ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО–КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ТЕХНОЛОГІЙ ЦИФРОВОГО РОЗВИТКУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: « Цифрова система практичного освоєння мови SQL з динамічною генерацією завдань та інтеграцією з Google Classroom»

на здобуття освітнього ступеня бакалавра
зі спеціальності: 121 «Інженерія програмного забезпечення»
освітньо –професійної програми: «Технології цифрового розвитку»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Назар Ступак
(підпис)

Виконав: здобувач вищої освіти гр. ТЦР –44
Назар Ступак

Керівник: Катерина Гордієнко

Рецензент: _____
науковий ступінь, вчене звання

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО–КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально –науковий інститут інформаційних технологій

Кафедра Технологій цифрового розвитку

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо –професійна програма «Технології цифрового розвитку»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Технологій цифрового розвитку
_____ Вікторія ЖЕБКА
« ____ » _____ 2026 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

_____
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: "Цифрова система практичного освоєння мови SQL з динамічною генерацією завдань та інтеграцією з Google Classroom» керівник кваліфікаційної роботи Катерина Гордієнко
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом Державного університету інформаційно –комунікаційних технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «___» _____ 2026 року.

3. Вихідні дані до кваліфікаційної роботи: технічне завдання на розробку веб – орієнтованої інтерактивної навчальної платформи для практичного освоєння мови SQL із динамічною генерацією завдань та інтеграцією з Google Classroom, методичні рекомендації до виконання бакалаврської роботи.

4. Зміст розрахунково–пояснювальної записки (перелік питань, які потрібно розробити)

Аналіз предметної області та аналогів

Проектування інформаційної системи

Реалізація та тестування системи

5. Перелік графічного матеріалу: у презентації подано основні етапи, результати та висновки кваліфікаційної роботи.

6. Дата видачі завдання «20» лютого 2026р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Ознайомлення з темою, збір та аналіз літератури	20.02.2026 – 25.02.2026	Виконано
2	Аналіз предметної області, огляд аналогів та порівняльний аналіз	26.02.2026 – 03.03.2026	Виконано
3	Визначення вимог, вибір технологічного стеку та проектування архітектури	04.03.2026 – 10.03.2026	Виконано
4	Проектування моделі даних, інтерфейсу та ключових механізмів	11.03.2026 – 17.03.2026	Виконано
5	Реалізація автентифікації, модулів керування БД та завданнями	18.03.2026 – 25.03.2026	Виконано
6	Реалізація автоматичної перевірки SQL та динамічної генерації завдань	26.03.2026 – 05.04.2026	Виконано
7	Реалізація статистики результатів, тестування системи та доопрацювання	06.04.2026 – 15.04.2026	Виконано
8	Оформлення пояснювальної записки, підготовка презентації та підготовка до захисту	16.04.2026 – 21.04.2026	Виконано

Здобувач(ка) вищої освіти _____ Назар Ступак

(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Катерина Гордієнко

(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 66 сторінок, 15 рисунків, 11 таблиць, 3 додатки, 39 джерел.

Мета роботи – розробити та реалізувати веб-орієнтовану інтерактивну навчальну платформу, яка забезпечує практичне освоєння мови SQL шляхом динамічної генерації індивідуальних наборів завдань, автоматичної перевірки SQL-запитів та інтеграції з Google Classroom.

Об'єктом дослідження є процес практичного навчання мови SQL у вищих навчальних закладах**.**

Предметом дослідження – методи та засоби автоматизації створення, розподілу та перевірки практичних завдань з SQL.

У роботі використано методи системного аналізу, порівняльного аналізу програмних аналогів, об'єктно-орієнтованого проектування, програмування на мові Python з використанням фреймворку Flask, а також методи тестування програмного забезпечення.

У першому розділі проведено аналіз сучасних підходів до навчання SQL та порівняльний огляд існуючих платформ (HackerRank, LeetCode, StrataScratch, LearnSQL.com та ін.), виявлено їх ключові недоліки. Другий розділ присвячено проектуванню системи: визначено функціональні та нефункціональні вимоги, обрано технологічний стек, розроблено архітектуру, модель даних та інтерфейс користувача. У третьому розділі описано процес реалізації основних модулів (автентифікація, керування базами даних і завданнями, автоматична перевірка запитів, динамічна генерація наборів, статистика результатів), проведено функціональне та безпекове тестування платформи.

Розроблена платформа дозволяє викладачам завантажувати власні SQLite-бази даних, створювати завдання, автоматично генерувати індивідуальні набори для кожного студента та отримувати детальну статистику виконання. Студенти отримують персоналізовані набори завдань та миттєвий зворотний зв'язок після виконання запитів. Реалізовано інтеграцію з Google Classroom.

Галузь застосування**:** вища освіта (спеціальність 121 «Інженерія програмного забезпечення»), системи дистанційного та змішаного навчання, курси підвищення кваліфікації.

КЛЮЧОВІ СЛОВА: SQL, ІНТЕРАКТИВНЕ НАВЧАННЯ, ДИНАМІЧНА ГЕНЕРАЦІЯ ЗАВДАНЬ, АВТОМАТИЧНА ПЕРЕВІРКА ЗАПИТІВ, FLASK, SQLITE, GOOGLE CLASSROOM, ОСВІТНЯ ПЛАТФОРМА, ІНДИВІДУАЛЬНІ НАБОРИ, ПРОГРЕС СТУДЕНТІВ.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ACID – Atomicity, Consistency, Isolation, Durability (атомарність, узгодженість, ізольованість, довговічність)

API – Application Programming Interface (інтерфейс програмування додатків)

CRUD – Create, Read, Update, Delete (створення, читання, оновлення, видалення)

CSS – Cascading Style Sheets (каскадні таблиці стилів)

DDL – Data Definition Language (мова визначення даних)

DML – Data Manipulation Language (мова маніпуляції даними)

ER–діаграма – Entity –Relationship diagram (діаграма «сутність –зв'язок»)

HTML – HyperText Markup Language (мова розмітки гіпертексту)

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту)

JSON – JavaScript Object Notation (нотація об'єктів JavaScript)

LMS – Learning Management System (система управління навчанням)

MVC – Model –View –Controller (модель –подання –контролер)

OAuth – Open Authorization (відкрита авторизація)

POST – метод HTTP для надсилання даних

GET – метод HTTP для отримання даних

SPA – Single Page Application (односторінковий додаток)

SQL – Structured Query Language (мова структурованих запитів)

ЗМІСТ

ВСТУП

Сучасний ландшафт інформаційних технологій постійно змінюється, а це означає нескінченне зростання потреби в експертах, які володіють навичками роботи з даними. SQL (мова структурованих запитів) є одним із основних інструментів під час роботи з реляційними базами даних, і це зумовлює значне включення SQL у підготовку фахівців з таких сфер, як розробка програмного забезпечення, аналіз даних, бізнес –аналітика та багатьох інших. На основі кількох досліджень і звітів (наприклад, опитування Stack Overflow Developer Survey) знання SQL є однією з трьох найпопулярніших за попитом навичок на ринку праці, поряд з Python та JavaScript.

