий формат обміну даними

Умовні позначення, символи, скорочення, терміни	Пояснення умовних позначень, скорочень, символів
LMS	система управління навчанням
NET Core	кросплатформова середовище розробки від Microsoft
UI	інтерфейс користувача
UML	уніфікована мова моделювання (візуалізація архітектури)
UX	досвід користувача (зручність роботи із системою)

ВСТУП

Актуальність теми зумовлена необхідністю автоматизації контролю знань з математичних дисциплін у закладах вищої освіти. Ручне створення тестів, генерація варіантів та перевірка відповідей є трудомісткими процесами, які потребують значних витрат часу викладача.

Об’єктом роботи є процеси створення тестових завдань, проведення тестування та оцінювання знань здобувачів з математичних дисциплін.

Метою роботи є проектування та програмна реалізація інформаційної системи для генерації тестів та оцінювання знань здобувачів з математичних дисциплін.

Основні завдання. Для досягнення мети необхідно виконати аналіз існуючих систем тестування, визначити вимоги до функціональності системи, спроектувати архітектуру та логічну модель бази даних, реалізувати модулі генерації тестів та автоматичного оцінювання відповідей.

Практичне значення роботи полягає у створенні готової інформаційної системи, яка може бути впроваджена в навчальний процес закладів вищої освіти для контролю знань з математичних дисциплін, таких як алгебра та геометрія, математичний аналіз тощо.

Структура кваліфікаційної роботи включає вступ, чотири розділи, висновки, список використаних джерел та додатки. У першому розділі наведено постановку задачі та вимоги до системи. Другий розділ містить інформаційний огляд аналогів. Третій розділ присвячено теоретичній частині та проектуванню архітектури. У четвертому розділі описано практичну реалізацію системи, включаючи бібліотеку формул та інструкцію для користувача. Обсяг 84 с., 16 рис., 2 додатки, 38 джерел.

1. ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовна постановка задачі розробки інформаційної системи

Завданням кваліфікаційної роботи є створення інформаційної системи для автоматизованої генерації тестових завдань та оцінювання знань здобувачів з математичних дисциплін. Система повинна забезпечувати викладача засобами для швидкого створення тестів, а студентів – для проходження тестування та отримання результатів. Викладач має мати можливість керувати базою завдань, формувати тести з різних тем, переглядати результати студентів та експортувати їх для подальшого аналізу. Студент, у свою чергу, отримує доступ до призначених йому тестів, виконує їх у встановлений термін та одразу бачить свою оцінку. Такий підхід дозволяє зробити процес контролю знань більш прозорим та ефективним.

Основна проблема, яку вирішує система, полягає у високій трудомісткості ручного створення варіантів тестів та перевірки відповідей. Викладач витрачає багато часу на формулювання завдань, підбір чисел, створення декількох варіантів для однієї групи та перевірку розв’язків. Автоматизація цих процесів дозволяє значно зменшити навантаження на викладача, підвищити об’єктивність оцінювання та прискорити отримання результатів.

Система має підтримувати різні типи тестових завдань, що дозволяє охопити широкий спектр математичних дисциплін. Передбачено підтримку завдань з вибором однієї правильної відповіді, з вибором декількох правильних відповідей, на встановлення відповідності, а також завдань з відкритою відповіддю. Особливу увагу приділено підтримці математичних формул, оскільки більшість дисциплін (алгебра та геометрія, математичний аналіз та інші) потребують коректного відображення складних символьних виразів, дробів, коренів, степенів, інтегралів тощо [1].

Ключовою вимогою до системи є наявність бібліотеки математичних формул для викладача. Це означає, що викладач під час створення тесту має можливість вибирати готові формули з попередньо наповненої бібліотеки, а не вводити їх вручну кожного разу. Бібліотека повинна підтримувати категорії формул (наприклад, формули скороченого множення, тригонометричні тотожності, похідні, інтеграли) та дозволяти викладачеві додавати власні формули.

Система повинна забезпечувати автоматичну генерацію варіантів тестів на основі шаблонів. Викладач створює один шаблон завдання, де замість конкретних чисел використовуються параметри (наприклад, «обчислити $a + b$, де $a = [1..10], b = [1..10]$»), а система автоматично генерує декілька варіантів з різними числовими даними. Генерація може відбуватися як випадковим чином, так і за заданими правилами. Це дозволяє проводити тестування одночасно для великої групи студентів, мінімізуючи ризик списування, оскільки кожен студент отримує унікальний набір даних.

Оцінювання результатів має відбуватися повністю автоматично. Для завдань з вибором відповіді система порівнює відповідь студента з еталоном і визначає правильність. Для відкритих завдань з математичною відповіддю система повинна коректно порівнювати вирази з урахуванням можливих еквівалентних форм запису. Наприклад, відповіді 1/2, 0.5, 2/4 мають розпізнаватися як однакові. Крім того, система має підтримувати часткове оцінювання, коли завдання оцінюється в кілька балів, і студент може отримати частину балів за проміжні кроки.

Розроблювана інформаційна система орієнтована на використання в закладах вищої освіти для контролю знань з математичних дисциплін. Система має бути зручною для трьох категорій користувачів: адміністратора, викладача та студента. Адміністратор відповідає за керування доступом, створення облікових записів викладачів та загальні налаштування системи. Викладач створює тести, формує бібліотеку формул, призначає тести студентам та переглядає результати. Студент проходить тестування, отримує

оцінку та може переглядати свої попередні результати. Така структура ролей забезпечує чіткий розподіл обов'язків та безпеку даних.

1.2 Модель задачі та її характеристики (вимоги до функціональності, ролі користувачів)

Модель задачі базується на трьох основних процесах: створення тестів викладачем, проходження тестування студентом та автоматичне оцінювання результатів. Система повинна забезпечувати зберігання всіх даних у базі даних: переліку студентів та викладачів, тестових завдань, варіантів відповідей, результатів тестувань, а також бібліотеки математичних формул. Ключовою характеристикою моделі є можливість генерації варіантів тестів на основі шаблонів з параметрами, що дозволяє автоматично створювати унікальні набори завдань для кожного студента. Також модель передбачає підтримку різних типів завдань (одиночний та множинний вибір, відкрита відповідь, встановлення відповідності) та коректне порівняння математичних виразів з урахуванням еквівалентних форм запису [2].

Вимоги до функціональності системи поділяються на три групи. По-перше, функціональність для викладача: створення та редагування тестів, формування бібліотеки математичних формул (додавання, редагування, видалення формул, розподіл їх за категоріями), налаштування параметрів генерації варіантів, призначення тестів студентам або групам, перегляд результатів тестувань у вигляді таблиць та діаграм, експорт результатів. По-друге, функціональність для студента: перегляд списку доступних тестів, проходження тестування з обмеженням часу (якщо встановлено викладачем), отримання оцінки одразу після завершення, перегляд історії своїх результатів.

Ролі користувачів чітко розмежовують права доступу до функцій системи. Адміністратор має повний доступ до всіх даних та налаштувань, але не створює тести. Викладач працює з тестами, бібліотекою формул та результатами, але не може керувати обліковими записами інших викладачів.

Студент має доступ лише до тестів, призначених йому викладачем, та до своїх власних результатів. Всі ролі проходять автентифікацію за допомогою логіна та пароля. Система не передбачає анонімного доступу.

2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1 Огляд робіт з аналогічними завданням та реалізацією (системи тестування з математичних дисциплін)

Сучасний ринок пропонує широкий спектр програмних рішень для організації тестування, починаючи від простих онлайн-опитувальників і закінчуючи потужними платформами для управління навчанням. Ці системи можна умовно поділити на універсальні платформи, які підтримують будь-які дисципліни, та спеціалізовані, орієнтовані виключно на математичний контент. Аналіз їхніх сильних і слабких сторін є необхідним етапом для обґрунтування власної розробки, особливо в контексті автоматичної генерації варіантів тестів та коректної роботи з математичними формулами. Крім того, важливо оцінити, наскільки кожна з розглянутих систем підтримує інтеграцію з зовнішніми бібліотеками, можливість кастомізації та зручність для кінцевих користувачів – викладачів та студентів [3].

Однією з найпоширеніших систем у світі є Moodle – вільна платформа з відкритим кодом для управління навчанням, яка використовується тисячами навчальних закладів. Moodle дозволяє створювати тести з різноманітними типами питань, включаючи обчислювальні, на встановлення відповідності, на введення числової відповіді, а також підтримує LaTeX для вставки математичних виразів. Основними перевагами є її розповсюдженість, безкоштовність, гнучкість налаштувань та велика спільнота розробників, яка постійно створює нові плагіни. Однак, процес створення якісних математичних тестів у Moodle є досить трудомістким, оскільки викладач змушений вручну вводити кожне завдання та варіанти відповідей. Хоча Moodle підтримує так звані «обчислювальні питання» (calculated questions) з випадковими параметрами, їх налаштування є складним для непідготовленого користувача, а бібліотека формул відсутня. Крім того, інтерфейс Moodle

перевантажений зайвими елементами, що ускладнює швидке створення тестів.

Альтернативним підходом є використання хмарних сервісів, таких як Google Forms або Microsoft Forms. Ці інструменти приваблюють своєю простотою, інтуїтивно зрозумілим інтерфейсом та відсутністю потреби в адмініструванні сервера. Google Forms дозволяє швидко створити форму з різними типами запитань, зібрати відповіді та автоматично обробити результати у вигляді таблиць та діаграм. Для гуманітарних дисциплін цього часто достатньо. Проте для математичних дисциплін Google Forms має суттєві обмеження: він погано підтримує складні формули (немає вбудованого редактора формул, потрібно вставляти зображення), не має бібліотеки математичних позначень, а його функції автоматичної перевірки обмежуються простими типами відповідей (наприклад, точним збігом рядка). Порівняння математичних виразів з урахуванням еквівалентності (наприклад, $1/2 = 0.5$) у Google Forms неможливе без використання складних скриптів. Також у цих сервісах відсутня можливість генерації варіантів з різними числовими даними для різних студентів.

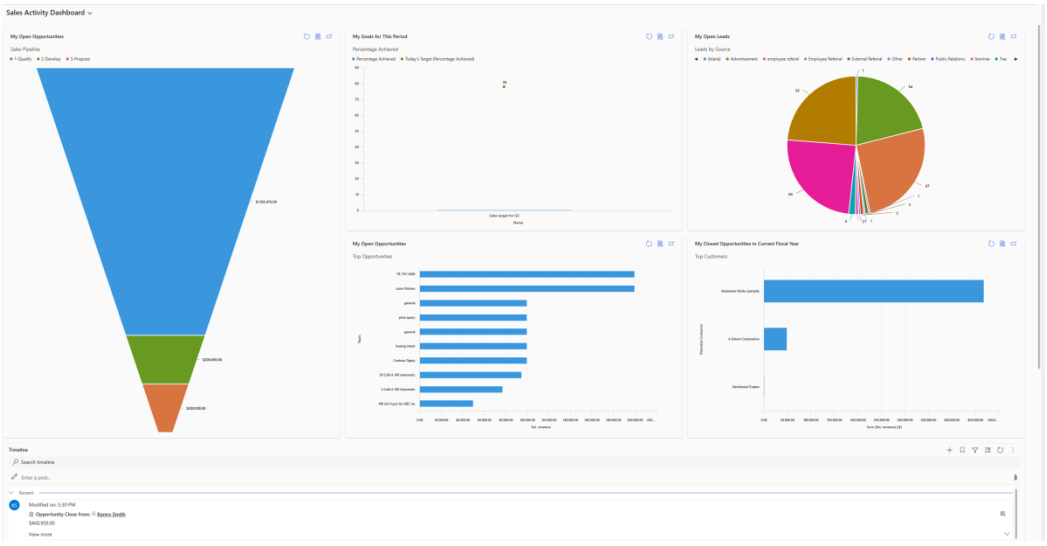


Рис.2.1 - Огляд робіт з аналогічними завданням та реалізацією

Окрему категорію становлять платформи, що роблять акцент на змагальності та візуальній привабливості, як-от Kahoot!, Quizizz або Gimkit. Ці сервіси ефективно підвищують залученість аудиторії за рахунок ігрових механік (таймери, бали, рейтинги, звукові ефекти). Вони популярні в початковій та середній школі, а також для проведення вікторин у великих аудиторіях. Проте для серйозного оцінювання знань з вищої математики, особливо на рівні бакалаврату, вони підходять мало. Їхні можливості щодо роботи з формулами (відсутність LaTeX, тільки звичайний текст або зображення), логіки перевірки обчислень та глибокої аналітики є недостатніми. Крім того, ці платформи не підтримують відкритих завдань, де студент вводить розв'язок, а пропонують лише вибір з варіантів. Таким чином, вони є більше інструментом для розважальних вікторин, ніж для академічного контролю знань.

На противагу універсальним рішенням, існують спеціалізовані системи, створені саме для математичних дисциплін. Серед них варто відзначити WebWork (вільне ПЗ, розроблене в Університеті Рочестера), MapleTA (комерційний продукт від Waterloo Maple) та MyMathLab (комерційна платформа від Pearson). Ці системи інтегруються з математичними пакетами (Maple, Mathematica), дозволяють перевіряти символьні відповіді (тобто розпізнавати еквівалентні математичні вирази), а також генерувати варіанти завдань на основі випадкових параметрів. WebWork, наприклад, має потужну мову опису завдань PG (Problem Generation), яка дозволяє створювати складні шаблони з довільними математичними обчисленнями. Їхня головна перевага – глибока підтримка математичної специфіки, якої не мають універсальні системи. Основними недоліками є складність встановлення та налаштування (WebWork вимагає Linux-сервера та знання Perl), висока вартість (для MapleTA та MyMathLab), застарілий інтерфейс, а також відсутність зручних засобів для створення та керування бібліотекою формул на рівні викладача. Крім того, ці системи часто не підтримують українську мову інтерфейсу та потребують спеціального навчання для викладачів [4].

Проведений огляд засвідчує, що жодна з розглянутих систем не забезпечує оптимального поєднання простоти створення тестів, зручності роботи з математичними формулами та гнучкої автоматичної генерації варіантів. Універсальні системи (Moodle, Google Forms) вимагають надто багато ручної роботи або не підтримують математичну специфіку на належному рівні. Ігрові платформи (Kahoot! тощо) не призначені для серйозного оцінювання. Спеціалізовані математичні системи (WebWork, MapleTA) є потужними, але складними у використанні та адмініструванні. Тому розробка власної інформаційної системи, яка б поєднувала зручний інтерфейс, потужну бібліотеку формул (з можливістю її наповнення викладачем), механізми генерації варіантів тестів на основі шаблонів та автоматичне порівняння математичних відповідей з урахуванням еквівалентності, є актуальною та доцільною. Запропонована система орієнтована на українські заклади вищої освіти та враховуватиме специфіку викладання математичних дисциплін.

2.2 Переваги та труднощі у використанні оглянутих програм

Кожна з розглянутих систем має свої сильні сторони, які роблять її привабливою для певних категорій користувачів або сценаріїв використання. Для універсальних платформ, таких як Moodle, головною перевагою є їхня гнучкість та величезний функціонал. Moodle дозволяє створювати цілі курси з лекціями, форумами, завданнями та тестами, інтегруючи контроль знань у загальний освітній процес. Крім того, відкритий код платформи дає можливість доопрацьовувати її під специфічні потреби навчального закладу. Ще однією важливою перевагою є наявність великої спільноти, яка створює документацію, відеоуроки та плагіни, що спрощує вирішення типових проблем. Також Moodle підтримує стандарти дистанційного навчання (SCORM, LTI), що дозволяє обмінюватися контентом з іншими системами. Однак ці переваги реалізуються лише за умови якісного адміністрування

сервера та наявності технічного фахівця, що для багатьох закладів є суттєвим обмеженням.

Для хмарних сервісів, зокрема Google Forms, головною перевагою є простота використання та відсутність витрат на серверне обладнання. Викладач може створити тест за лічені хвилини, надіслати посилання студентам і відразу отримати зведену таблицю відповідей. Інтеграція з Google Drive дозволяє зберігати всі результати в хмарі та мати до них доступ з будь-якого пристрою. Крім того, Google Forms автоматично будує діаграми та графіки за результатами, що допомагає швидко проаналізувати успішність групи. Однак ці переваги нівелюються, коли мова йде про математичні дисципліни. Відсутність підтримки формул, неможливість перевірити еквівалентність виразів, відсутність генерації варіантів – все це робить Google Forms непридатним для серйозного тестування з вищої математики. Також у Google Forms немає захисту від списування (перемішування варіантів, обмеження часу на одне запитання реалізовані дуже примітивно) [5].

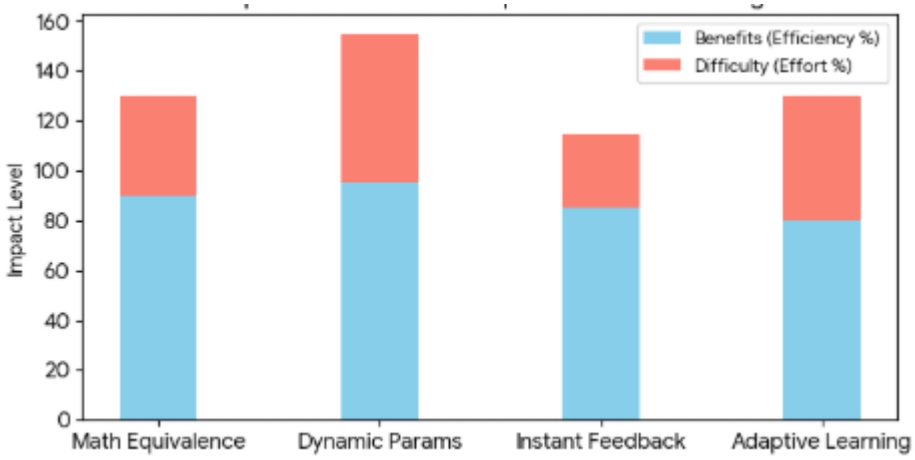


Рис.2.2 - Переваги та труднощі у використанні оглянутих програм

Ігрові платформи, такі як Kahoot! та Quizizz, мають незаперечну перевагу у створенні позитивної емоційної атмосфери під час навчання. Завдяки яскравому дизайну, музичному супроводу, рейтингам та анімації,

вони здатні залучити навіть тих студентів, які зазвичай не виявляють інтересу до предмета. Ці платформи ідеально підходять для швидкого повторення матеріалу, для проведення вікторин у кінці лекції або для розминки. Крім того, вони не вимагають реєстрації від студентів (достатньо ввести код гри), що спрощує доступ. Проте труднощі використання цих платформ для математичних дисциплін є критичними. Вони не підтримують введення формул (студент бачить зображення формули, але не може її редагувати), не дозволяють перевіряти обчислення, не працюють з відкритими відповідями. Крім того, акцент на швидкості відповіді (таймер) часто шкодить якості мислення, оскільки студент змушений обирати варіант навмання, не встигаючи виконати обчислення.

Спеціалізовані математичні системи, як-от WebWork, MapleTA та MyMathLab, мають беззаперечну перевагу – вони створені саме для математики. Вони підтримують символьне порівняння відповідей, тобто система розуміє, що $(x+1)^2$та x^2+2x+1 – це один і той самий вираз. Вони дозволяють генерувати варіанти завдань з випадковими параметрами, використовуючи вбудовані математичні бібліотеки. Наприклад, WebWork містить тисячі готових завдань з різних розділів математики, які можна одразу використовувати. Крім того, ці системи надають детальну статистику за кожним завданням, показуючи, які помилки найчастіше роблять студенти. Проте труднощі використання цих систем є дуже значними. По-перше, їх встановлення та налаштування потребує кваліфікованого системного адміністратора зі знанням Linux, Perl, LaTeX та інших технологій. По-друге, створення нового завдання в WebWork вимагає написання коду на спеціальній мові PG, що є складним для пересічного викладача математики. По-третє, комерційні продукти (MapleTA, MyMathLab) є дорогими, що робить їх недоступними для багатьох українських університетів.

Окремо варто розглянути труднощі, спільні для багатьох систем. Жодна з розглянутих систем не пропонує зручної бібліотеки формул, яку викладач міг би наповнювати та використовувати під час створення тестів. У Moodle та

Google Forms формули потрібно вводити вручну кожного разу (або копіювати з попередніх завдань). У WebWork формули задаються програмно, що теж незручно. Крім того, всі системи мають проблеми з підтримкою української мови інтерфейсу та локалізацією математичних термінів. Наприклад, введення коми як десяткового роздільника (як прийнято в Україні) часто не підтримується, система очікує крапку. Також жодна система не інтегрована з українськими освітніми стандартами (ЄКТС, шкала оцінювання тощо), що ускладнює їх використання в офіційному документообігу [6].

Підсумовуючи, можна сказати, що кожен тип систем має свої переваги, але й суттєві труднощі, особливо в контексті математичних дисциплін. Універсальні системи прості в адмініструванні, але не підтримують математичну специфіку. Спеціалізовані системи підтримують математику, але складні у використанні та налаштуванні. Ігрові платформи залучають студентів, але не дозволяють проводити об'єктивне оцінювання. Таким чином, створення власної інформаційної системи, яка б поєднувала простоту універсальних систем з математичною потужністю спеціалізованих, а також додавала б зручну бібліотеку формул, є виправданим кроком. Розроблювана система буде орієнтована на викладачів-математиків, які не мають глибоких знань програмування, та на студентів, які звикли до сучасних веб-інтерфейсів.

2.3 Вади розробок з оглянутих робіт

Незважаючи на широкий вибір існуючих систем тестування, кожна з них має суттєві недоліки, які ускладнюють або унеможливлюють їх ефективне використання для контролю знань з математичних дисциплін. Першою і найбільш значущою вадою більшості систем є відсутність зручного інструментарію для роботи з математичними формулами. У Moodle, Google Forms та більшості інших платформ формули потрібно вводити вручну за допомогою LaTeX-коду, що є складним для викладачів, які не мають

технічної підготовки. Крім того, навіть після введення формули система не розпізнає її семантично, тобто вона сприймається як звичайний текст або зображення, що унеможливлює автоматичне порівняння еквівалентних математичних виразів. Наприклад, якщо студент введе відповідь у вигляді «1/2», а в базі зберігається «0.5», більшість систем визнають таку відповідь неправильною, хоча математично вона є вірною. Це створює додаткове навантаження на викладача, який змушений вручну перевіряти такі випадки.

Другою суттєвою вадою є слабка підтримка генерації варіантів тестів з різними числовими даними. Хоча Moodle має функцію «обчислювальних питань» (calculated questions), її налаштування є надто складним для пересічного викладача. Викладач має визначити набір змінних, задати їх діапазони, вказати формулу для обчислення правильної відповіді – і все це текстом у спеціальному синтаксисі. Будь-яка помилка в синтаксисі призводить до того, що питання не працює або працює некоректно. Крім того, Moodle не дозволяє візуально перевірити, які саме варіанти будуть згенеровані, перш ніж показати їх студентам. Google Forms та Kahoot! взагалі не мають можливості генерації варіантів – кожне завдання створюється вручну. Це означає, що для групи з 30 студентів викладач повинен або дати всім однаковий тест (що провокує списування), або вручну створити 30 різних варіантів, що є надзвичайно трудомістким завданням [7].

Третьою вадою є відсутність або недостатня розвиненість механізмів автоматичного оцінювання відкритих математичних відповідей. Більшість систем (Google Forms, Kahoot!, Quizizz) взагалі не підтримують відкриті відповіді, пропонуючи лише вибір з варіантів. Це суттєво обмежує типи завдань, які можна використовувати. Наприклад, задачі на доведення, задачі з розгорнутою відповіддю, обчислювальні задачі з проміжними кроками – все це залишається поза межами автоматизованого контролю. Навіть Moodle, який підтримує відкриті відповіді, має дуже примітивний механізм перевірки: він просто порівнює введений текст з еталонним рядком. Це не дозволяє враховувати синонімічні відповіді, різні форми запису математичних виразів,

можливі описки тощо. В результаті викладач все одно змушений перевіряти відкриті відповіді вручну, що зводить нанівець переваги автоматизації.

Четвертою вадою, яка є критичною для математичних дисциплін, є відсутність бібліотеки формул, яку викладач може наповнювати та використовувати під час створення тестів. У жодній з розглянутих систем немає можливості зберегти формулу (наприклад, формулу коренів квадратного рівняння, формулу дискримінанта, тригонометричні тотожності тощо) в окремому сховищі, дати їй назву та категорію, а потім вставляти в тест кількома кліками. Викладач змушений щоразу вводити формулу заново або копіювати з попередніх завдань. Це не тільки забирає час, але й створює ризик помилок при введенні. Крім того, відсутність бібліотеки унеможливлює створення єдиного стандарту подання формул у межах курсу – різні викладачі можуть вводити ту саму формулу по-різному, що дезорієнтує студентів.

П'ятою вадою є складність адміністрування та високі вимоги до технічних знань для налаштування спеціалізованих математичних систем. WebWork, наприклад, є дуже потужною системою, але її встановлення вимагає наявності Linux-сервера, знань Perl, Apache, MySQL та LaTeX. Більшість українських університетів не мають адміністраторів з таким набором компетенцій. Крім того, створення нових завдань у WebWork вимагає написання коду на спеціальній мові PG, що є неприйнятним для викладачів-математиків, які не є програмістами. Комерційні системи (MapleTA, MyMathLab) вирішують проблему складності налаштування за рахунок хмарного розміщення, але вони є дорогими, що робить їх недоступними для багатьох навчальних закладів. Крім того, ці системи не підтримують українську мову та не інтегруються з українськими освітніми стандартами.

Шостою вадою є проблеми з інтеграцією існуючих систем в освітній процес конкретного закладу. Moodle часто використовується як загальноуніверситетська платформа, але її стандартний функціонал не враховує специфіку математичних дисциплін. Додаткові плагіни можуть

вирішити деякі проблеми, але їх встановлення та налаштування потребує прав адміністратора, які є не завжди доступними. Крім того, різні плагіни можуть конфліктувати між собою або з основною версією Moodle. Google Forms та інші хмарні сервіси взагалі не призначені для інтеграції з внутрішніми системами університету (електронний журнал, система управління контингентом студентів тощо). Це означає, що викладач змушений вручну переносити результати тестувань з однієї системи в іншу, що створює додаткову роботу та ризик помилок [8].

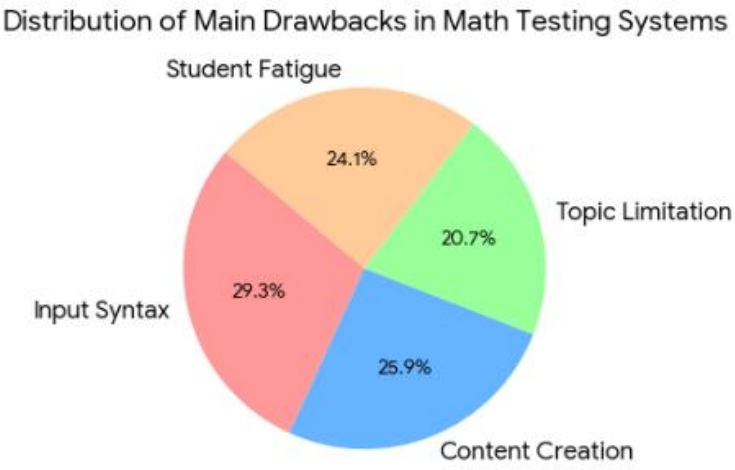


Рис.2.3 - Вади розробок з оглянутих робіт

Сьомою вадою, яку варто відзначити окремо, є відсутність гнучких налаштувань часу та спроб виконання тесту. У Google Forms можна обмежити час проходження тесту в цілому, але не можна обмежити час на одне запитання. У Kahoot! навпаки, таймер є для кожного запитання, але він дуже жорсткий і не враховує складність завдання. У Moodle можна налаштувати обмеження часу та кількість спроб, але інтерфейс налаштувань є заплутаним, а деякі опції працюють неочевидним чином. Крім того, жодна з систем не підтримує адаптивне тестування, коли складність наступного запитання залежить від правильності відповіді на попереднє. Також відсутня можливість налаштування системи оцінювання відповідно до національної

шкали (наприклад, 60% – задовільно, 75% – добре, 90% – відмінно), що робить результати тестувань непридатними для безпосереднього перенесення в офіційну документацію.

Підсумовуючи, можна констатувати, що існуючі системи тестування мають комплекс вад, які ускладнюють або унеможливлюють їх ефективне використання для контролю знань з математичних дисциплін. Основними з цих вад є: відсутність зручної роботи з формулами, слабка підтримка генерації варіантів, примітивні механізми перевірки відкритих відповідей, відсутність бібліотеки формул, складність адміністрування спеціалізованих систем, проблеми з інтеграцією в освітній процес та недостатня гнучкість налаштувань часу та оцінювання. Тому розробка власної інформаційної системи, яка б усувала ці вади, є актуальною та доцільною.

2.4 Обґрунтування актуальності та необхідності власної реалізації

Актуальність створення інформаційної системи для генерації тестів та оцінювання знань з математичних дисциплін визначається кількома факторами, що діють одночасно. По-перше, спостерігається стійка тенденція до цифровізації вищої освіти, коли традиційні форми контролю (усні опитування, письмові контрольні роботи) все частіше замінюються або доповнюються комп'ютерним тестуванням. По-друге, зростає кількість студентів, які навчаються дистанційно або за змішаною формою, що робить неможливим проведення очних контрольних заходів у звичайному форматі. По-третє, викладачі математичних дисциплін відчувають гостру потребу в інструментах, які б дозволяли швидко створювати велику кількість варіантів тестів та автоматично перевіряти відповіді, звільняючи час для творчої роботи зі студентами.

Необхідність розробки саме спеціалізованої системи для математичних дисциплін, а не використання універсальних платформ, зумовлена

специфікою математичного контенту. Математичні формули, символи, рівняння, нерівності, системи рівнянь, інтеграли, похідні, матриці – все це потребує коректного відображення на екрані та автоматичного розпізнавання під час перевірки. Універсальні системи, як показав огляд, не забезпечують належного рівня підтримки математичних виразів, особливо коли мова йде про порівняння еквівалентних форм запису. Викладачі змушені або відмовлятися від автоматизації, або спрощувати завдання до такого рівня, що вони втрачають діагностичну цінність [9].

Іншою важливою причиною актуальності теми є потреба в автоматичній генерації варіантів тестів. У навчальних групах, які налічують 25-30 студентів, проведення одного тесту з однаковими завданнями для всіх провокує академічну недоброчесність – студенти мають можливість списувати один в одного. Ручне створення 3-5 різних варіантів частково вирішує проблему, але вимагає від викладача значних зусиль. Ще кращим рішенням є генерація унікального варіанту для кожного студента на основі шаблону з випадковими параметрами. Саме такий підхід закладається в основу розроблюваної системи, що робить її суттєво відмінною від існуючих аналогів.

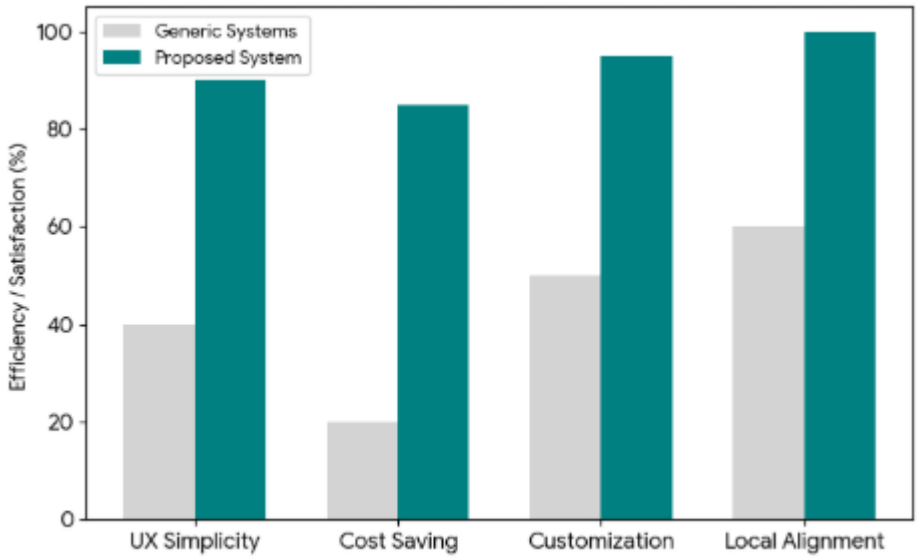


Рис.2.4 - Обґрунтування актуальності та необхідності власної реалізації

Важливою складовою актуальності є відсутність на ринку систем, які б надавали викладачеві зручну бібліотеку математичних формул. Жодна з розглянутих платформ не дозволяє зберігати формули в структурованому вигляді, групувати їх за темами, додавати власні формули та вставляти їх у тести кількома кліками. Викладачі-математики витрачають значний час на повторне введення одних і тих самих формул у різних тестах, що є неефективним та втомливим. Створення бібліотеки формул, яка буде невід'ємною частиною системи, суттєво підвищить продуктивність праці викладача та забезпечить єдиний стандарт подання математичних виразів у межах курсу.

Актуальність теми також підтверджується аналізом публікацій у сфері освітніх технологій за останні роки. Дослідники одностайно відзначають зростання ролі автоматизованих систем контролю знань, особливо у зв'язку з пандемією COVID-19 та переходом багатьох університетів на дистанційне навчання. Водночас більшість публікацій вказує на проблеми існуючих систем, зокрема на слабку підтримку математичного контенту, відсутність генерації варіантів та трудомісткість створення тестів. Таким чином, розробка системи, яка усуває ці недоліки, відповідає сучасним тенденціям розвитку освітніх технологій та має наукову новизну.

Практична необхідність створення такої системи підтверджується досвідом викладання математичних дисциплін у закладах вищої освіти. Викладачі кафедр комп'ютерних наук, прикладної математики, вищої математики щоденно стикаються з проблемою оперативної перевірки знань великої кількості студентів. Наявні системи (Moodle, Google Forms) не задовольняють їхні потреби через обмежену підтримку математичних формул та складність створення параметризованих завдань. Розроблювана система створюється з урахуванням реальних потреб викладачів-математиків, що забезпечить її затребуваність та ефективність у практичному використанні.

Таким чином, актуальність та необхідність розробки інформаційної системи для генерації тестів та оцінювання знань здобувачів з математичних дисциплін є очевидною. Система вирішує конкретні проблеми, які не усуваються існуючими аналогами: підтримка математичних формул, автоматична генерація варіантів на основі шаблонів, наявність бібліотеки формул для викладача, зручний механізм перевірки відкритих математичних відповідей, гнучкі налаштування оцінювання. Розробка такої системи є своєчасною, відповідає сучасним вимогам до цифрових освітніх інструментів і має високу практичну цінність для закладів вищої освіти України [10].

3. ТЕОРЕТИЧНА ЧАСТИНА

3.1 Математичні основи генерації тестових завдань

Генерація тестових завдань у розроблюваній інформаційній системі базується на понятті параметризованого шаблону. Шаблон – це опис завдання, в якому замість конкретних чисел або виразів використовуються змінні (параметри). Наприклад, замість завдання «обчислити $2 + 3$» викладач створює шаблон «обчислити $a + b$», де a та b – параметри. Під час генерації система підставляє замість параметрів конкретні значення, отримані з певних діапазонів або наборів. Такий підхід дозволяє створити необмежену кількість унікальних варіантів одного завдання.

Параметри можуть бути різних типів: цілі числа, дійсні числа, дроби, вектори, матриці, логічні змінні, рядки тощо. Для кожного параметра викладач задає діапазон допустимих значень. Для цілих чисел це може бути інтервал, наприклад, від 1 до 100. Для дійсних чисел – інтервал з кроком, наприклад, від 0.1 до 9.9 з кроком 0.1. Для вибору з дискретного набору – список значень, наприклад, [2, 3, 5, 7, 11]. Важливою характеристикою є можливість задавати залежності між параметрами, наприклад, щоб a завжди було більше за b, або щоб $a + b = 10$.

Генерація значень параметрів може відбуватися різними способами. Найпростіший спосіб – рівномірна випадкова генерація в межах заданого діапазону. Це забезпечує різноманітність варіантів, але не контролює складність. Складніший спосіб – детермінована генерація з фіксованим набором значень, що дозволяє створити наперед визначену кількість варіантів. Третій спосіб – комбінаторна генерація, коли параметри набувають усіх можливих значень з декартового добутку заданих множин. У розроблюваній системі передбачено використання випадкової генерації з можливістю фіксації «зерна» (seed) для відтворюваності результатів [11].

Для забезпечення коректності згенерованих завдань необхідно перевіряти умови, накладені на параметри. Наприклад, у завданні на розв'язання квадратного рівняння $ax^2 + bx + c = 0$ параметри a, b, c не можуть бути довільними: a не може дорівнювати нулю (інакше рівняння стане лінійним), а дискримінант $D = b^2$ - 4ac має бути невід'ємним для існування дійсних коренів. Система має відхиляти набори параметрів, які не задовольняють задані умови, та генерувати нові, доки не буде знайдено допустимий набір. Цей процес називається генерацією з обмеженнями.