Лекції, підручники, наукові роботи та інші статичні методи навчання SQL зазвичай не забезпечують достатнього практичного досвіду. Проблема полягає в тому, що студентам бракує індивідуальних завдань, автоматичної перевірки результатів і реальної бази даних у безпечному середовищі. Крім того, інтеграція навчальних інструментів безпосередньо в освітній процес із частим використанням у платформах керування навчанням, зокрема Google Classroom як широко застосовуваному рішенні для дистанційного та змішаного навчання, суттєво зросла після пандемії COVID –19.

Існуючі інтерактивні платформи (наприклад, LeetCode, HackerRank, SQLZoo, Vertabelo Academy) ґрунтуються на припущенні про заздалегідь визначений набір завдань, що періодично оновлюється, де викладачі мають лише кілька варіантів для створення власного контенту. У цьому контексті ключовими елементами, яких бракує в освітньому процесі, є механізм динамічної генерації персоналізованих наборів завдань, глибока інтеграція з системами керування навчанням та створення гнучкого механізму автоматизованого оцінювання, зумовленого базами даних, завантаженими на платформу. Актуальним є завдання створення конкретної вебплатформи, присвяченої саме вивченню практичної SQL –спроможності, адаптованої під потреби як викладачів, так і студентів: це допомагає підвищити

ефективність вивчення цього предмета та виконати його персоналізацію до індивідуальних потреб.

Мета роботи – розробити інтерактивну веб –орієнтовану навчальну платформу для практичного освоєння мови SQL з можливістю динамічної генерації індивідуальних наборів завдань, автоматичної перевірки SQL –запитів та інтеграції з Google Classroom.

Для досягнення поставленої мети визначено такі завдання:

- провести аналіз предметної області та існуючих аналогів;
- визначити функціональні та нефункціональні вимоги до системи;
- обрати технологічний стек та спроектувати архітектуру платформи;
- розробити модель даних та механізми динамічної генерації завдань і автоматичної перевірки;
- реалізувати основні модулі системи (керування базами даних, задачами, наборами завдань, автентифікацію, відстеження результатів);
- спроектувати та частково реалізувати інтеграцію з Google Classroom;

Об'єктом дослідження є процес практичного навчання мови SQL в умовах вищої освіти з використанням інтерактивних веб –платформ.

Предметом дослідження є веб –орієнтована платформа для автоматизованого створення, розподілу та перевірки практичних завдань з SQL з елементами динамічної генерації та інтеграції з системами управління навчанням.

У роботі використано такі методи дослідження: аналіз наукової та технічної літератури, порівняльний аналіз аналогів, методи системного аналізу та проектування програмного забезпечення, методи об'єктно –орієнтованого програмування, тестування програмного забезпечення.

Розроблена платформа може бути використана в навчальному процесі вищих навчальних закладів, на курсах підвищення кваліфікації, а також у самостійному вивченні SQL.

Структура роботи: робота складається зі вступу, трьох основних розділів, висновків, списку використаних джерел та додатків. У першому розділі проведено аналіз предметної області та аналогів. Другий розділ присвячено проектуванню

системи. Третій розділ містить опис реалізації та тестування. У додатках наведено скріншоти інтерфейсу, ER –діаграму та приклади коду. Загальний обсяг роботи – 87 сторінок.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА АНАЛОГІВ

1.1. Сучасні підходи до навчання мови SQL

Мова структурованих запитів SQL (Structured Query Language) є фундаментальним інструментом для роботи з реляційними базами даних і залишається однією з найбільш затребуваних навичок у сфері інформаційних технологій. Згідно з численними звітами та опитуваннями розробників, знання SQL входить до переліку ключових компетенцій для аналітиків даних, розробників програмного забезпечення та спеціалістів з бізнес –аналітики. Сучасні підходи до навчання SQL еволюціонували від традиційних методів, таких як лекції та підручники, до інтерактивних, практико –орієнтованих і гейміфікованих форм, що значно підвищують ефективність засвоєння матеріалу [7].

Традиційно навчання SQL базувалося на теоретичному викладі синтаксису та структури запитів через книги, університетські курси та статичні вправи. Такі методи забезпечували міцну теоретичну базу, але часто не давали достатньої практики, що призводило до труднощів у застосуванні знань на реальних даних. З появою цифрових технологій акцент змістився на інтерактивне навчання, де студенти можуть відразу виконувати запити в онлайн –середовищі та отримувати миттєвий зворотний зв'язок. Цей підхід дозволяє уникнути типових помилок початківців, таких як неправильне розуміння JOIN чи агрегатних функцій, і сприяє глибшому розумінню [8].

Одним з ключових сучасних підходів є використання інтерактивних онлайн-платформ, які пропонують практичні вправи з реальними або симульованими базами даних. Такі ресурси, як SQLBolt, SQLZoo чи HackerRank, дозволяють користувачам писати запити безпосередньо в браузері, перевіряти результати та прогресувати від простих SELECT – запитів до складних підзапитів і оптимізації. Цей метод навчання через практику (learning by doing) вважається одним з

найефективніших, оскільки імітує реальні робочі сценарії та розвиває навички вирішення проблем [9].

Інший важливий напрям – гейміфікація навчання, коли елементи ігор (бали, рівні, досягнення, лідерборди) інтегруються в освітній процес. Платформи на кшталт LeetCode чи Codec.org застосовують гейміфікацію для мотивації студентів: розв'язання завдань приносить очки, а конкуренція з іншими користувачами стимулює регулярну практику. Дослідження показують, що гейміфіковані підходи підвищують залученість на 30–50% порівняно з традиційними методами, особливо серед молодшого покоління, яке звикло до ігрових механік [10].

Адаптивне навчання, засноване на штучному інтелекті, є ще одним інноваційним підходом. Платформи, такі як DataCamp чи Vertabelo Academy, аналізують відповіді користувача та пропонують індивідуальні траєкторії: якщо студент помиляється в агрегатних функціях, система автоматично додає додаткові вправи на цю тему. Це дозволяє адаптувати матеріал до рівня знань і стилю навчання кожного учня, роблячи процес ефективнішим і менш фруструючим [11].

Крім того, сучасні методи включають інтеграцію SQL з іншими технологіями, такими як Python (наприклад, через бібліотеки pandas чи SQLAlchemy) або візуалізацію даних у інструментах на кшталт Tableau. Курси на Coursera чи Udemy часто поєднують SQL з реальними проектами, де студенти будують бази даних для аналізу даних з відкритих джерел. Такий контекстуальний підхід допомагає зрозуміти практичну цінність SQL у сучасних професіях, таких як data science чи backend –розробка.