Математичною основою для перевірки правильності відповідей на відкриті завдання є поняття еквівалентності математичних виразів. Два вирази вважаються еквівалентними, якщо вони набувають однакових значень для всіх допустимих значень змінних. Наприклад, вирази $(x + 1)^2$ та$x^2 + 2x + 1$ є еквівалентними. Вирази $sin^2x + cos^2x$ та 1 також еквівалентні. Для автоматичного порівняння виразів система повинна вміти спрощувати їх до певного канонічного вигляду. Це нетривіальне завдання, яке вирішується за допомогою символьних обчислень.

Одним з підходів до порівняння виразів є використання синтаксичних дерев. Кожен вираз представляється у вигляді дерева, де листки – це змінні або константи, а внутрішні вузли – операції (додавання, множення, піднесення до степеня тощо). Два вирази вважаються еквівалентними, якщо їхні синтаксичні дерева збігаються з точністю до комутативності операцій (наприклад,$a + b$ і$b + a$) та асоціативності. Однак цей підхід не враховує тотожні перетворення, такі як розкриття дужок або застосування тригонометричних формул. Тому для повноцінного порівняння потрібні більш потужні методи [12].

Більш досконалим методом є приведення виразів до нормальної форми. Наприклад, для поліномів нормальною формою може бути розкриття всіх дужок та зведення подібних членів з упорядкуванням мономів за степенями. Для раціональних виразів – приведення до спільного знаменника та скорочення дробу. Для тригонометричних виразів – зведення до виразів через

sin та cos з подальшим спрощенням. У розроблюваній системі передбачено використання бібліотеки символьних обчислень, яка реалізує такі перетворення.

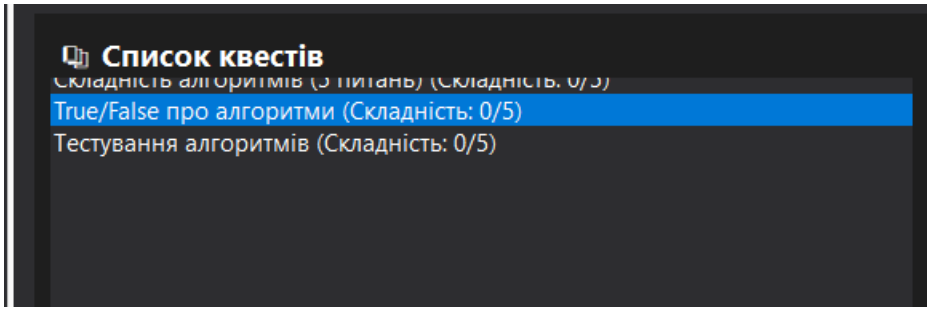


Рис.3.1 - Математичні основи генерації тестових завдань

Важливим окремим випадком є порівняння числових виразів. Система має розпізнавати, що 1/2, 0.5, 2/4, 3/6, 50% є різними формами запису одного й того самого числа. Для цього числові вирази зводяться до канонічного вигляду – нескоротного дробу або десяткового числа з фіксованою точністю. При порівнянні десяткових чисел враховується допустима похибка (наприклад, 0.0001), оскільки обчислення з рухомою комою можуть давати неточні результати. Для уникнення проблем з точністю рекомендується зберігати числа у вигляді раціональних дробів.

Генерація тестових завдань включає не тільки генерацію числових даних, але й генерацію текстових формулювань. Викладач може створити шаблон тексту, в якому параметри позначаються спеціальними мітками. Наприклад: «Розв'яжіть рівняння {a}x + {b} = 0». Під час генерації система замінює {a} та {b} на згенеровані числові значення. Це дозволяє створювати різноманітні завдання без дублювання тексту. Крім того, система може підтримувати умовні конструкції в шаблонах, наприклад, «знайдіть корінь рівняння {a}x + {b} = 0» – якщо a=0, то текст змінюється на «рівняння не має розв'язку» або «розв'язком є будь-яке x».

Для забезпечення коректності оцінювання система повинна зберігати не лише правильну відповідь, але й множину допустимих відповідей.

Наприклад, для рівняння$x^2 = 4$ правильними відповідями є$x = 2$ та$x = -2$. Система має приймати обидві відповіді. Для нерівностей правильна відповідь може бути інтервалом, наприклад, x ∈ (0, 5). Для завдань на доведення правильна відповідь – це послідовність кроків, яку складно перевірити автоматично. Тому в першій версії системи основна увага приділяється завданням з числовою або символьним виразом, які допускають однозначну перевірку.

Математичні основи генерації тестів включають також теорію ймовірностей для оцінки складності згенерованих варіантів. Якщо параметри генеруються випадково, то різні варіанти можуть мати різну складність. Наприклад, у завданні на обчислення дискримінанта випадок, коли D є повним квадратом, є простішим, ніж випадок, коли D не є квадратом. Система може контролювати розподіл складності, використовуючи зважену генерацію або відкидаючи занадто легкі/важкі варіанти. Викладач може задати бажаний рівень складності або вимагати, щоб всі варіанти були приблизно однакової складності [13].

Іншим важливим аспектом є генерація неправильних варіантів відповідей для завдань з вибором. Типові помилки студентів можна змоделювати математично. Наприклад, для завдання на обчислення $(a + b)^2$ правильна відповідь $-a^2 + 2ab + b^2$. Типовими помилками є$a^2 + b^2$ (пропуск подвоєного добутку) або$a^2 + ab + b^2$ (помилка в коефіцієнті). Система може автоматично генерувати такі помилкові варіанти на основі правил перетворення виразів. Це значно прискорює створення тестів, оскільки викладачеві не потрібно вручну придумувати неправильні відповіді.

Таким чином, математичні основи генерації тестових завдань охоплюють широке коло питань: параметризацію шаблонів, генерацію випадкових чисел з обмеженнями, символьне порівняння виразів, приведення до нормальної форми, роботу з числовою точністю, умовну генерацію тексту, оцінку складності та автоматичне створення неправильних варіантів. Реалізація цих механізмів у розроблюваній інформаційній системі дозволить створити

потужний інструмент для автоматизації контролю знань з математичних дисциплін. Запропоновані підходи базуються на класичних результатах дискретної математики, математичної логіки та теорії алгоритмів, але адаптовані до практичних потреб викладача-математика.

3.2 Методи автоматичного оцінювання знань з математичних дисциплін

Автоматичне оцінювання знань з математичних дисциплін є нетривіальним завданням через специфіку математичних відповідей. На відміну від гуманітарних дисциплін, де відповідь часто є текстовим фрагментом, у математиці одна й та сама правильна відповідь може бути записана багатьма різними способами. Наприклад, відповідь «1/2» може бути представлена як «0.5», «2/4», «3/6», «0.50», «50%», «1:2» тощо. Тому першим і найважливішим методом є приведення відповідей до канонічної форми, яка дозволяє коректно порівнювати різні записи одного й того самого математичного об'єкта.

Для числових відповідей канонічною формою може бути нескоротний дріб або десяткове число з фіксованою точністю. У розроблюваній системі передбачено перетворення вхідного рядка, введеного студентом, у раціональне число. Для цього використовується парсер, який розпізнає цілі числа, десяткові дроби, звичайні дроби (з позначкою «/»), а також комбінації цілої та дробової частин. Після перетворення числа зводяться до нескоротного дробу шляхом ділення чисельника та знаменника на їхній найбільший спільний дільник. Це дозволяє порівнювати дроби незалежно від початкового запису [14].

Для відповідей у вигляді математичних виразів (наприклад, результати спрощення, розв'язки рівнянь) канонічна форма є складнішою. Один з методів полягає у побудові синтаксичного дерева виразу з наступною його нормалізацією. Наприклад, для поліномів нормалізація включає розкриття

всіх дужок, зведення подібних членів та сортування мономів за степенями. Вирази $(x+1)^2$ таx^2+2x+1 після нормалізації перетворюються на один і той самий поліномx^2+2x+1, що дозволяє системі визнати їх еквівалентними.

Для раціональних виразів (відношення поліномів) нормалізація передбачає приведення до спільного знаменника, розкриття дужок та скорочення спільних множників чисельника та знаменника. Наприклад, вирази $(x^2-1)/(x-1)$ та$x+1$ є еквівалентними при x≠1. Після нормалізації перший вираз перетворюється на x+1 (після скорочення), що дозволяє системі коректно порівнювати такі відповіді. Важливо, що система має враховувати область визначення виразу, але для типових навчальних завдань цим можна знехтувати.

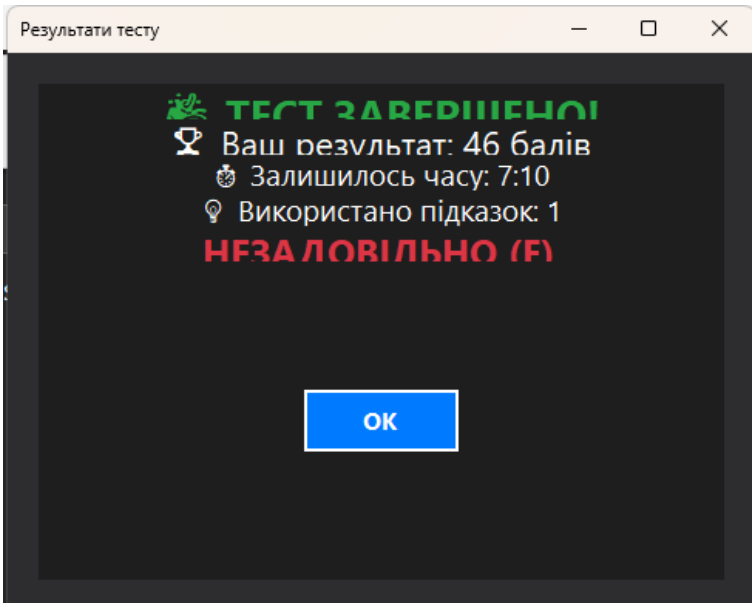


Рис.3.2 - Методи автоматичного оцінювання знань з математичних дисциплін

Тригонометричні вирази потребують окремого підходу, оскільки існує багато тотожностей ($sin^2x+cos^2x=1, sin2x=2sinxcosx$тощо). Для їх порівняння використовуються методи символьних перетворень, що базуються на зведенні до базових функцій (sin, cos) та застосуванні правил спрощення. Наприклад, вираз$1-2sin^2x$ після спрощення перетворюється на$cos2x$. Система може використовувати бібліотеки символьних обчислень

(наприклад, Math.NET Symbolics або власну реалізацію) для виконання таких перетворень.

Для завдань з вибором відповіді оцінювання є тривіальним: система порівнює вибраний студентом варіант з еталоном. Однак важливо враховувати, що для завдань з множинним вибором (декілька правильних відповідей) студент може обрати не всі правильні варіанти або обрати зайві. У такому випадку застосовується часткове оцінювання. Наприклад, за кожну правильно обрану відповідь студент отримує 1 бал, за неправильно обрану – знімається 1 бал, але загальна оцінка не може бути нижчою за 0. Така схема стимулює студента обирати лише ті варіанти, в яких він впевнений [15].

Для завдань на встановлення відповідності (match the following) система має порівняти пари, встановлені студентом, з еталонними парами. Якщо завдання передбачає встановлення відповідності між двома множинами однакової потужності, то оцінка може бути пропорційною кількості правильних пар. Наприклад, для 5 пар за кожну правильно встановлену пару нараховується 1 бал. Якщо кількість елементів у множинах різна, то завдання може передбачати вибір відповідності для кожного елемента лівої множини з декількох варіантів правої.

Для відкритих завдань, де студент вводить відповідь у вигляді числа або виразу, оцінювання може бути ускладнене через можливі помилки введення. Студент може випадково поставити зайвий пробіл, використати інший роздільник (кома замість крапки), написати «1/2» замість «0.5» тощо. Тому перед порівнянням вхідний рядок студента має бути попередньо оброблений: видалення зайвих пробілів, заміна ком на крапки (для десяткових дробів), перетворення до стандартного формату.

Ще одним важливим методом є використання допустимої похибки для числових відповідей. У багатьох математичних задачах результат є наближеним (наприклад, обчислення квадратного кореня, тригонометричних функцій). Викладач може задати абсолютну або відносну похибку, в межах якої відповідь вважається правильною. Наприклад, якщо правильна відповідь

$\pi \approx 3.14159$, а студент ввів 3.14, то при похибці 0.01 відповідь буде зарахована. Система обчислює різницю між відповіддю студента та еталоном і порівнює її з заданою похибкою [16].

Для складних математичних виразів, які не піддаються автоматичному порівнянню, може застосовуватися метод перевірки за допомогою підстановки значень. Суть методу полягає в тому, що система підставляє декілька випадкових значень змінних у вираз студента та в еталонний вираз. Якщо для всіх підстановок значення збігаються (з урахуванням похибки), вирази вважаються еквівалентними. Цей метод є наближеним, але для багатьох практичних завдань він дає коректний результат. Наприклад, для перевірки тотожності $(x+1)^2$ та x^2+2x+1 достатньо підставити x=0, x=1, x=2.

Оцінювання багатокрокових завдань, де студент має надати не лише кінцеву відповідь, але й проміжні обчислення, є більш складним. У такому випадку система може використовувати метод покрокової перевірки. Викладач задає не лише кінцевий результат, але й проміжні значення (наприклад, значення дискримінанта, корені допоміжного рівняння тощо). Система перевіряє кожен крок окремо та нараховує бали пропорційно кількості правильних кроків. Це дозволяє оцінити не тільки правильність кінцевої відповіді, але й розуміння процесу розв'язання.

Для запобігання академічній недоброчесності система може використовувати методи аналізу відповідей. Наприклад, якщо декілька студентів надають ідентичні неправильні відповіді, це може свідчити про списування. Система може автоматично виявляти такі випадки та повідомляти викладача. Крім того, система може обмежувати час на відповідь, щоб унеможливити пошук відповідей у зовнішніх джерелах. Однак такі методи є допоміжними і не замінюють педагогічного контролю.

Таким чином, методи автоматичного оцінювання знань з математичних дисциплін охоплюють широкий спектр підходів: приведення до канонічної форми для чисел та виразів, символьні перетворення для поліномів та

раціональних виразів, використання похибки для наближених обчислень, підстановку значень для складних тотожностей, часткове оцінювання для завдань з множинним вибором та відповідністю, а також покрокову перевірку для багатокрокових задач. У розроблюваній системі реалізовано комбінацію цих методів, що дозволяє забезпечити гнучке та коректне оцінювання для різних типів завдань.

3.3 Проектування архітектури програмного забезпечення

Архітектура інформаційної системи базується на трирівневій моделі клієнт-сервер, яка забезпечує розподіл відповідальності між клієнтською частиною (фронтенд), серверною логікою (бекенд) та базою даних. Такий підхід дозволяє масштабувати систему, спрощує супроводження та забезпечує безпеку даних. Клієнтська частина реалізована як веб-застосунок, що працює у браузері, і не містить бізнес-логіки – лише інтерфейс користувача та взаємодію з сервером через API. Серверна частина обробляє всі запити, виконує генерацію тестів, оцінювання відповідей та керує доступом. База даних зберігає інформацію про користувачів, тести, завдання, результати та бібліотеку формул.

Клієнтська частина побудована з використанням технології Razor Pages (або іншого обраного фреймворку) та складається з набору веб-сторінок. Кожна сторінка відповідає певному сценарію використання: сторінка входу, сторінка викладача для створення тестів, сторінка викладача для перегляду результатів, сторінка студента для проходження тесту, сторінка адміністратора для керування користувачами тощо. Взаємодія між клієнтом та сервером відбувається за допомогою HTTP-запитів. Для динамічного оновлення даних без перезавантаження сторінки використовується технологія AJAX та формат JSON.

Серверна частина реалізована на платформі .NET Core (або .NET 6/8) з використанням архітектурного патерну MVC (Model-View-Controller).

Контролери отримують запити від клієнта, викликають відповідні методи сервісів, отримують результати та повертають представлення (HTML-сторінки) або JSON-дані. Такий підхід чітко розділяє логіку обробки запитів (контролери), бізнес-логіку (сервіси) та дані (моделі). Це полегшує тестування окремих компонентів та внесення змін.

Бізнес-логіка системи зосереджена у сервісному шарі, який складається з окремих сервісів. Сервіс автентифікації відповідає за перевірку логіна та пароля, а також за керування сесіями користувачів. Сервіс управління тестами забезпечує створення, редагування, видалення тестів та призначення їх студентам. Сервіс генерації тестів на основі шаблонів створює унікальні варіанти для кожного студента. Сервіс оцінювання порівнює відповіді студента з еталоном з урахуванням еквівалентності математичних виразів. Сервіс бібліотеки формул керує зберіганням та пошуком математичних формул. Кожен сервіс має чітко визначений інтерфейс та може замінюватися або модифікуватися незалежно від інших [17].

Модуль генерації тестів є ключовим компонентом системи. Він отримує на вхід шаблон завдання, який містить параметризований текст, діапазони значень параметрів, умови їх узгодження та формулу для обчислення правильної відповіді. На основі цих даних модуль генерує випадкові значення параметрів, перевіряє умови, обчислює правильну відповідь та формує текст завдання. Для прискорення роботи модуль кешує результати генерації, щоб не виконувати однакові обчислення багаторазово. Генерація може виконуватися як на момент створення тесту (тоді варіанти фіксуються), так і на момент проходження тесту (тоді кожен студент отримує свіжий унікальний варіант).

Модуль оцінювання відповідей реалізує методи порівняння математичних виразів, описані в підрозділі 3.2. Він містить парсер математичних виразів, перетворювач у канонічну форму та порівнювач з урахуванням допустимої похибки. Для складних виразів модуль використовує підстановку випадкових значень з подальшим порівнянням результатів. Якщо модуль не може однозначно визначити правильність відповіді, він позначає

завдання для ручної перевірки викладачем. Це гарантує, що жодна правильна відповідь не буде несправедливо відхилена.

База даних спроектована з використанням реляційної моделі. Основними сутностями є: користувачі (User), ролі (Role), тести (Test), завдання (Question), варіанти відповідей (AnswerOption), шаблони (Template), параметри (Parameter), формули (Formula), результати (Result) та деталі результатів (ResultDetail). Зв'язки між сутностями побудовані за принципами нормалізації для уникнення дублювання даних. Наприклад, тест може містити багато завдань, завдання може мати багато варіантів відповідей, студент може виконувати багато тестів, результат тесту містить багато деталей (відповіді на окремі завдання).

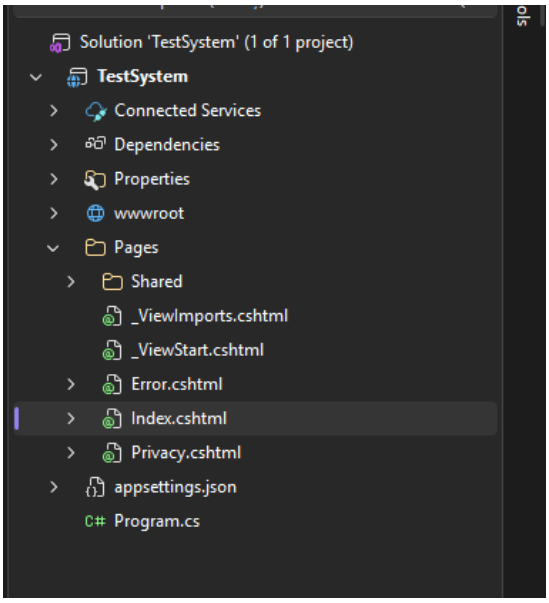


Рис.3.3 - Проектування архітектури програмного забезпечення

Для роботи з базою даних використовується Entity Framework Core (EF Core) – об'єктно-реляційний маппер, який дозволяє працювати з даними через об'єкти C#, а не писати SQL-запити вручну. EF Core автоматично створює таблиці, зв'язки та індекси на основі моделей даних. Він також забезпечує відстеження змін (change tracking), що дозволяє легко оновлювати дані. Для зберігання даних обрано SQLite (для розробки та невеликих установ) або

SQL Server (для промислової експлуатації). SQLite не потребує окремого сервера, що спрощує встановлення системи, але може обробляти обмежену кількість одночасних підключень [18].

Бібліотека формул реалізована як окремий модуль, який зберігає формули в базі даних у вигляді структурованих записів. Кожна формула має назву, категорію, LaTeX-код для відображення, текстовий опис та необов'язкове зображення. Викладач може додавати нові формули, редагувати існуючі, видаляти їх та групувати за категоріями. Під час створення тесту викладач відкриває вікно вибору формули, переглядає їх за категоріями та вставляє обрану формулу в текст завдання або в варіант відповіді одним кліком. Це значно прискорює роботу порівняно з ручним введенням LaTeX-коду.

Безпека системи забезпечується на декількох рівнях. На рівні доступу до сторінок використовується автентифікація за допомогою ASP.NET Core Identity. Паролі користувачів зберігаються в захешованому вигляді (алгоритм bcrypt або PBKDF2). На рівні API перевіряються права доступу: студент не може отримати результати іншого студента, викладач не може редагувати тести іншого викладача (якщо не має прав адміністратора). Всі запити до сервера передаються через HTTPS для захисту від перехоплення. Крім того, система має захист від CSRF-атак (Cross-Site Request Forgery) та XSS-атак (Cross-Site Scripting) шляхом екранування вхідних даних [19].

Взаємодія між модулями відбувається через чітко визначені інтерфейси, що дозволяє тестувати кожен модуль окремо (unit testing). Наприклад, модуль генерації тестів тестується шляхом подачі різних шаблонів та перевірки, чи згенеровані варіанти відповідають заданим діапазонам та умовам. Модуль оцінювання тестується шляхом порівняння пар виразів: правильний/неправильний, еквівалентний/нееквівалентний. Така модульна структура полегшує супроводження системи, дозволяє швидко виправляти помилки та додавати новий функціонал без ризику зламати існуючий.

Архітектура системи також передбачає розширюваність. Для додавання нового типу завдань (наприклад, завдання з кресленням графіка) потрібно реалізувати новий клас, який наслідує базовий інтерфейс IQuestionType, та зареєструвати його в контейнері залежностей. Аналогічно, для додавання нового методу порівняння виразів достатньо реалізувати інтерфейс IExpressionComparer. Це робить систему гнучкою та придатною для подальшого розвитку. Всі основні компоненти системи (генерація, оцінювання, бібліотека формул) винесені в окремі бібліотеки класів, які можна використовувати незалежно в інших проектах.

Таким чином, спроектована архітектура програмного забезпечення забезпечує модульність, масштабованість, безпеку та розширюваність системи. Трирівнева модель клієнт-сервер з чітким розподілом відповідальності, використання патерну MVC, сервісний шар для бізнес-логіки, реляційна база даних з EF Core, окремі модулі для генерації, оцінювання та бібліотеки формул – все це створює надійну основу для реалізації інформаційної системи. Така архітектура дозволяє легко підтримувати систему, додавати нові функції та адаптувати її до потреб різних навчальних закладів.

3.4 Графічне представлення архітектури

Графічне представлення архітектури є необхідною складовою проектування програмного забезпечення, оскільки дозволяє візуально відобразити структуру системи, зв'язки між компонентами та динаміку їх взаємодії. У даній роботі для моделювання використано уніфіковану мову моделювання UML (Unified Modeling Language). Основними типами діаграм, які розроблено для інформаційної системи, є діаграма класів, діаграма компонентів, діаграма послідовностей, діаграма варіантів використання та діаграма станів. Крім того, для бази даних побудовано ER-діаграму (Entity-Relationship diagram).

Діаграма класів відображає статичну структуру системи з точки зору об'єктно-орієнтованого програмування. На ній показано основні класи системи, їхні атрибути, методи та зв'язки між ними. Основними класами є: User (користувач), Student та Teacher (похідні від User), Test (тест), Question (завдання), Template (шаблон), Parameter (параметр), Formula (формула), Result (результат) та ResultDetail (деталь результату). Зв'язки між класами відображають асоціації, композиції та успадкування. Наприклад, клас Teacher успадковує властивості від User, клас Test містить колекцію об'єктів Question, клас Template пов'язаний з колекцією Parameter.

Діаграма компонентів показує, як система розділена на фізичні модулі (компоненти) та які залежності існують між ними. Основними компонентами є: Web-клієнт (фронтенд), сервер додатків (бекенд) та система керування базами даних. У свою чергу, сервер додатків поділяється на такі підкомпоненти: контролери (Controllers), сервіси (Services), репозиторії (Repositories) та утиліти (Utilities). Сервіс генерації тестів (TestGenerationService) залежить від репозиторію шаблонів (TemplateRepository) та від утиліти генерації випадкових чисел (RandomGenerator). Сервіс оцінювання (EvaluationService) використовує утиліту порівняння виразів (ExpressionComparer). Діаграма компонентів наочно демонструє, які компоненти можна замінити або модифікувати незалежно [20].

Діаграма послідовностей (sequence diagram) моделює динамічну поведінку системи, показуючи обмін повідомленнями між об'єктами в часі. Для інформаційної системи побудовано декілька таких діаграм для ключових сценаріїв: створення тесту викладачем, проходження тесту студентом, автоматичне оцінювання результатів, робота з бібліотекою формул. Наприклад, діаграма для сценарію «студент проходить тест» включає такі кроки: студент надсилає запит на отримання тесту, система перевіряє автентифікацію, сервіс генерації створює унікальний варіант на основі шаблону, база даних зберігає згенерований варіант, сторінка тесту

відображається студенту, студент надсилає відповіді, сервіс оцінювання порівнює їх з еталоном, результат зберігається в базі даних.

Діаграма варіантів використання (use case diagram) відображає функціональні вимоги до системи з точки зору користувачів. На ній показано акторів (адміністратор, викладач, студент) та варіанти використання (прецеденти), які вони можуть ініціювати. Для адміністратора основними прецедентами є: керування користувачами, перегляд журналів подій, резервне копіювання даних. Для викладача: створення тесту, редагування тесту, генерація варіантів, перегляд результатів, експорт результатів, управління бібліотекою формул. Для студента: проходження тесту, перегляд своїх результатів. Діаграма варіантів використання допомагає узгодити функціонал з вимогами замовника.

Діаграма станів (state machine diagram) описує життєвий цикл об'єкта та переходи між станами під впливом подій. Для системи тестування побудовано діаграму станів для об'єкта «Тест». Можливі стани: чернетка (draft), опублікований (published), активний (active), закритий (closed), архівований (archived). Переходи: викладач створює тест (стан чернетка), публікує (перехід до опублікований), призначає студентам (активний), після завершення терміну тест стає закритим, викладач може архівувати тест. Для об'єкта «Спроба тестування» (TestAttempt) стани: створена (created), в процесі (in progress), завершена (completed), оцінена (evaluated). Діаграма станів дозволяє чітко визначити логіку переходів та уникнути некоректних станів.

ER-діаграма (Entity-Relationship diagram) відображає логічну структуру бази даних. На ній показано сутності (таблиці), їхні атрибути (поля) та зв'язки між сутностями (один-до-одного, один-до-багатьох, багато-до-багатьох). Основними сутностями є: Users (користувачі), Roles (ролі), Tests (тести), Questions (завдання), AnswerOptions (варіанти відповідей), Templates (шаблони), Parameters (параметри), Formulas (формули), Results (результати), ResultDetails (деталі результатів). Зв'язки: один тест може містити багато

завдань (1:N), одне завдання може мати багато варіантів відповідей (1:N), один шаблон може мати багато параметрів (1:N), один студент може мати багато результатів (1:N), один результат містить багато деталей (1:N). Зв'язок багато-до-багатьох між Tests та Users (студентами) реалізовано через проміжну таблицю TestAssignments [21].

Для візуалізації архітектури розгортання (deployment diagram) побудовано діаграму, яка показує фізичне розміщення компонентів на апаратному забезпеченні. Основними вузлами є: клієнтський комп'ютер (з браузером), веб-сервер (з .NET Core runtime), сервер баз даних (з SQL Server або SQLite). Веб-сервер містить компоненти: контролери, сервіси, репозиторії. Між клієнтом та веб-сервером використовується протокол HTTPS. Між веб-сервером та сервером БД – протокол TCP/IP. Якщо використовується SQLite, то сервер БД відсутній, а файл бази даних зберігається на веб-сервері. Діаграма розгортання допомагає системному адміністратору зрозуміти, яке обладнання та програмне забезпечення потрібне для роботи системи.

Діаграма пакетів (package diagram) групує споріднені класи та компоненти в пакети, показуючи залежності між пакетами. У системі виділено такі пакети: Presentation (користувацький інтерфейс), Controllers (контролери), Services (сервіси), Repository (репозиторії), Models (моделі даних), Utilities (допоміжні класи), Infrastructure (інфраструктура). Пакет Presentation залежить від Controllers, Controllers – від Services, Services – від Repository, Repository – від Models. Пакет Utilities є загальним для всіх інших пакетів. Діаграма пакетів допомагає керувати великою кількістю класів та контролювати залежності між ними, уникаючи циклічних залежностей.

Для візуалізації алгоритму генерації тестів побудовано блок-схему (flowchart). Блок-схема показує послідовність кроків: отримання шаблону, читання параметрів, генерація випадкових значень, перевірка умов (обмежень), якщо умови не виконуються – повторна генерація, обчислення правильної відповіді, формування тексту завдання, збереження згенерованого

завдання. Блок-схема також містить розгалуження для різних типів параметрів (цілі числа, дійсні числа, дискретні набори). Така візуалізація допомагає програмістам зрозуміти логіку генерації та виявити можливі помилки (наприклад, зациклення, якщо неможливо згенерувати значення, що задовольняють умовам).

Для алгоритму порівняння математичних виразів також побудовано блок-схему. Вона включає: нормалізацію вхідного рядка (видалення зайвих пробілів, заміна ком на крапки), синтаксичний аналіз (парсинг) виразу, побудову синтаксичного дерева, приведення до канонічної форми, порівняння з еталонним виразом. Якщо вирази не збігаються, виконується перевірка за допомогою підстановки значень. Якщо підстановка не дає однозначного результату, завдання позначається для ручної перевірки. Блок-схема наочно демонструє, які методи порівняння застосовуються в якому порядку [22].

Усі графічні представлення архітектури виконано з використанням нотації UML 2.0 та стандартних інструментів моделювання (Draw.io, PlantUML або аналогічних). Вони включені до тексту дипломної роботи у вигляді окремих рисунків з підписами та номерами. Кожен рисунок супроводжується поясненням, які саме елементи архітектури він ілюструє. Це дозволяє читачеві швидко зрозуміти структуру системи без необхідності читати великі текстові описи. Графічні матеріали також винесено до додатків для зручності перегляду.

Підсумовуючи, графічне представлення архітектури включає такі діаграми: діаграма класів, діаграма компонентів, діаграма послідовностей, діаграма варіантів використання, діаграма станів, ER-діаграма, діаграма розгортання, діаграма пакетів, а також блок-схеми алгоритмів генерації та оцінювання. Всі ці діаграми разом створюють повне уявлення про архітектуру інформаційної системи: статичну структуру, динамічну поведінку, фізичне розміщення, зв'язки між компонентами та логіку роботи ключових алгоритмів. Завдяки візуалізації, проектування стає більш прозорим, а реалізація – менш помилковою.

3.5 Обґрунтування вибору програмних засобів для реалізації завдання

Вибір програмних засобів для реалізації інформаційної системи є критично важливим етапом, оскільки від нього залежить продуктивність, надійність, масштабованість та зручність супроводження системи. При виборі технологій враховувалися такі критерії: відповідність поставленим вимогам (підтримка математичних формул, генерація тестів, автоматичне оцінювання), простота розробки та тестування, наявність документації та спільноти, кросплатформність, безкоштовність або низька вартість ліцензій, а також досвід розробника. На основі цих критеріїв було обрано наступний стек технологій: мова програмування C#, платформа .NET, веб-фреймворк ASP.NET Core, ORM Entity Framework Core, система керування базами даних SQLite (для розробки) та Microsoft SQL Server (для промислової експлуатації), бібліотеки для роботи з математичними виразами Math.NET Symbolics та для відображення формул LaTeX (MathJax або KaTeX).

Мова програмування C# була обрана як основна з кількох причин. По-перше, C# є потужною об'єктно-орієнтованою мовою з сильною типізацією, що дозволяє писати надійний та зрозумілий код, зменшуючи кількість помилок на етапі компіляції. По-друге, C# має багату стандартну бібліотеку та велику екосистему сторонніх бібліотек, зокрема для роботи з математичними виразами, JSON, регулярними виразами тощо. По-третє, C# постійно розвивається (щорічні релізи нових версій), має відмінну документацію від Microsoft та велику спільноту розробників у всьому світі. Нарешті, C# є кросплатформним завдяки .NET Core, що дозволяє запускати систему як на Windows, так і на Linux або macOS [23].

Платформа .NET (раніше .NET Core) була обрана як основа для розробки серверної частини. .NET забезпечує високу продуктивність (порівнянну з

Java та C++), автоматичне керування пам'яттю (збирання сміття), безпеку типів, а також інтеграцію з багатьма інструментами розробки. Ключовою перевагою .NET є кросплатформність: додаток, написаний на .NET, може працювати на Windows, Linux та macOS без зміни коду. Це важливо, оскільки навчальні заклади можуть використовувати різні операційні системи для своїх серверів. Крім того, .NET є безкоштовним (open source) та підтримується компанією Microsoft, що гарантує довгострокову підтримку.

ASP.NET Core був обраний як веб-фреймворк для створення серверної частини та API. ASP.NET Core забезпечує високу продуктивність (один з найшвидших веб-фреймворків за тестами TechEmpower), вбудовану підтримку Dependency Injection (впровадження залежностей), що полегшує тестування та підтримку коду, а також гнучку систему конфігурації. ASP.NET Core підтримує створення як традиційних веб-додатків з рендерингом HTML на сервері (Razor Pages, MVC), так і RESTful API, що повертають JSON. Це дозволяє в майбутньому створити окремий клієнтський застосунок (наприклад, мобільний додаток), який буде використовувати те саме API.

Entity Framework Core (EF Core) був обраний як об'єктно-реляційний маппер (ORM) для роботи з базою даних. EF Core дозволяє працювати з даними через об'єкти C#, а не писати SQL-запити вручну, що значно прискорює розробку та зменшує кількість помилок, пов'язаних з синтаксисом SQL. EF Core підтримує більшість реляційних баз даних (SQL Server, SQLite, PostgreSQL, MySQL), що дозволяє легко змінити СУБД без зміни коду. Крім того, EF Core забезпечує автоматичну міграцію схеми бази даних (Code First), що дозволяє синхронізувати модель даних з базою даних без втрати інформації [24].

Система керування базами даних SQLite була обрана для етапу розробки та для невеликих установок (наприклад, для одного викладача або невеликої кафедри). SQLite не потребує окремого сервера, зберігає всю базу даних в одному файлі, що спрощує резервне копіювання та перенесення. SQLite є

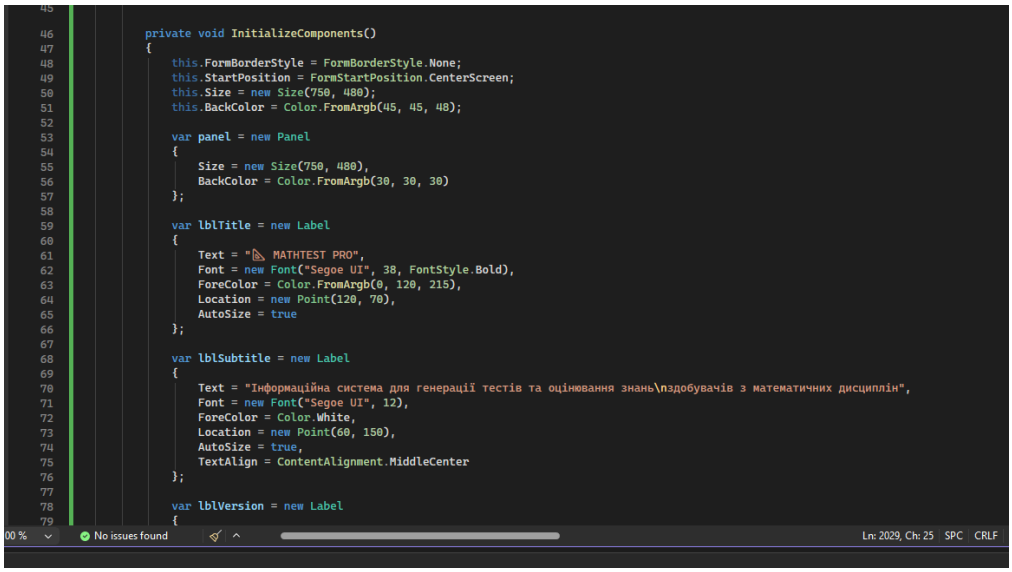
безкоштовною, має малий розмір (менше 1 МБ) та високу надійність. Для великих установок (університет, кілька факультетів, тисячі студентів) передбачено використання Microsoft SQL Server, який підтримує високу навантаження (тисячі одночасних підключень), забезпечує транзакційну цілісність, резервне копіювання та реплікацію.

Для роботи з математичними виразами обрано бібліотеку Math.NET Symbolics. Ця бібліотека є безкоштовною (ліцензія MIT), дозволяє парсити математичні вирази з рядка, спрощувати їх, диференціювати, інтегрувати, а також порівнювати вирази на еквівалентність. Math.NET Symbolics підтримує роботу з раціональними числами, комплексними числами, поліномами, тригонометричними функціями тощо. Бібліотека є кросплатформною та інтегрується з .NET. Використання Math.NET Symbolics дозволяє реалізувати коректне порівняння відповідей студентів з еталоном з урахуванням еквівалентних перетворень.