У контексті вищої освіти та професійного розвитку дедалі популярнішими стають змішані (blended learning) моделі, де онлайн –платформи доповнюють офлайн –заняття. Викладачі можуть використовувати інструменти на кшталт Google Classroom для розподілу завдань, а студенти – виконувати їх на інтерактивних сайтах. Це сприяє формуванню навичок самостійного навчання та співпраці.

Сучасні підходи до навчання SQL характеризуються переходом від пасивного засвоєння теорії до активної практики, персоналізації та мотивації через

ігрові елементи. Вони роблять вивчення доступнішим і ефективнішим, відповідаючи вимогам цифрової епохи.

1.2. Огляд існуючих платформ для практичного освоєння SQL

Існує значна кількість інтерактивних онлайн – платформ, призначених для практичного освоєння мови SQL. Ці ресурси пропонують користувачам можливість виконувати запити безпосередньо в браузері, отримувати миттєву перевірку результатів та прогресувати через структуровані курси або набори завдань. Більшість платформ орієнтовані на різні рівні користувачів – від початківців до досвідчених фахівців – і використовують реальні або симульовані бази даних для вправ. Огляд основних платформ дозволяє оцінити їх функціональні можливості та виявити спільні тенденції в розвитку таких інструментів [19].

Однією з найвідоміших платформ є HackerRank, яка пропонує великий набір SQL – завдань різної складності. Користувачі можуть практикувати базові операції (SELECT, WHERE), агрегатні функції, JOIN та складні запити на даних з реальних сценаріїв, таких як аналіз продажів чи обробка логів. Перевірка результатів відбувається автоматично шляхом порівняння виведення запиту з еталонним. Платформа підтримує лідерборди та сертифікацію, що мотивує до конкуренції. HackerRank інтегрується з процесами найму на роботу, дозволяючи компаніям тестувати кандидатів [19].

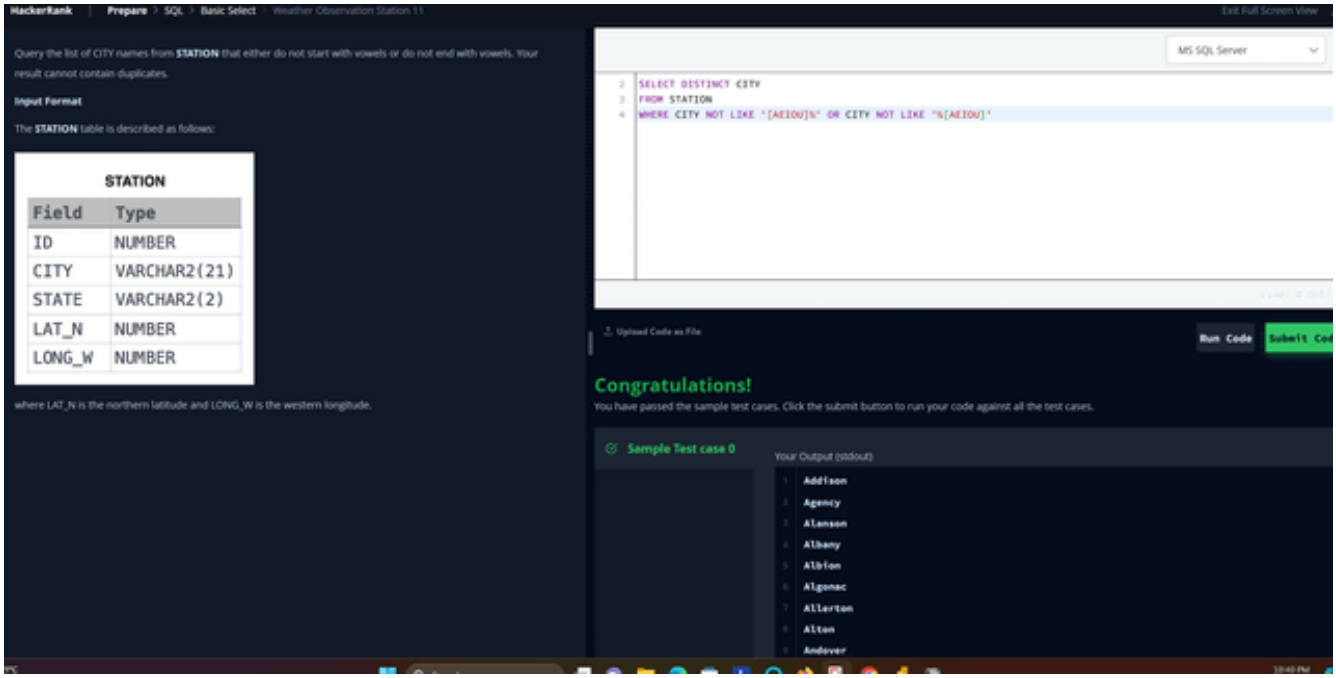

Рис. 0.1 Інтерфейс платформи HackerRank для виконання SQL –завдань

Ще одна популярна платформа – LeetCode – фокусується на алгоритмічних та базових завданнях, включаючи великий розділ SQL. Завдання поділені за рівнями складності (Easy, Medium, Hard) і часто базуються на реальних інтерв'ю в компаніях, таких як Google чи Amazon. Користувачі пишуть запити в онлайн – редакторі, отримують миттєвий фідбек та можуть переглядати рішення інших. LeetCode підтримує обговорення завдань у спільноті, що сприяє глибшому розумінню альтернативних підходів до розв'язання [20].

StrataScratch виділяється орієнтацією на реальні бізнес – кейси. Платформа містить тисячі завдань на основі даних з компаній, таких як Airbnb чи Netflix, з акцентом на аналітичні запити (window functions, CTE). Автоматична перевірка враховує не лише правильність результату, а й ефективність запиту. Доступні курси та проекти для портфоліо роблять її корисною для підготовки до роботи аналітиком даних [19].

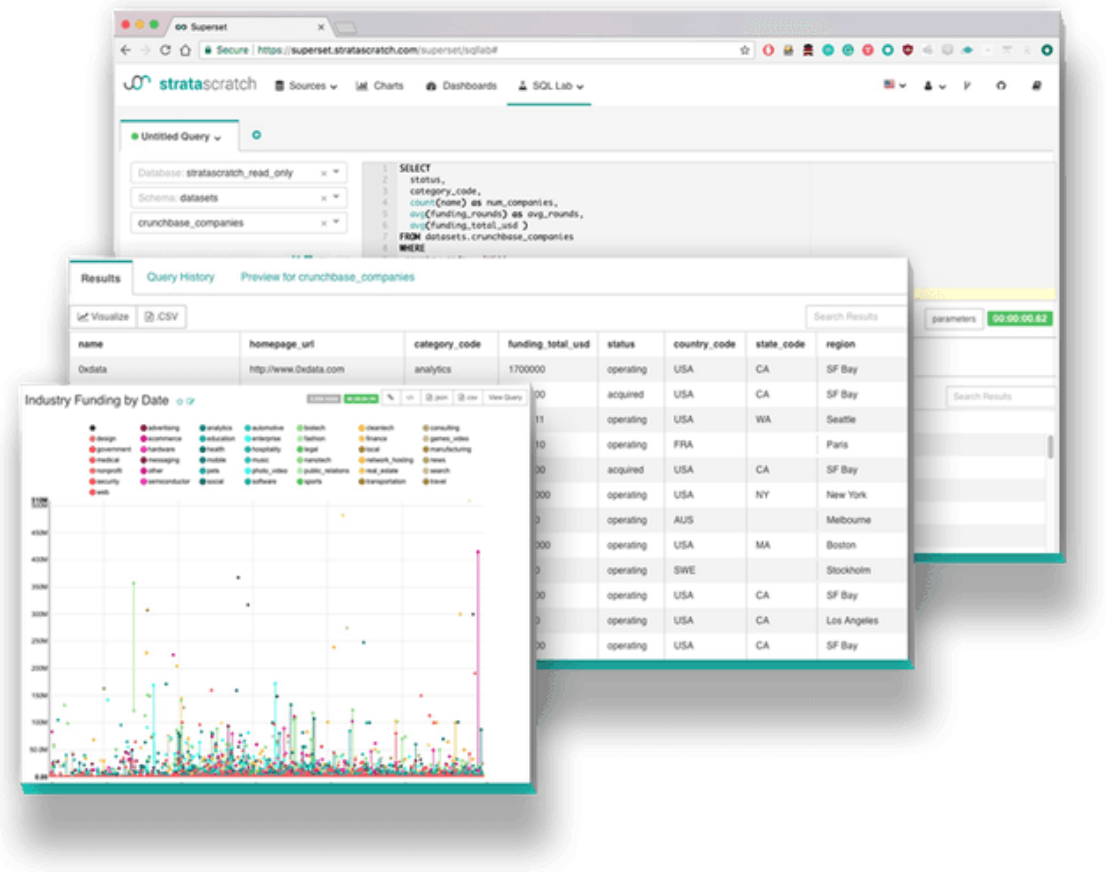

Рис. 1.2 Приклад бізнес-завдання на платформі StrataScratch

LearnSQL.com (від Vertabelo) пропонує структуровані курси з інтерактивними вправами. Користувачі працюють з віртуальними базами даних, виконуючи запити крок за кроком. Платформа акцентує на прогресивному навчанні: від базового синтаксису до складних тем, таких як оптимізація запитів. Особливістю є можливість створення власних треків навчання та детальний аналіз помилок [20].

Vertabelo Academy надає безкоштовні та платні курси з великою кількістю інтерактивних вправ. Кожне завдання супроводжується поясненнями та візуалізацією схеми бази даних. Платформа підтримує кілька діалектів SQL (PostgreSQL, MySQL, Oracle), що корисно для вивчення відмінностей. Додатково доступні блоги та ресурси для поглибленого вивчення [10].

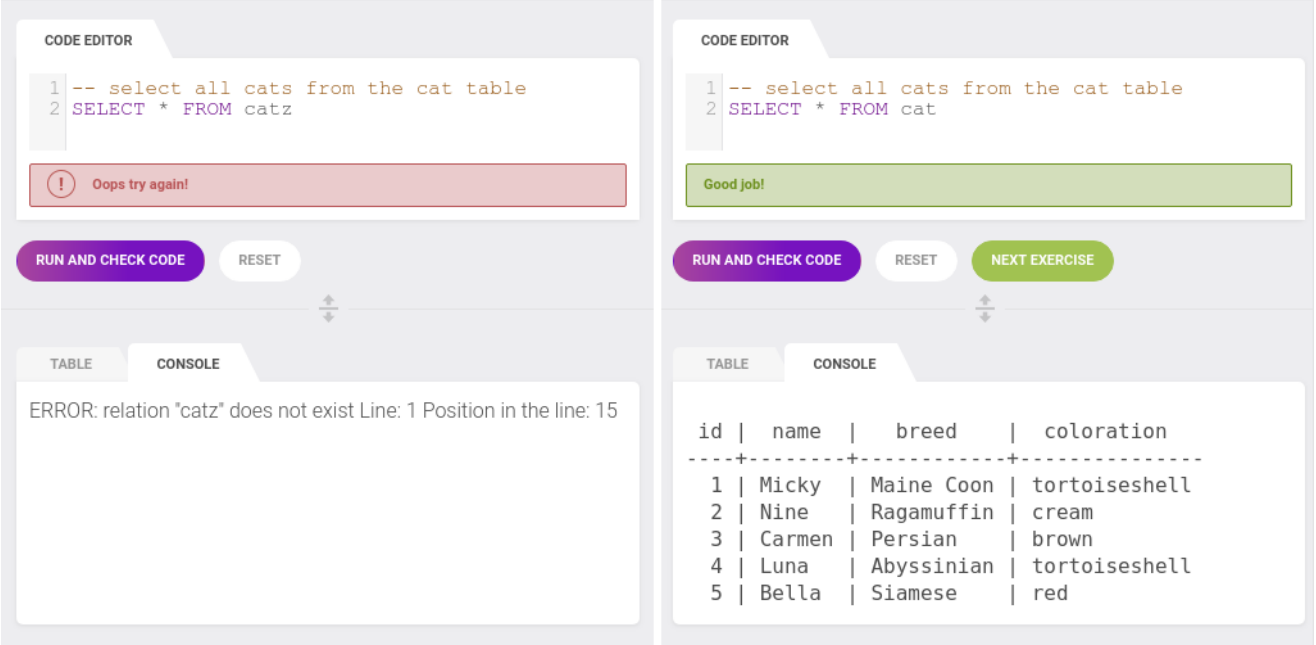

Рис. 1.3 Інтерактивний редактор запитів на Vertabelo Academy

SQLZoo – одна з найстаріших платформ – пропонує прості інтерактивні туторіали з вправами на стандартних базах даних (наприклад, world чи nobel). Користувачі можуть відразу тестувати запити та бачити результати. Платформа безкоштовна, не вимагає реєстрації для базових вправ і фокусується на класичних темах: SELECT, JOIN, агрегати [10].

DataCamp та Mode Analytics пропонують інтеграцію SQL з аналізом даних. DataCamp поєднує SQL з R та Python у проєктно –орієнтованих курсах, тоді як Mode надає реальні датасети для практики аналітичних запитів з візуалізацією результатів.