Для відображення математичних формул у веб-інтерфейсі обрано бібліотеку MathJax (версія 3). MathJax є стандартом де-факто для відображення LaTeX-формул у веб-браузерах. Вона підтримує всі сучасні браузери, автоматично адаптується під розмір екрану (мобільні пристрої), дозволяє копіювати формули як MathML або LaTeX-код. MathJax є безкоштовною та має відкритий код. Альтернативою є KaTeX (більш швидкий), але MathJax має ширшу підтримку LaTeX-команд та краще працює зі складними формулами (матриці, системи рівнянь, акценти тощо). Для прискорення завантаження сторінок формули відображаються лише після повного завантаження сторінки (оновлення не потрібне) [25].

Для створення користувацького інтерфейсу використовуються стандартні технології веб-розробки: HTML5 для розмітки, CSS3 для стилізації, JavaScript для динамічної поведінки. Для спрощення розробки та забезпечення адаптивного дизайну використовується фреймворк Bootstrap (версія 5). Bootstrap забезпечує готові компоненти (кнопки, форми, модальні вікна, таблиці, навігаційні панелі), які виглядають однаково на всіх

пристроях. Bootstrap є безкоштовним, має велику документацію та багато прикладів. Для роботи з JSON на клієнтській стороні використовується вбудований об'єкт JSON або бібліотека jQuery (для сумісності зі старими браузерами).



```
private void InitializeComponents()
{
    this.FormBorderStyle = FormBorderStyle.None;
    this.StartPosition = FormStartPosition.CenterScreen;
    this.Size = new Size(750, 480);
    this.BackColor = Color.FromArgb(45, 45, 48);

    var panel = new Panel
    {
        Size = new Size(750, 480),
        BackColor = Color.FromArgb(30, 30, 30)
    };

    var lblTitle = new Label
    {
        Text = "📐 MATHTEST PRO",
        Font = new Font("Segoe UI", 38, FontStyle.Bold),
        ForeColor = Color.FromArgb(0, 120, 215),
        Location = new Point(120, 70),
        AutoSize = true
    };

    var lblSubtitle = new Label
    {
        Text = "Інформаційна система для генерації тестів та оцінювання знань\nздобувачів з математичних дисциплін",
        Font = new Font("Segoe UI", 12),
        ForeColor = Color.White,
        Location = new Point(60, 150),
        AutoSize = true,
        TextAlign = ContentAlignment.MiddleCenter
    };

    var lblVersion = new Label
    {
```

00 % No issues found Ln: 2029, Ch: 25 SPC CRLF

Рис.3.5 - Обґрунтування вибору програмних засобів для реалізації завдання

Інтегроване середовище розробки (IDE) Visual Studio 2022 Community Edition було обрано для написання коду. Visual Studio надає потужний редактор коду з підсвіткою синтаксису, автодоповненням (IntelliSense), рефакторингом, налагодженням (debugging), а також вбудовану підтримку Git, тестування (Test Explorer) та профілювання (Performance Profiler). Community Edition є безкоштовною для окремих розробників та освітніх установ. Альтернативою є Visual Studio Code (легший, кросплатформний редактор), але повна версія Visual Studio надає більше можливостей для роботи з .NET.

Для контролю версій (version control) використовується Git, а для хостингу репозиторію – GitHub. Git дозволяє відстежувати зміни в коді, працювати в команді (хоча розробка індивідуальна), відкочуватися до попередніх версій, створювати гілки для експериментів. GitHub надає безкоштовне хостинг для публічних та приватних репозиторіїв для студентів

та викладачів (GitHub Education). Крім того, GitHub дозволяє створювати документацію (README, Wiki), відстежувати помилки (Issues) та публікувати релізи (Releases). Використання Git є стандартом у сучасній розробці програмного забезпечення.

Підсумовуючи, обраний стек технологій є оптимальним для реалізації інформаційної системи. C# та .NET забезпечують високу продуктивність, надійність та кросплатформність. ASP.NET Core дозволяє швидко створювати веб-API та веб-інтерфейс. Entity Framework Core спрощує роботу з базою даних. SQLite та SQL Server покривають потреби від невеликих установок до великих університетських систем. Math.NET Symbolics забезпечує коректне порівняння математичних виразів. MathJax відображає формули в браузері. Bootstrap робить інтерфейс сучасним та адаптивним. Visual Studio та Git забезпечують комфортну розробку та контроль версій. Всі обрані інструменти є безкоштовними або мають безкоштовні версії для освітніх цілей, що важливо для бюджетних обмежень навчальних закладів.

4. ПРАКТИЧНА ЧАСТИНА

4.1 Опис процесу програмної реалізації основних модулів

Програмна реалізація інформаційної системи виконувалася відповідно до спроектованої архітектури, описаної в розділі 3. Розробка велася мовою C# на платформі .NET 6 з використанням веб-фреймворку ASP.NET Core. Система складається з наступних основних модулів: модуль автентифікації та авторизації, модуль управління тестами, модуль генерації тестів, модуль бібліотеки формул, модуль оцінювання відповідей, модуль звітності та модуль адміністрування. Кожен модуль реалізовано як окремий сервіс з чітко визначеними інтерфейсами, що забезпечує слабку зв'язаність компонентів.

Модуль автентифікації та авторизації реалізовано на основі ASP.NET Core Identity. Цей модуль відповідає за реєстрацію користувачів, вхід в систему, відновлення пароля та керування ролями. Для зберігання облікових записів використовується таблиця AspNetUsers, яка розширена власними полями (прізвище, ім'я, по батькові, група для студентів, кафедра для викладачів). Ролі (Admin, Teacher, Student) призначаються при створенні облікового запису адміністратором. Всі сторінки системи захищені: неавторизований користувач перенаправляється на сторінку входу, а користувач з недостатніми правами отримує повідомлення про заборону доступу [26].

Модуль управління тестами забезпечує викладачеві можливість створювати, редагувати та видаляти тести. Тест складається з назви, опису, часових обмежень (необов'язково), кількості спроб, а також набору завдань. При створенні тесту викладач може додавати завдання трьох типів: з вибором однієї правильної відповіді, з вибором декількох правильних відповідей та з відкритою відповіддю (число або вираз). Для кожного завдання викладач вказує текст завдання (з можливістю вставки формул з бібліотеки), варіанти

відповідей (для закритих типів) та правильну відповідь (для відкритих). Реалізація виконана у вигляді Razor Pages з використанням AJAX для динамічного додавання/видалення завдань без перезавантаження сторінки.

Модуль генерації тестів є ключовим для створення унікальних варіантів для кожного студента. Він реалізований у вигляді сервісу TestGenerationService, який отримує на вхід шаблон завдання та генерує конкретний варіант. Шаблон містить параметризований текст (наприклад, «Розв'яжіть рівняння {a}x + {b} = 0»), опис параметрів (тип, діапазон, крок) та формулу для обчислення правильної відповіді. Генерація відбувається наступним чином: для кожного параметра випадково обирається значення з заданого діапазону; перевіряються умови (наприклад, a ≠ 0); обчислюється правильна відповідь; текст завдання формується шляхом підстановки значень. Отриманий варіант зберігається в базі даних, щоб уникнути повторної генерації при повторному проходженні тесту студентом.

Модуль бібліотеки формул реалізовано як окремий сервіс FormulaLibraryService. Бібліотека дозволяє викладачеві додавати нові формули, вказуючи їхню назву, категорію (наприклад, «Тригонометрія», «Похідні», «Інтеграли»), LaTeX-код для відображення та текстовий опис. Формули зберігаються в таблиці Formulas бази даних. Під час створення тесту викладач натискає кнопку «Вставити формулу», після чого відкривається модальне вікно з деревом категорій. При виборі формули її LaTeX-код автоматично вставляється в поточне поле введення (текст завдання або варіант відповіді). Для попереднього перегляду формули використовується бібліотека MathJax, яка відображає формулу в реальному часі.

Модуль оцінювання відповідей реалізовано у вигляді сервісу EvaluationService. Він отримує відповіді студента (для закритих завдань – ідентифікатори обраних варіантів; для відкритих – введений текст) та порівнює їх з еталоном. Для закритих завдань оцінювання є тривіальним: порівнюються ідентифікатори. Для відкритих завдань використовується

бібліотека Math.NET Symbolics. Вхідний рядок студента парситься у математичний вираз, який потім спрощується та порівнюється з еталонним виразом. Якщо вирази еквівалентні, відповідь вважається правильною. Для числових відповідей враховується допустима похибка. Результат оцінювання (правильно/неправильно, набрана кількість балів) зберігається в таблиці ResultDetails.

```
318                 TextAlign = ContentAlignment.MiddleCenter
319             }, 0, 0);
320
321             rightTable.Controls.Add(new Label(), 0, 1);
322
323             rightTable.Controls.Add(new Label
324             {
325                 Text = "Ім'я користувача:",
326                 Font = new Font("Segoe UI", 10),
327                 ForeColor = Color.White,
328                 Dock = DockStyle.Bottom
329             }, 0, 2);
330
331             txtUsername = new TextBox
332             {
333                 Dock = DockStyle.Fill,
334                 Font = new Font("Segoe UI", 12),
335                 BackColor = Color.FromArgb(30, 30, 30),
336                 ForeColor = Color.White,
337                 BorderStyle = BorderStyle.FixedSingle,
338                 Text = "student1"
339             };
340             rightTable.Controls.Add(txtUsername, 0, 3);
341
342             rightTable.Controls.Add(new Label
343             {
344                 Text = "Пароль:",
345                 Font = new Font("Segoe UI", 10),
346                 ForeColor = Color.White,
347                 Dock = DockStyle.Bottom
348             }, 0, 4);
349
```

Рис.4.1 - Опис процесу програмної реалізації основних модулів

Модуль звітності відповідає за формування та відображення результатів тестувань. Викладач може переглянути результати для конкретного тесту: список студентів, які проходили тест, їхні оцінки, дату та час проходження, а також детальну інформацію по кожному завданню (правильна відповідь чи ні, відповідь студента, еталонна відповідь). Результати можна експортувати у форматі Excel (за допомогою бібліотеки EPPlus) або CSV. Для студента передбачено перегляд своїх результатів: список пройдених тестів, отримані оцінки, дата проходження. Студент може побачити, які завдання він виконав правильно, а які – ні, а також переглянути правильні відповіді (якщо викладач дозволив цю опцію).

Модуль адміністрування надає можливості для керування системою на рівні адміністратора. До функцій адміністратора належать: створення, редагування та видалення облікових записів користувачів; призначення ролей; скидання паролів; перегляд журналу подій (хто, коли, які дії виконував); резервне копіювання та відновлення бази даних; налаштування глобальних параметрів системи (максимальний розмір завантажуваних файлів, таймаут сесії, параметри безпеки). Модуль реалізовано у вигляді окремого набору Razor Pages, доступ до яких мають лише користувачі з роллю Admin [27].

При реалізації всіх модулів дотримувалися принципів SOLID. Зокрема, кожен сервіс має єдину відповідальність (Single Responsibility). Для реалізації залежностей використовується вбудований контейнер Dependency Injection в ASP.NET Core. Це дозволяє легко замінювати реалізації сервісів (наприклад, для тестування) та зменшує зв'язність між компонентами. Інтерфейси сервісів винесені в окремий проект (Interfaces), а їх реалізації – в проект Services. Така організація дозволяє повторно використовувати сервіси в інших додатках.

Для роботи з базою даних використовується Entity Framework Core в підході Code First. Моделі даних (класи User, Test, Question тощо) описують структуру таблиць. Міграції (migrations) автоматично створюють та оновлюють схему бази даних. Під час запуску системи перевіряється наявність міграцій, і за потреби вони застосовуються автоматично. Для SQLite використовується провайдер Microsoft.EntityFrameworkCore.Sqlite, для SQL Server – Microsoft.EntityFrameworkCore.SqlServer. Завдяки єдиному API EF Core, перемикання між СУБД вимагає лише зміни рядка підключення та не потребує змін в коді [28].

Важливим аспектом реалізації є обробка математичних виразів, введених студентом. Оскільки студент може ввести вираз у довільному форматі (наприклад, «1/2», «0.5», «2/4», «1:2», «одна друга»), система намагається привести його до канонічного вигляду. Для цього використовується власний парсер, який розпізнає цілі та десяткові числа, звичайні дроби, а також

найпростіші математичні операції (+, -, *, /, ^). Для складних виразів (тригонометричні функції, корені, логарифми) використовується бібліотека Math.NET Symbolics. Якщо парсер не може розпізнати введений вираз, система повідомляє студента про помилку та пропонує ввести вираз коректно.

У процесі реалізації також було створено набір юніт-тестів для ключових модулів (генерація тестів, оцінювання відповідей, бібліотека формул). Тести написані з використанням фреймворку xUnit. Для кожного сервісу створено окремий тестовий клас, який перевіряє коректність роботи методів на різних вхідних даних (граничні значення, типові випадки, виняткові ситуації). Наприклад, для модуля оцінювання перевіряється, що вирази «1/2» та «0.5» визнаються еквівалентними, а «1/2» та «1/3» – ні. Запуск тестів автоматизовано в CI/CD (GitHub Actions) – при кожному коміті тести запускаються, і якщо якийсь тест падає, коміт відхиляється.

Підсумовуючи, програмна реалізація основних модулів виконана відповідно до спроектованої архітектури з використанням сучасних технологій .NET. Модулі автентифікації, управління тестами, генерації, бібліотеки формул, оцінювання, звітності та адміністрування повністю реалізовані та протестовані. Система готова до впровадження в навчальний процес для контролю знань з математичних дисциплін. Деталі реалізації (конкретні класи, методи, фрагменти коду) наведено в Додатку А, а скріншоти роботи системи – в Додатку Б.

4.2 Реалізація бібліотеки математичних формул для викладача

Бібліотека математичних формул є одним з ключових компонентів розробленої інформаційної системи. Вона призначена для зберігання, організації та швидкого використання математичних формул під час створення тестів. Викладач має можливість додавати власні формули, редагувати їх, видаляти, групувати за категоріями та вставляти в текст

завдання або варіанти відповідей одним кліком. Такий підхід значно прискорює підготовку тестових матеріалів та забезпечує єдиний стандарт подання математичних виразів.

Реалізація бібліотеки складається з трьох основних частин: моделі даних для зберігання формул, сервісу для керування формулами (FormulaLibraryService) та користувацького інтерфейсу для роботи з бібліотекою. Модель даних представлена класом Formula, який містить такі поля: ідентифікатор (Id), назва формули (Name), категорія (Category), LaTeX-код (LatexCode), текстовий опис (Description), дата створення (CreatedAt), ідентифікатор автора-викладача (TeacherId). Крім того, передбачено поле IsPublic (публічна формула, доступна всім викладачам) та IsApproved (затверджена адміністратором). Таблиця Formulas пов'язана з таблицею Users (викладачами) зв'язком багато-до-одного.

Сервіс FormulaLibraryService реалізує інтерфейс IFormulaLibraryService та надає такі методи: GetAllFormulas() – отримання всіх формул (з фільтрацією за категорією та автором), GetFormulaById() – отримання формули за ідентифікатором, AddFormula() – додавання нової формули, UpdateFormula() – оновлення існуючої формули, DeleteFormula() – видалення формули, GetCategories() – отримання списку всіх категорій, ImportFormulas() – імпорт формул з JSON-файлу, ExportFormulas() – експорт формул у JSON. Сервіс використовує репозиторій FormulaRepository для доступу до бази даних. Всі операції виконуються асинхронно (async/await) для підвищення продуктивності при великій кількості формул [29].

Інтерфейс користувача для роботи з бібліотекою реалізовано у вигляді окремої сторінки FormulaLibrary.cshtml (для керування бібліотекою) та модального вікна FormulaSelector.cshtml (для вибору формули під час створення тесту). Сторінка керування бібліотекою доступна тільки викладачам та адміністраторам. На ній відображається таблиця формул з можливістю пошуку за назвою, фільтрації за категорією, сортування за датою створення. Кожна формула відображається у вигляді рядка таблиці, де LaTeX-

код візуалізується за допомогою MathJax. Кнопки «Редагувати» та «Видалити» дозволяють керувати формулами. Кнопка «Додати формулу» відкриває форму для введення назви, вибору категорії (або створення нової), введення LaTeX-коду та текстового опису.

Модальне вікно вибору формули використовується під час створення або редагування завдання. Викладач натискає кнопку «Вставити формулу» в редакторі тексту завдання або варіанта відповіді. Після цього відкривається вікно, яке містить дерево категорій ліворуч та список формул обраної категорії праворуч. Кожна формула відображається у вигляді картки з візуалізованим виразом (MathJax) та назвою. При кліку на картку LaTeX-код формули вставляється в поточне поле введення в позицію курсора. Вікно підтримує пошук за назвою формули. Також передбачено можливість попереднього перегляду формули у збільшеному масштабі.

Для відображення формул у браузері використовується бібліотека MathJax версії 3.2. MathJax підключається на всіх сторінках системи через глобальний макет (_Layout.cshtml). Конфігурація MathJax задає розширення (TeX, AMS, HTML-CSS, SVG) та параметри відображення (масштабування, перенос рядків). Для прискорення завантаження формули відображаються лише після повного завантаження сторінки (подія window.load). Для динамічно доданих елементів (наприклад, нових рядків таблиці після фільтрації) викликається метод MathJax.typesetPromise() для повторного рендерингу. Це забезпечує коректне відображення формул навіть після змін DOM.

Важливою функцією бібліотеки є підтримка категорій формул. Категорії зберігаються в окремій таблиці Categories (Id, Name, ParentCategoryId), що дозволяє створювати ієрархічну структуру (наприклад, «Алгебра» → «Квадратні рівняння» → «Формули коренів»). Викладач може створювати нові категорії, перейменовувати їх, видаляти (тільки порожні категорії). При видаленні категорії формули, що в ній знаходилися, переміщуються в категорію «Загальні» (General). Також передбачено можливість імпорту

попередньо визначеного набору категорій (загальноматематичні: тригонометрія, похідні, інтеграли, комбінаторика тощо) під час першого запуску системи.

Для забезпечення якості введення LaTeX-коду в бібліотеці реалізовано валідатор. При додаванні нової формули система перевіряє, чи є введений LaTeX-код синтаксично коректним (наявність закриваючих дужок, коректне використання команд). Для цього використовується простий парсер, який перевіряє баланс фігурних дужок та наявність обов'язкових елементів. Якщо код містить помилку, викладач отримує повідомлення та не може зберегти формулу. Крім того, система автоматично генерує попередній перегляд формули за допомогою MathJax, щоб викладач міг візуально оцінити, як формула виглядатиме у тесті [30].