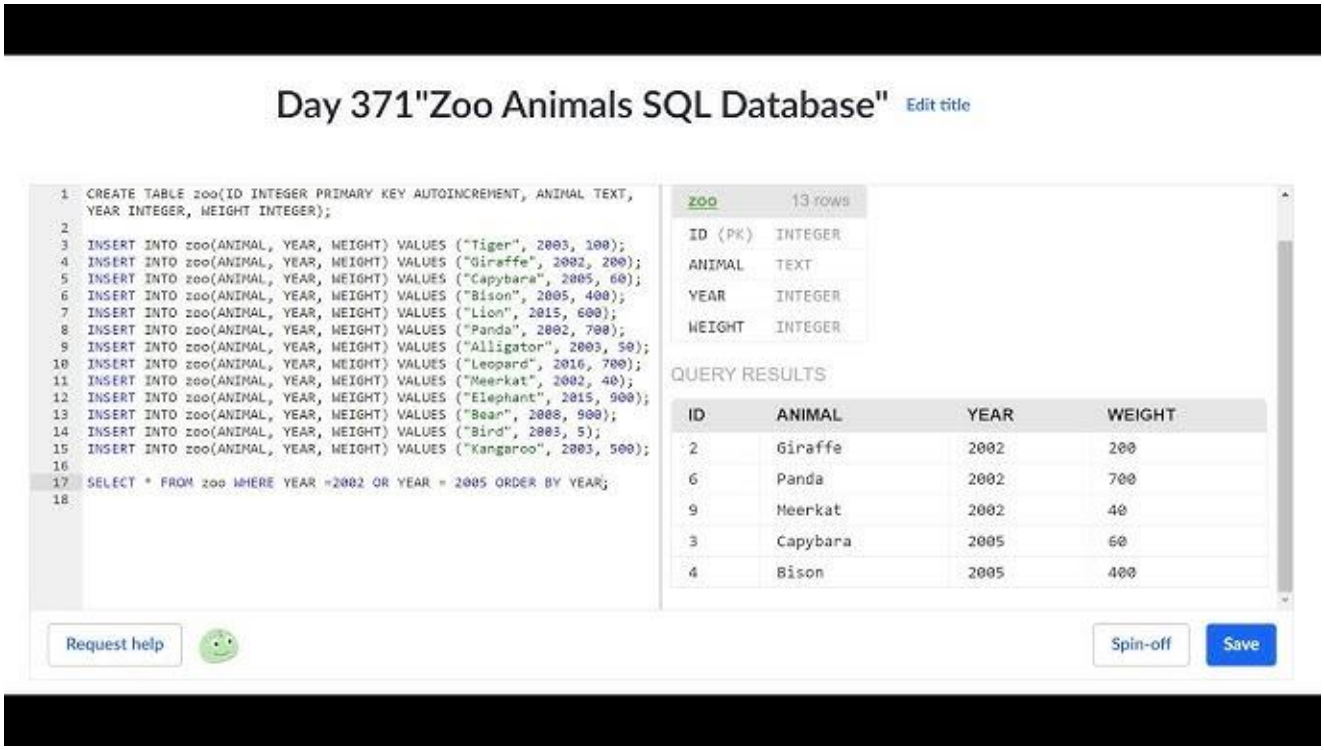

Рис. 1.4 Вправа з базою даних на платформі SQLZoo

Загалом, існуючі платформи забезпечують потужні інструменти для практичного освоєння SQL, з автоматичною перевіркою та різноманітними завданнями. Однак більшість з них використовують фіксовані бази даних і не дозволяють викладачам завантажувати власні датасети чи генерувати індивідуальні набори завдань для груп студентів. Інтеграція з системами управління навчанням (наприклад, Google Classroom) також обмежена або відсутня, що ускладнює використання в організованому освітньому процесі [19; 20; 10].

1.3. Порівняльний аналіз функціональних можливостей аналогів

Для об’єктивної оцінки можливостей існуючих інтерактивних платформ проведено порівняльний аналіз за ключовими функціональними характеристиками, що є критичними для практичного освоєння SQL у освітньому процесі. До аналізу включено платформи, які були розглянуті в попередньому підрозділі: HackerRank, LeetCode, StrataScratch, LearnSQL.com (Vertabelo), SQLZoo та DataCamp. Основні критерії порівняння: автоматична перевірка SQL –запитів, можливість створення власних завдань викладачем, підтримка завантаження

користувацьких баз даних, динамічна генерація індивідуальних наборів завдань, інтеграція з системами управління навчанням (LMS, зокрема Google Classroom), гейміфікація та лідерборди, а також доступність (безкоштовний/платний доступ). Ці критерії обрано з урахуванням вимог до гнучкості та адаптації платформи для використання викладачами в організованому навчанні [19; 20].

Таблиця 1.1

Порівняльна характеристика функціональних можливостей інтерактивних платформ для освоєння SQL

Критерій	HackerRank	LeetCode	StrataScratch	LearnSQL.com (Vertabelo)	SQLZoo	DataCamp
Автоматична перевірка запитів	Так	Так	Так	Так	Так	Так
Створення власних завдань викладачем	Ні	Ні	Частково	Ні	Ні	Частково
Завантаження власних баз даних	Ні	Ні	Ні	Ні	Ні	Ні
Динамічна генерація індивідуальних наборів	Ні	Ні	Ні	Ні	Ні	Частково
Інтеграція з LMS (Google Classroom)	Ні	Ні	Ні	Ні	Ні	Частково
Гейміфікація та лідерборди	Так	Так	Так	Частково	Ні	Так
Безкоштовний доступ	Частково	Частково	Частково	Частково	Так	Частково

Як видно з таблиці, усі розглянуті платформи забезпечують автоматичну перевірку SQL – запитів шляхом порівняння результатів з еталонними, що є базовою функцією для інтерактивного навчання. HackerRank та LeetCode виділяються сильною гейміфікацією та орієнтацією на підготовку до співбесід, з лідербордами та рівнями складності, але не дозволяють викладачам створювати чи

кастомізувати завдання [19]. StrataScratch та DataCamp пропонують часткову підтримку створення власного контенту (наприклад, проєкти чи кастомні датасети у преміум –версіях), але без повноцінного завантаження користувацьких баз даних чи динамічної рандомізації для груп студентів [19].

LearnSQL.com та Vertabelo Academy фокусуються на структурованих курсах з детальними поясненнями, але завдання фіксовані, без можливостей для викладачів генерувати індивідуальні набори чи інтегрувати платформу з зовнішніми LMS [20]. SQLZoo є повністю безкоштовним і простим у використанні, але обмежується статичними вправами без гейміфікації чи персоналізації [10]. Жодна з платформ не підтримує завантаження власних баз даних викладачем чи повноцінну інтеграцію з Google Classroom, що обмежує їх застосування в груповому навчанні. Часткова інтеграція в DataCamp можлива через API для корпоративних клієнтів, але не для індивідуальних викладачів.

Порівняльний аналіз демонструє, що існуючі рішення добре підходять для самостійного навчання чи підготовки до співбесід, але мають суттєві обмеження щодо кастомізації контенту викладачем, індивідуалізації завдань для студентів та інтеграції з освітніми екосистемами. Це підтверджує необхідність розробки нової платформи, яка б поєднувала сильні сторони аналогів з відсутніми функціями, такими як динамічна генерація та глибока інтеграція.

1.4. Виявлення недоліків існуючих рішень та обґрунтування необхідності розробки нової платформи

На основі огляду та порівняльного аналізу існуючих інтерактивних платформ для практичного освоєння SQL можна виділити низку спільних недоліків, які обмежують їх ефективне використання в організованому освітньому процесі, особливо в умовах вищої освіти. Більшість платформ орієнтовані переважно на самостійне навчання або підготовку до співбесід, а не на групову роботу викладача зі студентами. Основні проблеми пов’язані з відсутністю гнучкості для викладачів, обмеженою персоналізацією завдань та слабкою інтеграцією з сучасними системами управління навчанням [19; 20].

Таблиця 1.2

Основні недоліки існуючих інтерактивних платформ для освоєння SQL

Недолік	HackerRank	LeetCode	StrataScratch	LearnSQL.com (Vertabelo)		SQLZoo		DataCamp
Відсутність завантаження власних баз даних викладачем	Так	Так	Так	Так		Так		Так
Неможливість створення власних завдань викладачем	Так	Так	Частково	Так		Так		Частково
Відсутність динамічної генерації індивідуальних наборів для студентів	Так	Так	Так	Так		Так		Так
Обмежена або відсутня інтеграція з Google Classroom/LMS	Так	Так	Так	Так		Так		Частково
Фіксовані бази даних без можливості кастомізації			Так	Так	Так	Так	Так	Частково
Орієнтація на індивідуальне навчання, а не групове			Так	Так	Так	Так	Так	Частково

Як показано в таблиці, ключовим недоліком більшості платформ є неможливість завантаження власних баз даних викладачем, що обмежує використання реальних або спеціалізованих датасетів, адаптованих до конкретного курсу чи теми. Наприклад, HackerRank та LeetCode пропонують лише фіксовані сценарії, орієнтовані на інтерв’ю, без можливості створення індивідуальних завдань для групи студентів [19]. StrataScratch та DataCamp дозволяють часткову

кастомізацію в преміум –версіях, але не підтримують повноцінне завантаження користувацьких баз чи динамічну рандомізацію наборів для запобігання списуванню [19].

Іншим суттєвим недоліком є відсутність динамічної генерації індивідуальних наборів завдань. У груповому навчанні це призводить до однакових завдань для всіх студентів, що знижує об’єктивність оцінки та стимулює нечесну поведінку. Жодна з платформ не реалізує автоматичну рандомізацію на основі критеріїв (тема, складність, кількість), що є критичним для практичних занять у великих групах [20].

Інтеграція з системами управління навчанням, такими як Google Classroom, практично відсутня. DataCamp пропонує часткову інтеграцію через API для корпоративних клієнтів, але не для індивідуальних викладачів, а інші платформи взагалі не підтримують автоматичну передачу завдань чи оцінок. Це ускладнює використання в змішаному навчанні, де викладач потребує єдиної екосистеми для розподілу завдань та моніторингу прогресу [10].

Крім того, більшість платформ не надають викладачам інструментів для детального відстеження прогресу групи (наприклад, статистика по студентах з можливістю ручного коригування оцінок) та фокусуються на індивідуальному досвіді. SQLZoo, будучи повністю безкоштовним, є надто простим і не підтримує гейміфікацію чи персоналізацію, що знижує мотивацію студентів у довгостроковій перспективі [10].

Виявлені недоліки обґрунтовують необхідність розробки нової інтерактивної платформи, яка б усунула ці прогалини. Запропонована система дозволить викладачам завантажувати власні бази даних (SQLite), створювати та категоризовувати завдання, генерувати індивідуальні набори з рандомізацією за критеріями (тема, складність, кількість), автоматично перевіряти запити та інтегруватися з Google Classroom для розподілу завдань і передачі оцінок. Такий підхід забезпечить гнучкість, персоналізацію та ефективний контроль у груповому навчанні, підвищивши якість практичного освоєння SQL в освітньому процесі.

1.5. Висновки

Проведений аналіз предметної області показав, що мова SQL залишається однією з ключових навичок у сфері інформаційних технологій, а сучасні підходи до її вивчення еволюціонували від традиційних теоретичних методів до інтерактивних, практико-орієнтованих і Whiteміфікованих форм навчання. Інтерактивні платформи, адаптивне навчання та інтеграція з іншими інструментами значно підвищують ефективність засвоєння матеріалу, роблячи процес більш мотивуючим і адаптованим до індивідуальних потреб.

Огляд існуючих платформ (HackerRank, LeetCode, StrataScratch, LearnSQL.com, SQLZoo, DataCamp та інші) демонструє їх сильні сторони в автоматичній перевірці запитів, різноманітті завдань та гейміфікації, що сприяє самостійному навчанню та підготовці до професійної діяльності. Порівняльний аналіз функціональних можливостей виявив спільні переваги, такі як миттєвий зворотний зв’язок і структуровані вправи, але також підкреслив суттєві обмеження: відсутність підтримки завантаження власних баз даних, створення кастомних завдань викладачем, динамічної генерації індивідуальних наборів та повноцінної інтеграції з системами управління навчанням, зокрема Google Classroom.

Виявлені недоліки – фіксовані датасети, відсутність персоналізації для групового навчання та слабка адаптація до освітнього процесу – обмежують використання цих платформ у вищих навчальних закладах. Це обґрунтовує необхідність розробки нової інтерактивної веб-платформи, яка б поєднувала автоматичну перевірку SQL-запитів на основі завантажених баз даних, динамічну рандомізацію завдань для індивідуалізації, інструменти для викладачів та інтеграцію з Google Classroom. Така система дозволить усунути прогалини аналогів і підвищити ефективність практичного освоєння SQL в організованому навчанні.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Визначення вимог до системи

Визначення вимог є ключовим етапом проектування інформаційної системи, оскільки воно дозволяє чітко окреслити функціональні можливості платформи, а також забезпечити її надійність, зручність та ефективність у використанні. Функціональні вимоги описують, що саме система повинна виконувати, тобто конкретні функції для користувачів різних ролей (викладач, студент). Нефункціональні вимоги визначають якісні характеристики системи, такі як продуктивність, безпека, масштабованість та інтерфейс. Вимоги сформульовано на основі аналізу предметної області, недоліків аналогів та технічного завдання до платформи, з урахуванням специфіки практичного навчання SQL у вищих навчальних закладах.

Функціональні вимоги поділено на групи залежно від ролей користувачів та основних модулів системи. Вони охоплюють керування базами даних, створення та перевірку завдань, генерацію індивідуальних наборів, автентифікацію та інтеграцію з Google Classroom.