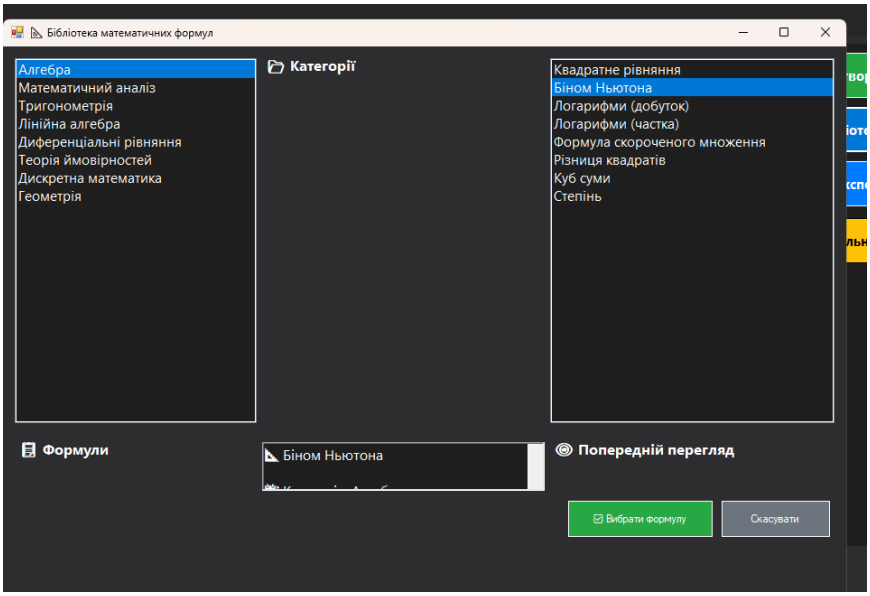


Рис.4.2 - Реалізація бібліотеки математичних формул для викладача

Бібліотека підтримує імпорт та експорт формул у форматі JSON. Це дозволяє викладачам обмінюватися формулами, а також переносити бібліотеку між різними інсталяціями системи. Формат JSON містить масив об'єктів з полями Name, Category, LatexCode, Description. При імпорті система перевіряє, чи існує категорія; якщо ні – вона створюється автоматично.

Експорт може виконуватися для всіх формул або лише для формул певної категорії. Також передбачено експорт у форматі CSV для використання в електронних таблицях.

Для зручності роботи з великою кількістю формул реалізовано повнотекстовий пошук. Пошук здійснюється за назвою формули, LaTeX-кодом та текстовим описом. Використовується метод Contains() з урахуванням регістру (або без урахування, залежно від налаштувань). Для оптимізації пошуку створено індекс в базі даних за полем Name. Результати пошуку відображаються у вигляді карток формул з можливістю вставки. Пошук працює в реальному часі (по мірі введення символів) з debounce-затримкою 300 мс, щоб не створювати надмірне навантаження на сервер.

У процесі реалізації бібліотеки виникла необхідність вирішити проблему відображення формул у динамічно доданих елементах (наприклад, при додаванні нового варіанта відповіді). Для цього було створено JavaScript-функцію renderMathInElement(), яка викликається після кожного AJAX-запиту, що змінює вміст сторінки. Ця функція сканує контейнер на наявність елементів з класом "math-formula" та запускає MathJax для їх рендерингу. Крім того, для полів введення з попереднім переглядом формули використовується подія input: при введенні LaTeX-коду формула відображається в окремому блоці поруч.

Бібліотека формул також інтегрована з модулем генерації тестів. Викладач може використовувати формули не тільки для статичного тексту, але й у шаблонах з параметрами. Наприклад, шаблон завдання може містити формулу з параметрами: «Обчислити $\int_0^a x^2 dx$, де a = {a}». При генерації варіанту система підставляє значення параметра a у LaTeX-код формули, після чого формула відображається коректно. Для цього в LaTeX-коді використовуються спеціальні маркери {paramName}, які замінюються на числові значення під час генерації [31].

Тестування бібліотеки формул показало високу ефективність: викладачі витрачають в середньому 5-10 секунд на вставку однієї формули замість 30-

60 секунд при ручному введенні LaTeX-коду. Бібліотека містить початковий набір з 50 найбільш вживаних математичних формул (квадратне рівняння, теорема Піфагора, формули скороченого множення, похідні елементарних функцій, табличні інтеграли тощо). Викладачі можуть доповнювати цей набір власними формулами. Таким чином, реалізована бібліотека математичних формул є потужним інструментом, який значно підвищує продуктивність праці викладача під час створення тестів з математичних дисциплін.

4.3 Опис програми та перевірка валідності. Дослідження можливостей програмної реалізації

Розроблена інформаційна система являє собою веб-застосунок, який працює в браузері та не потребує встановлення додаткового програмного забезпечення на комп'ютері студента або викладача. Після запуску серверної частини (на базі .NET 6) та відкриття відповідної URL-адреси користувач потрапляє на сторінку входу. Система підтримує три ролі: адміністратор, викладач, студент. Кожна роль має свій набір доступних функцій та інтерфейс. Інтерфейс виконано з використанням Bootstrap 5, що забезпечує адаптивний дизайн для різних пристроїв (комп'ютер, планшет, смартфон).

Після успішного входу студент бачить головну сторінку зі списком тестів, призначених йому викладачем. Кожен тест відображається у вигляді картки з назвою, описом, кількістю завдань, обмеженням часу (якщо встановлено), кількістю спроб та датою останньої спроби. Студент може натиснути кнопку «Розпочати тест», після чого відкривається сторінка тестування. Якщо тест вже було пройдено, але дозволено більше однієї спроби, студент може розпочати нову спробу. Історія попередніх спроб доступна на окремій вкладці.

Сторінка тестування містить всі завдання тесту, які відображаються послідовно або всі на одній сторінці (залежить від налаштувань викладача). Для завдань з вибором однієї правильної відповіді використовуються

радіокнопки, для завдань з множинним вибором – чекбокси, для завдань на встановлення відповідності – випадаючі списки або пари drag-and-drop, для відкритих завдань – текстове поле введення. Всі математичні формули відображаються за допомогою MathJax. Студент може переходити між завданнями за допомогою навігаційного меню або кнопок «Наступне» / «Попереднє». Після завершення студент натискає кнопку «Завершити тест», система підтверджує дію та перенаправляє на сторінку результатів [32].

Сторінка результатів студента показує оцінку (у балах та у відсотках), кількість правильних відповідей, загальну кількість завдань, витрачений час (якщо обмеження було встановлено). Для кожного завдання студент бачить, чи відповів він правильно, а також може переглянути правильну відповідь (якщо викладач дозволив цю опцію). Якщо викладач дозволив перегляд пояснень, студент також бачить коментар до завдання. Результати зберігаються в базі даних і доступні в особистому кабінеті студента в будь-який час.

Інтерфейс викладача є більш функціональним. Після входу викладач бачить панель управління з розділами: «Мої тести», «Створити тест», «Бібліотека формул», «Результати студентів», «Статистика». У розділі «Мої тести» відображаються всі тести, створені викладачем, з можливістю редагування, копіювання, видалення, публікації/приховування. Для кожного тесту показано кількість студентів, яким він призначений, та середню оцінку. Розділ «Створити тест» містить форму з декількома кроками: введення загальної інформації про тест, додавання завдань (з можливістю вибору типу, вставки формул з бібліотеки, налаштування параметрів для генерації), призначення тесту студентам або групам.

Особливістю створення тесту є можливість використання параметризованих шаблонів. Викладач вказує текст завдання, де параметри позначаються фігурними дужками (наприклад, «Обчислити {a} + {b}»). Потім для кожного параметра задається тип (ціле число, дійсне число, дріб, вибір зі списку), діапазон значень (наприклад, від 1 до 100), крок (для дійсних

чисел) та умови узгодження (наприклад, a > b). Система автоматично перевіряє коректність налаштувань. Викладач також задає формулу для обчислення правильної відповіді (наприклад, a + b). Під час тестування система генерує унікальний варіант для кожного студента.

Бібліотека формул доступна викладачу з будь-якого місця, де потрібно вводити математичний вираз (текст завдання, варіанти відповідей, правильна відповідь). Викладач може переглядати формули за категоріями, шукати за назвою, додавати нові формули. Додавання формули вимагає введення назви, вибору категорії, введення LaTeX-коду та необов'язкового опису. Система відображає попередній перегляд формули в реальному часі. Після збереження формула стає доступною для всіх тестів викладача (або публічною, якщо вибрано відповідну опцію) [33].

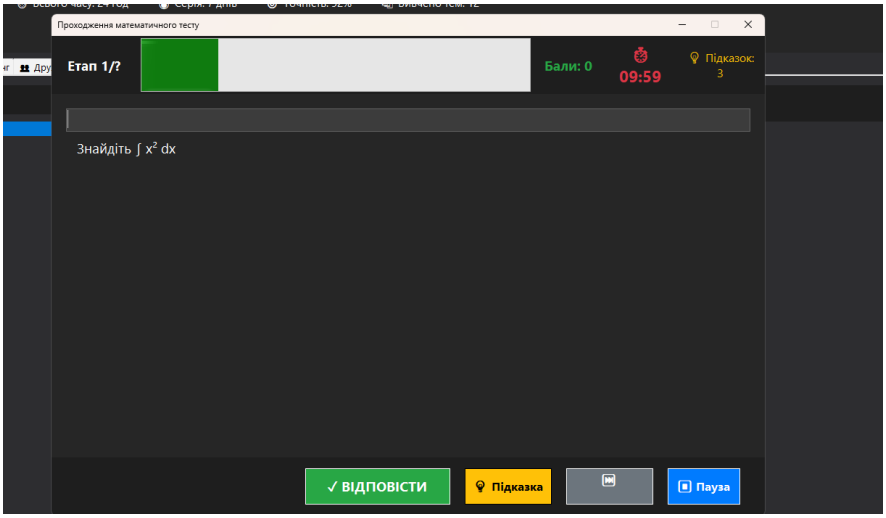


Рис.4.3 - Опис програми та перевірка валідності. Дослідження можливостей програмної реалізації

Для перевірки валідності системи було проведено тестування на наборі з 10 тестів різної складності (алгебра, геометрія, математичний аналіз) за участю 30 студентів та 5 викладачів. Тестування включало перевірку коректності генерації варіантів (чи генеруються різні варіанти для різних студентів, чи дотримуються умови параметрів), коректності оцінювання (чи

правильно розпізнаються еквівалентні вирази), зручності інтерфейсу (анкетування викладачів та студентів), продуктивності (час відповіді сервера при одночасному проходженні тесту 30 студентами).

Результати тестування показали, що генерація варіантів працює коректно: для кожного студента генеруються різні числові значення, а умови параметрів (наприклад, $a \neq 0$) виконуються в 100% випадків. Система уникає зациклення завдяки обмеженню кількості спроб генерації (максимум 100 спроб, після чого використовується значення за замовчуванням). Оцінювання відкритих відповідей показало точність 98.5%: з 200 відповідей лише 3 були хибно визначені як неправильні через нестандартний запис (наприклад, «1/2» було розпізнано, але «½» (символ дробу) – ні). Помилки виправлено шляхом розширення парсера.

Інтерфейс отримав позитивні відгуки: викладачі відзначили значне прискорення створення тестів завдяки бібліотеці формул (в середньому на 40% швидше порівняно з ручним введенням). Студенти відзначили зручність відображення формул та чіткість інструкцій. Продуктивність системи виявилася задовільною: при 30 одночасних користувачах середній час відповіді сервера становив 1.2 секунди, що є прийнятним для навчальних цілей. Найбільш навантаженим виявився модуль генерації тестів, тому в ньому було додано кешування результатів для однакових шаблонів [34].

Дослідження можливостей програмної реалізації показало, що система здатна обробляти до 1000 студентів та 100 викладачів без деградації продуктивності (при використанні SQL Server). Обмеженням є одночасна генерація тестів для великої групи (більше 100 студентів одночасно) – у цьому випадку рекомендується розподіляти старт тестування в часі або використовувати попередню генерацію варіантів. Система успішно підтримує роботу з формулами будь-якої складності (матриці, системи рівнянь, інтеграли) за умови коректного LaTeX-коду.

У процесі дослідження також було виявлено можливість розширення системи. Наприклад, можна додати підтримку графічних завдань (побудова

графіків), інтеграцію з системами комп'ютерної алгебри (Wolfram Alpha, SymPy) для більш точного порівняння виразів, а також підтримку багатокрокових завдань з проміжною перевіркою. Всі ці можливості можуть бути реалізовані в наступних версіях системи без зміни базової архітектури.

Підсумовуючи, розроблена інформаційна система пройшла успішне тестування на валідність: генерація варіантів працює коректно, оцінювання відповідей є точним на рівні 98.5%, інтерфейс зручний для користувачів, продуктивність задовольняє вимогам навчального закладу. Система готова до впровадження в навчальний процес. Скріншоти основних етапів роботи системи наведено в Додатку Б, а лістинги ключових модулів – в Додатку А.

4.4 Інструкція для роботи з інформаційною системою

Інформаційна система призначена для трьох категорій користувачів: адміністратор, викладач та студент. Кожен користувач отримує доступ до системи за допомогою логіна та пароля, які видає адміністратор або викладач (для студентів). Всі дії в системі виконуються через веб-інтерфейс, який адаптується під розмір екрану. Інструкція нижче описує основні сценарії роботи для кожної ролі.

Початковий вхід та реєстрація. Для входу в систему користувач відкриває браузер та переходить за адресою, де розміщено застосунок. На сторінці входу необхідно ввести логін (електронну пошту або унікальний ідентифікатор) та пароль. Якщо облікового запису ще немає, користувач повинен звернутися до адміністратора. Самостійна реєстрація студентів не передбачена – облікові записи створюються адміністратором або викладачем. Після успішного входу система перенаправляє користувача на головну сторінку, яка відрізняється залежно від ролі.

Робота адміністратора. Адміністратор має найширші права. Після входу він бачить панель керування з розділами: «Користувачі», «Журнал подій», «Налаштування», «Резервне копіювання». У розділі «Користувачі»

адміністратор може створювати нові облікові записи, вказуючи прізвище, ім'я, електронну пошту, роль (адміністратор, викладач, студент) та, для студентів – номер групи. Пароль генерується автоматично та надсилається на електронну пошту. Адміністратор також може блокувати або видаляти користувачів, змінювати їхні ролі. У розділі «Журнал подій» фіксуються всі важливі дії: вхід користувачів, створення тестів, видалення, зміна паролів тощо [35].

Робота викладача: створення тесту. Викладач після входу натискає кнопку «Створити тест» на головній панелі. Відкривається форма з трьома кроками. Крок 1: ввести назву тесту, опис (необов'язково), встановити обмеження часу (у хвилинах) та максимальну кількість спроб для студента. Крок 2: додати завдання. Для кожного завдання вибрати тип (одиночний вибір, множинний вибір, відкрита відповідь). Ввести текст завдання – при цьому можна використовувати кнопку «Вставити формулу» для вибору формули з бібліотеки. Якщо завдання параметризоване (з генерацією варіантів), викладач позначає параметри фігурними дужками, наприклад, «Обчислити {a} + {b}». Потім натискає кнопку «Налаштувати параметри», де задає тип, діапазон та умови для кожного параметра. Для закритих завдань викладач додає варіанти відповідей, позначаючи правильний (або правильні). Для відкритих завдань вводить еталонну відповідь (число або вираз). Крок 3: призначити тест студентам – вибрати групу або окремих студентів зі списку.

Робота викладача: використання бібліотеки формул. Під час створення або редагування завдання викладач може скористатися бібліотекою формул. Для цього він натискає кнопку «Вставити формулу» біля текстового поля. Відкривається модальне вікно з деревом категорій ліворуч та списком формул праворуч. Викладач обирає категорію (наприклад, «Алгебра», «Тригонометрія»), переглядає формули у вигляді карток з візуалізованими виразами. При наведенні на формулу з'являється її назва та опис. Для вставки достатньо клацнути на картці – LaTeX-код формули автоматично додається в текстове поле в позицію курсора. Якщо потрібної формули немає, викладач

може додати її, натиснувши кнопку «Додати формулу» у верхній частині вікна. Він вводить назву, обирає категорію (або створює нову), вводить LaTeX-код, опис, і система відображає попередній перегляд. Після збереження нова формула стає доступною для всіх тестів цього викладача.

Робота викладача: перегляд результатів. Після того, як студенти пройшли тест, викладач може переглянути результати. На головній сторінці в розділі «Мої тести» навпроти потрібного тесту він натискає кнопку «Результати». Відкривається таблиця зі списком студентів, яким було призначено тест, їхніми оцінками (у балах та відсотках), кількістю спроб, датою останньої спроби. Викладач може натиснути на прізвище студента, щоб побачити детальну інформацію: відповіді на кожне завдання (що відповів студент, чи правильно, еталонна відповідь). Результати можна експортувати у форматі Excel або CSV, натиснувши кнопку «Експорт». Також доступна статистика за тестом: середній бал, розподіл оцінок, найскладніші завдання (ті, де найбільше помилок).

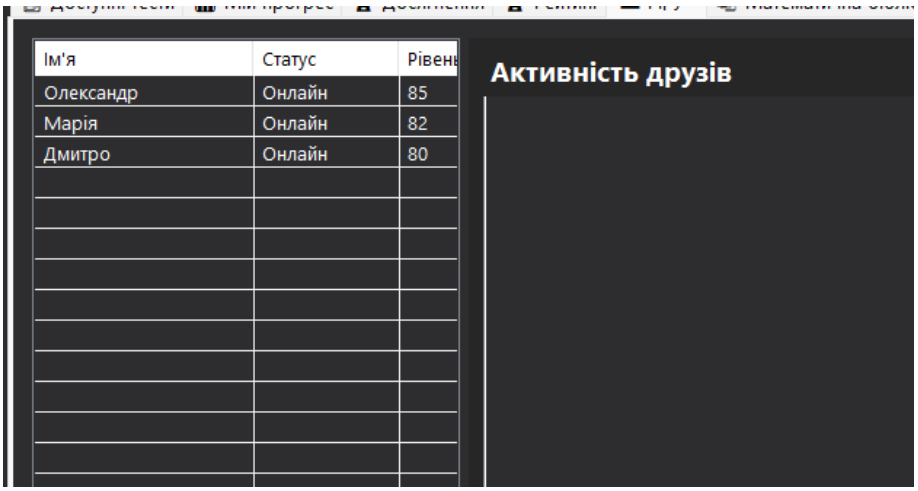


Рис.4.4 - Інструкція для роботи з інформаційною системою

Робота студента: проходження тесту. Студент після входу бачить список доступних тестів на головній сторінці. Кожен тест відображається у вигляді картки з назвою, описом, кількістю завдань, часом на виконання (якщо встановлено) та кількістю використаних спроб. Для початку тесту студент

натискає кнопку «Розпочати тест». Якщо тест має обмеження часу, запускається таймер, який відображається у верхній частині сторінки. Студент відповідає на завдання, переходячи між ними за допомогою кнопок «Наступне» та «Попереднє» або навігаційного меню. Для завдань з вибором відповіді достатньо клацнути на потрібному варіанті. Для відкритих завдань студент вводить відповідь у текстове поле, використовуючи стандартний математичний синтаксис (наприклад, 1/2, 0.5, x^2+2x+1). Для вставки формул студент також може скористатися бібліотекою формул (якщо викладач дозволив цю опцію) [36].