Таблиця 2.1

Функціональні вимоги до системи

Код вимоги	Опис вимоги	Роль користувача
ФВ – 01	Автентифікація та реєстрація користувачів з розрізненням ролей (викладач/студент)	Викладач, студент
ФВ – 02	Завантаження та керування SQLite-базами даних (завантаження файлу, перегляд схеми, активація)	Викладач
ФВ – 03	Створення, редагування, видалення та категоризація завдань (тема, підтема, складність, еталонний запит)	Викладач
ФВ – 04	Автоматична перевірка SQL-запитів студентів шляхом порівняння результатів з еталонними	Студент

Продовження таблиці 2.1

ФВ – 05	Динамічна генерація індивідуальних наборів завдань за критеріями (кількість, теми, складність) з рандомізацією	Викладач
ФВ – 06	Доступ до призначених наборів завдань за унікальним кодом або через дашборд	Студент
ФВ – 07	Відстеження прогресу студентів (статистика, історія спроб, відсоток правильних відповідей)	Викладач, студент
ФВ – 08	Інтеграція з Google Classroom (отримання списку студентів, публікація завдань, передача оцінок)	Викладач
ФВ – 09	Збереження історії спроб студентів з відображенням результатів та запитів	Студент, викладач

Ці вимоги забезпечують повний цикл роботи з практичними завданнями: від створення контенту викладачем до виконання та перевірки студентом. Особливий акцент зроблено на динамічній генерації наборів (ФВ – 05), що запобігає списуванню та індивідуалізує навчання, а також на інтеграції з Google Classroom (ФВ – 08), яка спрощує використання платформи в реальному освітньому процесі.

Нефункціональні вимоги визначають критерії якості системи та обмеження, що впливають на її експлуатацію. Вони сформульовані з урахуванням обмеженого технологічного стеку (Flask, SQLite) та вимог до безпеки при роботі з SQL-запитами.

Таблиця 2.2

Нефункціональні вимоги до системи

Код вимоги	Опис вимоги	Критерій оцінки
НФВ – 01	Безпека виконання SQL-запитів (тільки SELECT, read –only режим, таймаут)	Відсутність виконання DDL/DML

Продовження таблиці 2.2

НФВ – 02	Зручність інтерфейсу (інтуїтивність, адаптивність до мобільних пристроїв)	Відповідність Bootstrap 5
НФВ – 03	Продуктивність (час виконання запиту < 5 с, підтримка до 100 одночасних користувачів)	Тестування навантаження
НФВ – 04	Надійність (збереження даних, обробка помилок, резервне копіювання)	Відсутність втрати даних
НФВ – 05	Масштабованість (можливість розширення для інших СУБД у майбутньому)	Модульна архітектура
НФВ – 06	Доступність (підтримка сучасних браузерів, українська мова інтерфейсу)	Крос –браузерна сумісність
НФВ – 07	Конфіденційність даних (хешування паролів, захист сесій)	Відповідність OWASP

Нефункціональні вимоги забезпечують безпечну та стабільну роботу системи, особливо в контексті виконання користувацьких SQL-запитів, де ризики SQL-ін’єкцій чи пошкодження даних мінімізовано через обмеження на тип запитів та read-only доступ до баз. Вимоги до продуктивності та зручності враховують використання платформи в навчальних групах середнього розміру.

Сформульовані вимоги стали основою для подальшого проектування архітектури, моделі даних та інтерфейсу системи. Вони дозволяють створити гнучку, безпечну та орієнтовану на користувача платформу, яка усуває недоліки аналогів і відповідає потребам практичного навчання SQL.

2.2. Вибір технологічного стеку

2.2.1. Мова програмування та фреймворк

Для реалізації веб-орієнтованої платформи обрано мову програмування Python у поєднанні з фреймворком Flask. Python є однією з найбільш популярних мов програмування завдяки своїй простоті, читабельності синтаксису та великій екосистемі бібліотек, що значно прискорює розробку. Він ідеально підходить для прототипування та створення веб-додатків середньої складності, що відповідає

обсягу бакалаврської роботи. Мова має потужну стандартну бібліотеку для роботи з файлами, JSON та базами даних, а також широку підтримку спільноти, що полегшує вирішення типових завдань, таких як обробка запитів та виконання SQL-коду.

Обґрунтуванням вибору Python стало його домінування в освітніх та наукових проєктах, де акцент робиться на швидкій розробці та зрозумілості коду. Альтернативи, такі як JavaScript (з Node.js) чи Java, є більш громіздкими для швидкого прототипу, а PHP менш сучасний для нових проєктів. Python забезпечує легку інтеграцію з SQLite через вбудований модуль sqlite3, що критично для виконання користувацьких запитів у безпечному режимі.

Серед фреймворків для Python розглянуто Django та FastAPI, але обрано Flask через його мінімалістичність і гнучкість. Flask є легковаговим мікрофреймворком, який надає лише базові інструменти (маршрутизація, шаблони, сесії), дозволяючи розробнику контролювати структуру проєкту без нав’язування жорсткої архітектури, як у Django з його "батарейками включено". Це зменшує обсяг коду та час навчання, що важливо для обмеженого терміну виконання дипломної роботи. Flask добре підходить для REST-подібних додатків і легко розширюється розширеннями (наприклад, Flask-Bootstrap для інтерфейсу чи Flask-Login для автентифікації). FastAPI, хоча й швидший для асинхронних операцій, є надмірним для синхронної логіки платформи, де основне навантаження – виконання SQL-запитів.

Такий вибір забезпечує баланс між простотою реалізації, продуктивністю та можливістю демонстрації ключових аспектів веб-розробки в рамках роботи. Код на Flask є зрозумілим для рецензентів, а Python дозволяє легко інтегрувати майбутні розширення, такі як API для Google Classroom.

2.2.2. База даних платформи

Як основну базу даних для зберігання метаданих платформи (користувачі, завдання, набори завдань, спроби студентів тощо) обрано SQLite. SQLite є легкою, файл-орієнтованою реляційною системою управління базами даних, яка не вимагає

окремого серверного процесу та зберігає всю базу в одному файлі (у проєкті – platform.db). Це робить її ідеальним вибором для веб-додатків середньої складності, особливо в освітніх проєктах, де не передбачено високе навантаження чи одночасний доступ тисяч користувачів.

Обґрунтування вибору SQLite базується на її простоті інтеграції з Python через вбудований модуль sqlite3, який не потребує додаткових драйверів чи конфігурації. База даних платформи містить таблиці для автентифікації, керування завантаженими файлами, завданнями та статистикою, що не вимагає складних транзакцій чи реплікації. SQLite забезпечує повну підтримку SQL-стандарту, ACID-транзакцій та достатню продуктивність для операцій читання/запису в межах одного додатка. Альтернативи, такі як PostgreSQL чи MySQL, були відхилені через необхідність налаштування окремого сервера, що ускладнило б розгортання та демонстрацію проєкту.