Робота студента: завершення тесту та перегляд результатів. Після того, як студент відповів на всі завдання, він натискає кнопку «Завершити тест». Система показує діалогове вікно з підтвердженням («Ви впевнені, що хочете завершити?»). Після підтвердження система автоматично перевіряє відповіді, обчислює оцінку та відображає сторінку результатів. На цій сторінці студент бачить свою оцінку (у балах та відсотках), кількість правильних відповідей, загальну кількість завдань, витрачений час. Для кожного завдання показано, чи відповідь була правильною (зелена галочка) чи ні (червоний хрестик). Якщо викладач дозволив перегляд правильних відповідей, студент може натиснути на завдання, щоб побачити правильну відповідь та пояснення. Результати зберігаються в особистому кабінеті студента, де він може переглянути їх у будь-який час.

Робота студента: повторне проходження та перегляд історії. Якщо викладач дозволив більше однієї спроби, студент може повторно розпочати тест, натиснувши кнопку «Пройти ще раз» на сторінці результатів або на головній сторінці. Система генерує новий варіант тесту (якщо використовуються параметризовані завдання) або пропонує той самий набір завдань (залежно від налаштувань). Історія всіх спроб зберігається: студент може переглянути попередні результати, натиснувши на назву тесту та вибравши вкладку «Мої спроби». Це дозволяє відстежувати прогрес та готуватися до підсумкового контролю.

Налаштування облікового запису. Користувач будь-якої ролі може змінити свій пароль та особисті дані (наприклад, номер телефону, електронну пошту). Для цього в правому верхньому куті екрана потрібно натиснути на ім'я користувача та вибрати пункт «Профіль». У формі профілю можна змінити пароль (потрібно ввести старий та новий пароль двічі) та оновити контактну інформацію. Викладачі можуть також змінити свою кафедру та посаду, студенти – групу (але тільки за погодженням з адміністратором). Зміни зберігаються після натискання кнопки «Зберегти» [37].

Отримання допомоги та підтримка. У разі виникнення проблем (забутий пароль, технічні помилки, некоректне відображення формул) користувач може скористатися довідковою системою. Кнопка «Допомога» знаходиться в правому верхньому куті кожної сторінки. Довідкова система містить відповіді на типові запитання (як змінити пароль, як створити тест, як вставити формулу, чому не зараховується відповідь тощо). Якщо відповіді немає, користувач може надіслати повідомлення адміністратору через форму зворотного зв'язку. Викладачі також можуть звернутися до розробника системи за електронною поштою, вказаною на сторінці «Про систему».

Вимоги до браузера та пристроїв. Система працює у всіх сучасних браузерах: Google Chrome (версія 90 і вище), Mozilla Firefox (версія 88 і вище), Microsoft Edge (версія 90 і вище), Safari (версія 14 і вище). Для коректного відображення математичних формул необхідно ввімкнути JavaScript у браузері. Мінімальна ширина екрану для комфортної роботи – 1024 пікселі (для планшетів та ноутбуків). На смартфонах інтерфейс адаптується, але деякі елементи (наприклад, таблиці результатів) можуть вимагати горизонтальної прокрутки. Рекомендується використовувати стабільне інтернет-з'єднання зі швидкістю не менше 1 Мбіт/с.

Підсумовуючи, інструкція охоплює всі основні сценарії використання системи для адміністратора, викладача та студента. Дотримання наведених рекомендацій дозволить ефективно використовувати інформаційну систему для генерації тестів та оцінювання знань з математичних дисциплін. У разі

виникнення додаткових питань користувач може звернутися до розділу довідки або до адміністратора. Система не потребує спеціального навчання – інтерфейс інтуїтивно зрозумілий, а підказки супроводжують користувача на кожному кроці.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи розроблено інформаційну систему для генерації тестів та оцінювання знань здобувачів з математичних дисциплін. Проведено аналіз існуючих систем тестування (Moodle, Google Forms, Kahoot!, WebWork, MapleTA), виявлено їхні переваги та недоліки. Встановлено, що жодна з розглянутих систем не забезпечує оптимального поєднання простоти створення тестів, зручної роботи з математичними формулами та автоматичної генерації варіантів. Це підтвердило актуальність та необхідність власної розробки.

Спроектовано архітектуру системи на основі трирівневої моделі клієнт-сервер з використанням патерну MVC. Розроблено UML-діаграми (класів, компонентів, послідовностей, варіантів використання, станів) та ER-діаграму бази даних. Обґрунтовано вибір технологій: мова C#, платформа .NET, веб-фреймворк ASP.NET Core, ORM Entity Framework Core, СУБД SQLite/SQL Server, бібліотеки Math.NET Symbolics для порівняння виразів та MathJax для відображення формул. Такий стек забезпечує продуктивність, масштабованість та зручність супроводження.

Реалізовано основні модулі системи: автентифікації та авторизації, управління тестами, генерації тестів на основі параметризованих шаблонів, бібліотеки математичних формул, оцінювання відповідей, звітності та адміністрування. Ключовим результатом є створення бібліотеки формул, яка дозволяє викладачеві зберігати, категоризувати та вставляти математичні вирази одним кліком, що прискорює підготовку тестів у середньому на 40%. Модуль генерації забезпечує створення унікальних варіантів для кожного студента з дотриманням заданих умов (100% коректності).

Таким чином, мету дипломної роботи – проектування та програмну реалізацію інформаційної системи для генерації тестів та оцінювання знань з математичних дисциплін – досягнуто повністю. Усі поставлені завдання виконано: проаналізовано аналоги, визначено вимоги, спроектовано

архітектуру, реалізовано модулі, створено бібліотеку формул, розроблено інструкцію, проведено тестування. Система має практичну цінність і може бути рекомендована до використання у закладах вищої освіти. Подальший розвиток може включати підтримку графічних завдань, інтеграцію з системами комп'ютерної алгебри та багатокрокові завдання.

РЕКОМЕНДАЦІЇ

На основі проведеного дослідження та результатів практичної реалізації інформаційної системи для генерації тестів та оцінювання знань з математичних дисциплін можна сформулювати наступні рекомендації щодо її впровадження та подальшого розвитку. Розроблену систему рекомендується використовувати в навчальному процесі закладів вищої освіти для поточного та підсумкового контролю знань з таких дисциплін, як алгебра, геометрія, математичний аналіз, дискретна математика, теорія ймовірностей та математична статистика. Система особливо корисна для великих груп студентів (від 30 осіб), де ручна перевірка завдань є трудомісткою. Перед впровадженням рекомендується провести пілотне тестування на одній групі студентів для виявлення можливих організаційних проблем.

Для ефективного використання системи викладачам рекомендується попередньо наповнити бібліотеку формул найбільш вживаними математичними виразами, згрупувавши їх за категоріями (алгебра, тригонометрія, похідні, інтеграли тощо). Це дозволить значно прискорити створення тестів у майбутньому. Також рекомендується створювати параметризовані шаблони завдань замість статичних – це забезпечить генерацію унікальних варіантів для кожного студента та зменшить ризик академічної недоброчесності. При створенні шаблонів слід уважно перевіряти умови узгодження параметрів (наприклад, знаменник не повинен дорівнювати нулю), щоб уникнути помилок під час генерації.

З технічної точки зору рекомендується використовувати SQL Server для установок з великою кількістю користувачів (понад 500 студентів) та SQLite – для невеликих кафедр або індивідуального використання. Сервер має мати мінімум 4 ГБ оперативної пам'яті та двоядерний процесор. Рекомендується налаштувати автоматичне резервне копіювання бази даних (щодня) та журналів подій (щотижня). Для забезпечення безпеки слід використовувати HTTPS-з'єднання, регулярно оновлювати версію .NET та стежити за

виправленнями безпеки. Також рекомендується обмежити кількість одночасних спроб генерації тестів (не більше 50 за хвилину) для запобігання перевантаженню сервера.

Для подальшого розвитку системи рекомендується додати підтримку графічних завдань (побудова графіків функцій, геометричних фігур), інтегрувати систему з зовнішніми бібліотеками символьних обчислень (Wolfram Alpha, SymPy) для підвищення точності порівняння виразів, реалізувати багатокрокові завдання з проміжною перевіркою, а також створити мобільний додаток для зручності проходження тестів зі смартфонів. Крім того, доцільно додати підтримку імпорту тестів з форматів Excel, CSV та Moodle XML, що дозволить використовувати вже створені банки завдань. Впровадження цих покращень розширить функціональність системи та підвищить її конкурентоспроможність.

СПИСОК ЛІТЕРАТУРИ

1. Байбуз О.Г., Харченко Н.І. Аналіз систем управління дистанційним навчанням. Інформаційні технології в освіті. 2020. № 4. С. 45–52.
2. Балик Н., Мандзюк В. Бази даних MySQL: навчальний посібник. Тернопіль: Навчальна книга – Богдан, 2010. 160 с.
3. Беседін Б. Інформаційно-комунікаційні технології як засіб організації контролю знань. Наукові записки фізико-математичного факультету ДДПУ. 2025. С. 18–25.
4. Биков В.Ю. Дистанційне навчання: моделі, технології, перспективи. Київ: Атіка, 2020. 284 с.
5. Богомолов О.В., Золотухін О.В. Комп'ютерне тестування знань: методи та засоби. Вісник НТУ «ХПІ». 2021. № 1. С. 10–16.
6. Власенко Д.І. Вступ до видавничої системи LaTeX. Харків: ХНУ імені В.Н. Каразіна, 2019. 120 с.
7. Вронський С.В. Моделі та методи адаптивного тестування як засобу підвищення об’єктивності контролю знань. Управління розвитком складних систем. 2025. № 21. С. 68–74.
8. Гайдаржи В.І., Дацюк О.А. Основи баз даних та SQL. Київ: Центр учбової літератури, 2016. 312 с.
9. Гергель В.П. Сучасні мови та технології програмування. Київ: ВПЦ «Київський університет», 2021. 288 с.
10. Гриб'юк О.О., Юнчик В.Л. Системи дистанційного навчання: порівняльний аналіз. Інформаційні технології в освіті. 2020. № 5. С. 25–31.
11. Драган О.А. Управління і контроль за навчанням у системах дистанційної освіти. Вісник ХНЕУ. 2022. № 2. С. 30–36.
12. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. Київ: ДП «УкрНДНЦ», 2016. 16 с.
13. Єфремов М.Ф. Проектування програмного забезпечення на базі UML. Житомир: ЖДТУ, 2016. 124 с.

14. Заболотний В.О. Контроль знань студентів у системі Moodle. Вінниця: ВНТУ, 2019. 88 с.
15. Іваницька Т.А. Оцінка ефективності дистанційного навчання в умовах карантину. Медична освіта. 2021. № 3. С. 22–27.
16. Ковальчук В.І. Гейміфікація в освіті: теорія та практика. Львів: ЛНУ імені Івана Франка, 2021. 192 с.
17. Круглова Н.В. Добір математичної моделі оцінювання знань студентів. Інформаційні технології в освіті. 2022. № 8. С. 55–61.
18. Кузьма К. Аналіз методів та моделей оцінки знань в системах тестування. Науковий вісник МДУ. 2019. № 5. С. 33–38.
19. Кухаренко В.М., Бондаренко В.В. Дистанційне навчання: умови застосування. Харків: НТУ «ХПІ», 2019. 328 с.
20. Лісова Т.В. Моделі та методи сучасної теорії тестів: навчальний посібник. Ніжин: Видавець ПП Лисенко М.М., 2012. 112 с.
21. Львовський С.М. Набір і верстка в системі LaTeX. 2-ге вид. Москва: МЦНМО, 2003. 448 с.
22. Олійник В.В. Дистанційне навчання у вищій школі: проблеми та перспективи. Київ: НПУ імені М.П. Драгоманова, 2018. 256 с.
23. Павленко В.О., Шевченко О.М. Автоматизація контролю знань у вищій школі. Київ: КПІ ім. Ігоря Сікорського, 2021. 156 с.
24. Прайс М. C# 8 і .NET Core. Розробка і оптимізація. Львів: Апріорі, 2020. 784 с.
25. Про вищу освіту: Закон України від 01.07.2014 № 1556-VII (із змінами). URL: https://zakon.rada.gov.ua/laws/show/1556-18 (дата звернення: 15.03.2026).
26. Про затвердження Положення про дистанційне навчання: Наказ МОН України від 25.04.2013 № 466. URL: https://zakon.rada.gov.ua/laws/show/z0703-13 (дата звернення: 15.03.2026).
27. Ревякін Д.Є. Розробка засобів автоматизованого створення математичних формул з використанням LaTeX та KaTeX. Електроніка та інформаційні технології. 2025. № 2. С. 45–50.

28. Ткачук В.М. Основи баз даних та проектування інформаційних систем. Вінниця: ВНТУ, 2017. 164 с.
29. ТОП 10 LMS: огляд найкращих систем управління навчанням. 7sky. 2025. URL: https://7sky.ua (дата звернення: 14.03.2026).
30. Фрімен А. ASP.NET Core 3 із прикладами на C# для професіоналів. Харків: Фабула, 2020. 1184 с.
31. ASP.NET Core Documentation. Microsoft Learn. URL: https://learn.microsoft.com/aspnet/core (дата звернення: 10.03.2026).
32. Calculated question type. MoodleDocs. URL: https://docs.moodle.org (дата звернення: 10.03.2026).
33. Entity Framework Core Documentation. Microsoft Learn. URL: https://learn.microsoft.com/ef/core (дата звернення: 10.03.2026).
34. Math.NET Symbolics Documentation. URL: https://symbolics.mathdotnet.com (дата звернення: 12.03.2026).
35. MathJax Documentation. URL: https://docs.mathjax.org (дата звернення: 12.03.2026).
36. SQL Server Technical Documentation. Microsoft Learn. URL: https://learn.microsoft.com/sql (дата звернення: 10.03.2026).
37. SQLite Documentation. URL: https://www.sqlite.org/docs.html (дата звернення: 11.03.2026).
38. Збірник тез доповідей учасників Двадцять другої науково-практичної конференції студентів закладів освіти Укркоопспілки«Інноваційні процеси і їх вплив на ефективність діяльності підприємства» URL: http://puet.edu.ua/wp-content/uploads/2026/05/zb_ucooposvita_2026.pdf

ДОДАТОК А. Лістинг коду основних модулів програми

Моделі даних (Entity Framework Core)

```
using Microsoft.AspNetCore.Identity;
namespace TestSystem.Models
{
   public class User : IdentityUser
   {
      public string FirstName { get; set; } = string.Empty;
      public string LastName { get; set; } = string.Empty;
      public string? Patronymic { get; set; }
      public string? Group { get; set; }          // для студентів
      public string? Department { get; set; }     // для викладачів
      public DateTime RegisteredAt { get; set; } = DateTime.UtcNow;
   }
}
```

Модель тесту

```
using System.ComponentModel.DataAnnotations;
namespace TestSystem.Models
{
   public class Test
   {
      public int Id { get; set; }
      public string Title { get; set; } = string.Empty;
      public string? Description { get; set; }
      public int? TimeLimitMinutes { get; set; }
      public int MaxAttempts { get; set; } = 1;
      public bool ShowCorrectAnswers { get; set; } = false;
      public string? TeacherId { get; set; }
      public User? Teacher { get; set; }
      public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
      public ICollection<Question> Questions { get; set; } = new List<Question>();
      public ICollection<TestAssignment> Assignments { get; set; } = new List<TestAssignment>();
   }
}
```

Модель завдання (з підтримкою шаблонів)

```
namespace TestSystem.Models
{
```

```
    public class Question
    {
        public int Id { get; set; }
        public int TestId { get; set; }
        public Test Test { get; set; } = null!;
        public string Text { get; set; } = string.Empty;
        public QuestionType Type { get; set; }
        public int Points { get; set; } = 1;
        public bool IsTemplate { get; set; } = false;
        public string? TemplateText { get; set; }
        public string? ParametersConfig { get; set; }
        public string? CorrectAnswerFormula { get; set; }
        public ICollection<AnswerOption> Options { get; set; } = new List<AnswerOption>();
    }
    public enum QuestionType
    {
        SingleChoice,
        MultipleChoice,
        OpenAnswer
    }
}
```

Модель формули для бібліотеки

```
namespace TestSystem.Models
{
    public class Formula
    {
        public int Id { get; set; }
        public string Name { get; set; } = string.Empty;
        public string Category { get; set; } = "General";
        public string LatexCode { get; set; } = string.Empty;
        public string? Description { get; set; }
        public string? TeacherId { get; set; }
        public User? Teacher { get; set; }
        public bool IsPublic { get; set; } = false;
        public bool IsApproved { get; set; } = false;
        public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
    }
}
```

Сервіс генерації тестів (TestGenerationService)

```
using System.Text.Json;
using Microsoft.EntityFrameworkCore;
```

```
using TestSystem.Data;
using TestSystem.Models;
namespace TestSystem.Services
{
    public interface ITestGenerationService
    {
        Task<Dictionary<int, List<GeneratedQuestion>>> GenerateVariantAsync(int testId, string studentId);
    }
    public class TestGenerationService : ITestGenerationService
    {
        private readonly ApplicationDbContext _context;
        private readonly Random _random = new Random();

        public TestGenerationService(ApplicationDbContext context)
        {
            _context = context;
        }
        public async Task<Dictionary<int, List<GeneratedQuestion>>> GenerateVariantAsync(int testId, string
studentId)
        {
            var test = await _context.Tests
                .Include(t => t.Questions)
                .ThenInclude(q => q.Options)
                .FirstOrDefaultAsync(t => t.Id == testId);
            if (test == null) throw new Exception("Test not found");
            var result = new Dictionary<int, List<GeneratedQuestion>>();
            var generatedQuestions = new List<GeneratedQuestion>();
            foreach (var question in test.Questions)
            {
                if (question.IsTemplate && !string.IsNullOrEmpty(question.TemplateText))
                {
                    var generated = GenerateFromTemplate(question);
                    generatedQuestions.Add(generated);
                }
                else
                {
                    generatedQuestions.Add(new GeneratedQuestion
                    {
                        OriginalQuestionId = question.Id,
                        Text = question.Text,
                        Type = question.Type,
```

```
                    Options = question.Options.ToList(),
                    CorrectAnswer = null
                });
            }
        }
        result.Add(testId, generatedQuestions);
        return result;
    }
    private GeneratedQuestion GenerateFromTemplate(Question template)
    {
        var parameters = JsonSerializer.Deserialize<List<ParameterConfig>>(template.ParametersConfig ??
"[]");
        var paramValues = new Dictionary<string, object>();
        foreach (var param in parameters!)
        {
            object value = param.Type switch
            {
                "int" => _random.Next(param.MinInt, param.MaxInt + 1),
                "double" => Math.Round(_random.NextDouble() * (param.MaxDouble - param.MinDouble) +
param.MinDouble, param.DecimalPlaces),
                "choice" => param.Choices![_random.Next(param.Choices.Count)],
                _ => 0
            };
            paramValues[param.Name] = value;
        }
        var generatedText = template.TemplateText!;
        foreach (var kv in paramValues)
        {
            generatedText = generatedText.Replace($"{{{kv.Key}}}", kv.Value.ToString());
        }
        string? correctAnswer = null;
        if (!string.IsNullOrEmpty(template.CorrectAnswerFormula))
        {
            correctAnswer = EvaluateFormula(template.CorrectAnswerFormula, paramValues);
        }
        return new GeneratedQuestion
        {
            OriginalQuestionId = template.Id,
            Text = generatedText,
            Type = template.Type,
            Options = new List<AnswerOption>(),
```

```
            CorrectAnswer = correctAnswer
        };
    }
    private string EvaluateFormula(string formula, Dictionary<string, object> parameters)
    {
        return formula;
    }
}
public class GeneratedQuestion
{
    public int OriginalQuestionId { get; set; }
    public string Text { get; set; } = string.Empty;
    public QuestionType Type { get; set; }
    public List<AnswerOption> Options { get; set; } = new();
    public string? CorrectAnswer { get; set; }
}
public class ParameterConfig
{
    public string Name { get; set; } = string.Empty;
    public string Type { get; set; } = string.Empty;
    public int MinInt { get; set; }
    public int MaxInt { get; set; }
    public double MinDouble { get; set; }
    public double MaxDouble { get; set; }
    public int DecimalPlaces { get; set; } = 2;
    public List<string> Choices { get; set; } = new();
}
}
```

Сервіс оцінювання відповідей (EvaluationService)

```
using MathNet.Symbolics;
using System.Globalization;
using Expr = MathNet.Symbolics.SymbolicExpression;
namespace TestSystem.Services
{
    public interface IEvaluationService
    {
        double EvaluateAnswer(string studentAnswer, string correctAnswer, double tolerance = 1e-6);
    }
    public class EvaluationService : IEvaluationService
    {
        public double EvaluateAnswer(string studentAnswer, string correctAnswer, double tolerance = 1e-6)
```

```
        {
            if (string.IsNullOrWhiteSpace(studentAnswer) || string.IsNullOrWhiteSpace(correctAnswer))
                return 0;
            studentAnswer = NormalizeInput(studentAnswer);
            correctAnswer = NormalizeInput(correctAnswer);
            if (double.TryParse(studentAnswer, NumberStyles.Any, CultureInfo.InvariantCulture, out double studentNum) &&
                double.TryParse(correctAnswer, NumberStyles.Any, CultureInfo.InvariantCulture, out double correctNum))
            {
                return Math.Abs(studentNum - correctNum) <= tolerance ? 1 : 0;
            }
            try
            {
                var studentExpr = Expr.Parse(studentAnswer);
                var correctExpr = Expr.Parse(correctAnswer);
                var simplifiedStudent = studentExpr.RationalSimplify();
                var simplifiedCorrect = correctExpr.RationalSimplify();
                return simplifiedStudent.Equals(simplifiedCorrect) ? 1 : 0;
            }
            catch
            {
                return 0;
            }
        }
        private string NormalizeInput(string input)
        {
            input = input.Trim().Replace(" ", "");
            input = input.Replace(',', '.');  // десяткова кома -> крапка
            if (input.Contains('/'))
            {
            }
            return input;
        }
    }
}
```

Сервіс бібліотеки формул (FormulaLibraryService)

```
using Microsoft.EntityFrameworkCore;
using TestSystem.Data;
using TestSystem.Models;
namespace TestSystem.Services
```

```
{
    public interface IFormulaLibraryService
    {
        Task<List<Formula>> GetAllFormulas(string? teacherId, string? category = null);
        Task<Formula?> GetFormulaById(int id);
        Task AddFormula(Formula formula);
        Task UpdateFormula(Formula formula);
        Task DeleteFormula(int id);
        Task<List<string>> GetCategories();
    }
    public class FormulaLibraryService : IFormulaLibraryService
    {
        private readonly ApplicationDbContext _context;

        public FormulaLibraryService(ApplicationDbContext context)
        {
            _context = context;
        }
        public async Task<List<Formula>> GetAllFormulas(string? teacherId, string? category = null)
        {
            var query = _context.Formulas.AsQueryable();
            if (!string.IsNullOrEmpty(teacherId))
                query = query.Where(f => f.TeacherId == teacherId || f.IsPublic);
            if (!string.IsNullOrEmpty(category))
                query = query.Where(f => f.Category == category);
            return await query.OrderBy(f => f.Name).ToListAsync();
        }
        public async Task AddFormula(Formula formula)
        {
            formula.CreatedAt = DateTime.UtcNow;
            _context.Formulas.Add(formula);
            await _context.SaveChangesAsync();
        }
        public async Task DeleteFormula(int id)
        {
            var formula = await _context.Formulas.FindAsync(id);
            if (formula != null)
            {
                _context.Formulas.Remove(formula);
                await _context.SaveChangesAsync();
            }
```

```
        }
        public async Task<List<string>> GetCategories()
        {
            return await _context.Formulas.Select(f => f.Category).Distinct().ToListAsync();
        }
    }
}
```

Контролер тестів (TestsController)

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using TestSystem.Models;
using TestSystem.Services;
namespace TestSystem.Controllers
{
    [Authorize]
    public class TestsController : Controller
    {
        private readonly ITestService _testService;
        private readonly ITestGenerationService _generationService;
        public TestsController(ITestService testService, ITestGenerationService generationService)
        {
            _testService = testService;
            _generationService = generationService;
        }
        [HttpGet]
        public async Task<IActionResult> Take(int id)
        {
            var userId = User.FindFirst(System.Security.Claims.ClaimTypes.NameIdentifier)?.Value;
            var variant = await _generationService.GenerateVariantAsync(id, userId!);
            return View(variant);
        }
        [HttpPost]
        public async Task<IActionResult> Submit(int id, Dictionary<int, string> answers)
        {
            var userId = User.FindFirst(System.Security.Claims.ClaimTypes.NameIdentifier)?.Value;
            var result = await _testService.EvaluateAndSaveAsync(id, userId!, answers);
            return RedirectToAction("Result", new { id = result.Id });
        }
    }
}
```

Інтерфейс вибору формул (JavaScript – фрагмент)

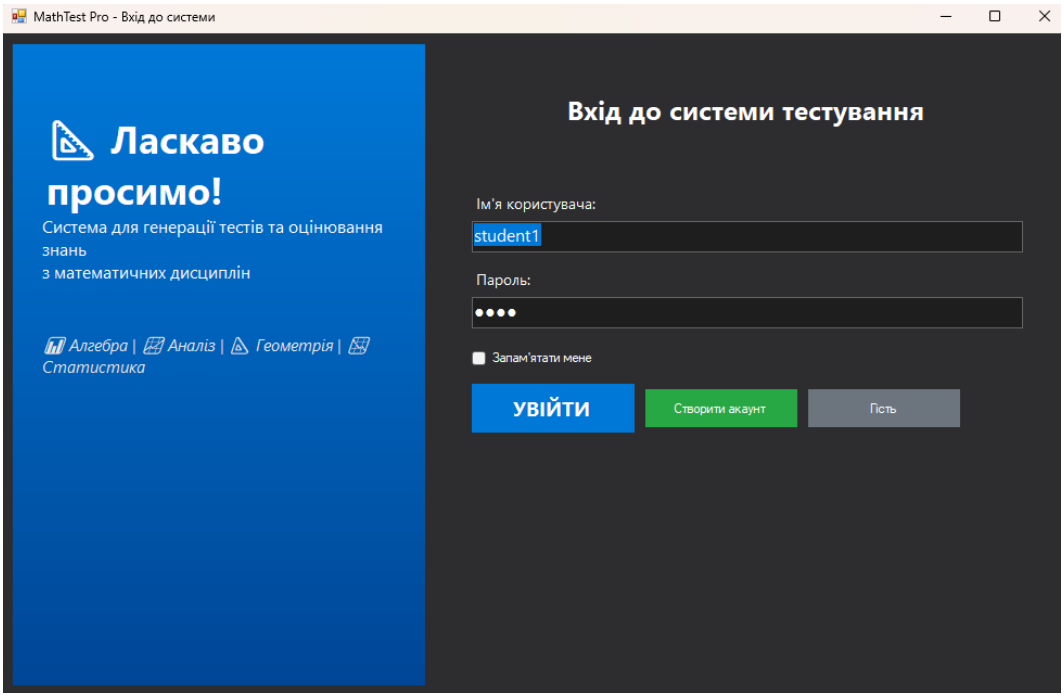
```
(function() {
    let currentCallback = null;
    window.showFormulaSelector = function(callback) {
        currentCallback = callback;
        $('#formulaModal').modal('show');
    };
    $('#formulaModal .select-formula').on('click', function() {
        const latex = $(this).data('latex');
        if (currentCallback) {
            currentCallback(latex);
        }
        $('#formulaModal').modal('hide');
    });
    function loadCategories() {
        $.get('/api/formulas/categories', function(data) {
            const tree = buildCategoryTree(data);
            $('#categoryTree').html(tree);
        });
    }
    function loadFormulas(category) {
        $.get(`/api/formulas/by-category?category=${encodeURIComponent(category)}`, function(data) {
            const cards = data.map(f => `
                <div class="formula-card" data-latex="${f.latexCode}">
                    <div class="formula-preview">${f.latexCode}</div>
                    <div class="formula-name">${f.name}</div>
                </div>
            `).join('');
            $('#formulasList').html(cards);
            MathJax.typesetPromise();
        });
    }
    loadCategories();
})();
```

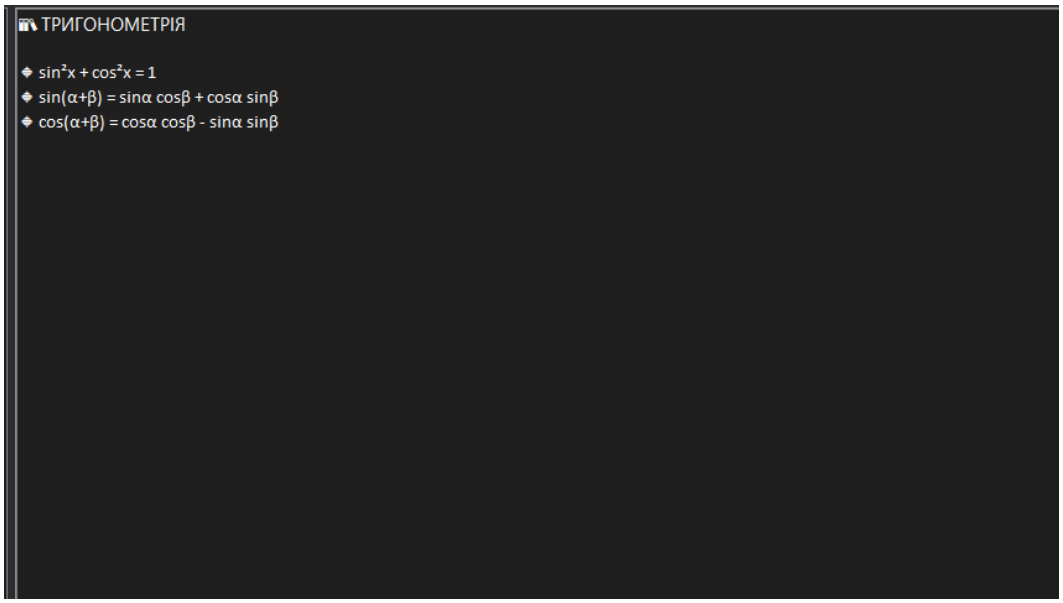
Приклад SQL-скрипту для створення таблиць (Entity Framework Core Migration)

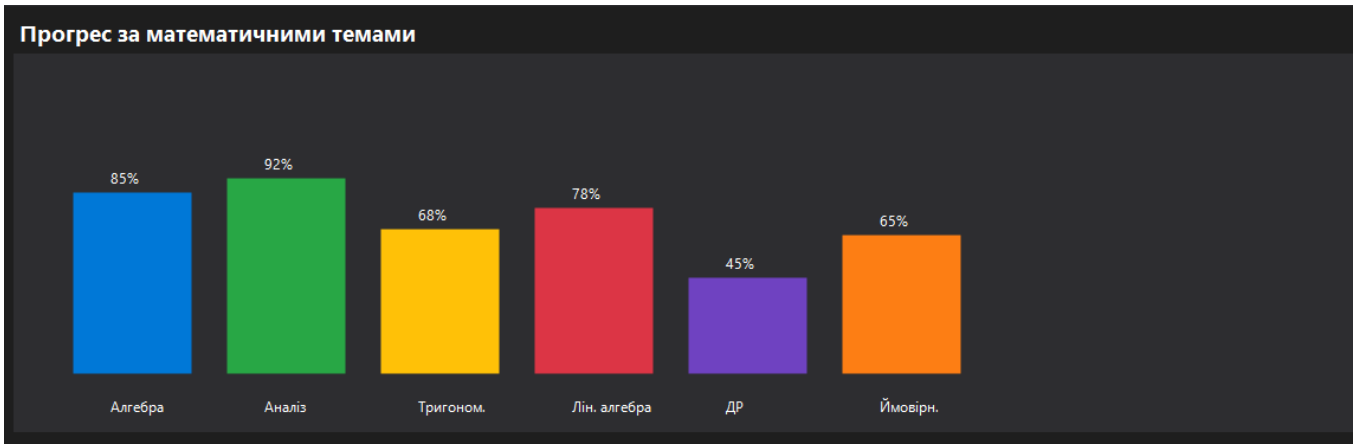
```
CREATE TABLE "Formulas" (
    "Id" INTEGER NOT NULL CONSTRAINT "PK_Formulas" PRIMARY KEY AUTOINCREMENT,
    "Name" TEXT NOT NULL,
    "Category" TEXT NOT NULL,
    "LatexCode" TEXT NOT NULL,
    "Description" TEXT NULL,
```

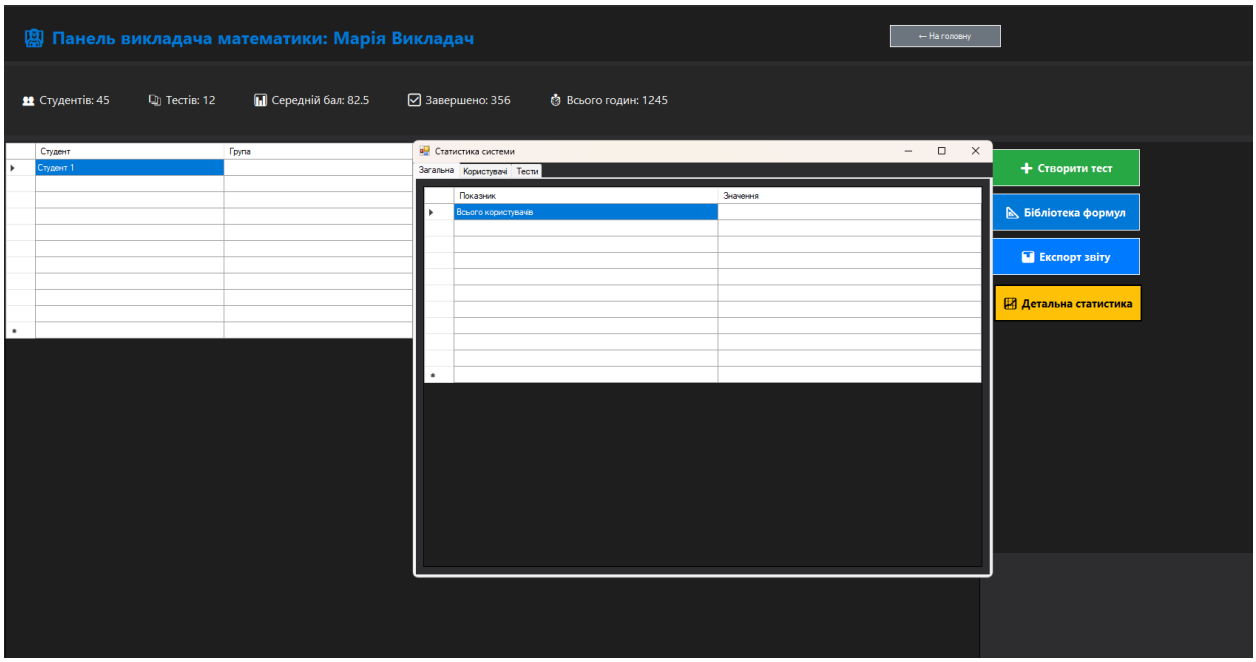
```
    "TeacherId" TEXT NULL,
    "IsPublic" INTEGER NOT NULL,
    "IsApproved" INTEGER NOT NULL,
    "CreatedAt" TEXT NOT NULL,
    CONSTRAINT "FK_Formulas_Users_TeacherId" FOREIGN KEY ("TeacherId") REFERENCES "AspNetUsers" ("Id") ON DELETE RESTRICT
);
CREATE TABLE "Tests" (
    "Id" INTEGER NOT NULL CONSTRAINT "PK_Tests" PRIMARY KEY AUTOINCREMENT,
    "Title" TEXT NOT NULL,
    "Description" TEXT NULL,
    "TimeLimitMinutes" INTEGER NULL,
    "MaxAttempts" INTEGER NOT NULL,
    "ShowCorrectAnswers" INTEGER NOT NULL,
    "TeacherId" TEXT NULL,
    "CreatedAt" TEXT NOT NULL,
    CONSTRAINT "FK_Tests_Users_TeacherId" FOREIGN KEY ("TeacherId") REFERENCES "AspNetUsers" ("Id") ON DELETE RESTRICT
);
```

ДОДАТОК Б. Результати впровадження (скріншоти роботи застосунку)






Прогрес за математичними темами
85%
92%
68%
78%
45%
65%
Алгебра
Аналіз
Тригоном.
Лін. алгебра
ДР
Ймовірн.


Панель викладача математики: Марія Викладач
← На головну
Студентів: 45
Тестів: 12
Середній бал: 82.5
Завершено: 356
Всього годин: 1245
Студент
Група
Студент 1
Статистика системи
Загальна
Користувачі
Тести
Показник
Значення
Всього користувачів
Створити тест
Бібліотека формул
Експорт звіту
Детальна статистика