Додатково, для користувацьких баз даних, які завантажують викладачі для завдань, також використано формат SQLite. Це дозволяє легко зберігати та підключатися до файлів у read-only режимі, забезпечуючи безпеку (запобігання модифікаціям чи ін’єкціям). Виконання студентських запитів відбувається в ізольованому з’єднанні з обмеженнями (тільки SELECT), що мінімізує ризики. Такий підхід гарантує портативність, простоту резервного копіювання та сумісність з вимогами до навчальної платформи, де акцент на практичній роботі з файлами баз даних без серверної інфраструктури. SQLite ідеально відповідає масштабам проєкту та спрощує подальше розгортання, наприклад, за допомогою Docker для демонстрації.

2.2.3. Робота з користувацькими базами даних

Для роботи з користувацькими базами даних, які завантажують викладачі для створення завдань, обрано виключно формат SQLite. Кожна база даних зберігається як окремий файл у директорії media/databases/ з унікальним ідентифікатором, що забезпечує просте завантаження, зберігання та доступ. Викладач завантажує готовий SQLite-файл (наприклад, дамп бази даних), вказує назву та опис, після чого

файл стає доступним для прив’язки до завдань. Під час створення задачі викладач вказує еталонний SELECT-запит, який виконується на цій базі для автоматичного збереження очікуваного результату у форматі JSON.

Обґрунтуванням вибору SQLite для користувацьких баз є його файлова природа, яка не вимагає серверного процесу та спрощує обробку файлів у Flask за допомогою стандартних інструментів (werkzeug.utils.secure_filename та uuid для унікальності). Це дозволяє легко реалізувати read-only режим підключення (uri=True з параметром mode=ro), що запобігає будь-яким модифікаціям бази даних студентами. Виконання запитів відбувається в ізольованому з’єднанні з обмеженнями: дозволено тільки SELECT-запити, встановлено таймаут та обробку помилок, що мінімізує ризики SQL-ін’єкцій чи нескінченних циклів.

Альтернативи, такі як підтримка MySQL, PostgreSQL чи інших СУБД, були відхилені через значне ускладнення системи: потреба в sandbox-оточенні (наприклад, Docker-контейнери для кожного запиту), різні драйвери та потенційні вразливості безпеки. Багатоваріантність форматів вимагала б додаткової валідації та конвертації, що вийшло б за рамки обсягу роботи та збільшило б ризики. SQLite забезпечує повну сумісність з навчальними цілями, оскільки є легким, портативним і широко використовуваним для прототипів баз даних, дозволяючи викладачам створювати прості схеми (наприклад, chinook.db чи власні таблиці співробітників).

Механізм роботи реалізовано через окремий сервіс query_executor.py, який приймає ім’я файлу бази та запит, підключається в read-only режимі та повертає результати як список словників для порівняння. Це гарантує безпеку, ефективність та простоту масштабування в межах проєкту, роблячи платформу зручною для використання в освітньому середовищі без додаткової інфраструктури.

2.2.4. Фронтенд-компоненти

Для фронтенд-частини платформи обрано шаблонізатор Jinja2 у поєднанні з бібліотекою Bootstrap 5. Jinja2 є вбудованим шаблонізатором Flask, що забезпечує динамічне генерування HTML-сторінок з можливістю вставки даних з backend, умовних конструкцій, циклів та спадкування шаблонів. Це дозволяє створювати

чисту, структуровану розмітку без необхідності в окремому фронтенд-фреймворку, зберігаючи простоту та узгодженість з backend-логікою. Jinja2 підтримує розширення, такі як макроси та фільтри, що спрощує повторне використання компонентів інтерфейсу, наприклад, для відображення таблиць завдань чи форм.

Обґрунтуванням вибору Jinja2 стало його тісна інтеграція з Flask, що мінімізує залежності та час розробки. Альтернативи, такі як React чи Vue.js для створення односторінкового додатка (SPA), були відхилені через надмірну складність для бакалаврської роботи: вони вимагають окремого бандлера (наприклад, Vite чи Webpack), управління станом та API-ендпоінтів, що збільшило б обсяг проєкту та відволікло від основної логіки (SQL-перевірка та генерація завдань). Jinja2 забезпечує серверний рендеринг, що покращує SEO та швидкість завантаження для навчальної платформи з відносно простим інтерфейсом.

Bootstrap 5 обрано для стилізації та адаптивності інтерфейсу. Ця бібліотека надає готові компоненти (форми, таблиці, модальні вікна, навігацію, прогрес-бар), що дозволяють швидко створити сучасний, респонсивний дизайн без написання власного CSS. Інтеграція здійснюється через розширення Flask–Bootstrap, яке спрощує підключення та використання. Bootstrap забезпечує крос-браузерну сумісність та мобільну адаптивність, що важливо для доступу студентів з різних пристроїв. Відмова від складніших фреймворків, таких як Tailwind CSS чи Material UI, обґрунтована бажанням зберегти легкість проєкту та уникнути додаткових залежностей.

Такий підхід до фронтенду робить платформу простою у підтримці, швидкою в розробці та орієнтованою на функціональність, а не на візуальну складність. Інтерфейс залишається інтуїтивним для викладачів та студентів, з акцентом на зручність роботи з формами та таблицями, що відповідає освітнім цілям системи.

2.2.5. Інші бібліотеки та інструменти

Окрім основного фреймворку та баз даних, для реалізації платформи використано низку додаткових бібліотек та інструментів, які забезпечують безпеку, зручність роботи з файлами, генерацію унікальних ідентифікаторів та інші

допоміжні функції. Ці компоненти обрано з урахуванням мінімалізму Flask, щоб розширити його можливості без ускладнення архітектури. Вони є стандартними для Python-розробки веб-додатків і не вимагають значних ресурсів, що відповідає масштабам проєкту.

Перелік основних додаткових бібліотек та інструментів:

- Werkzeug: використовується для хешування паролів (generate_password_hash та check_password_hash) та безпечної обробки завантажених файлів (secure_filename), що забезпечує захист від типових вразливостей.
- UUID: стандартний модуль Python для генерації унікальних ідентифікаторів файлів баз даних та share_code для індивідуальних наборів завдань, гарантуючи відсутність колізій.
- Random: модуль для рандомізації вибору завдань під час генерації індивідуальних наборів, що запобігає однаковості та списуванню.
- JSON: вбудований модуль для серіалізації та десеріалізації очікуваних результатів запитів, зберігання критеріїв генерації (topics, difficulties) у JSON-форматі.
- OS: для роботи з файловою системою (створення директорій, перевірка шляхів до баз даних), що спрощує керування завантаженими файлами.

Для майбутньої інтеграції з Google Classroom планується використання бібліотек google-api-python-client та google-auth-oauthlib, які дозволять реалізувати OAuth 2.0 автентифікацію та доступ до API Classroom.

Використання цих бібліотек та інструментів забезпечує повноту функціоналу без надмірних залежностей, роблячи код чистим, безпечним та легким для підтримки. Вони ідеально доповнюють базовий стек, дозволяючи зосередитися на ключовій логіці платформи , а саме – перевірці SQL-запитів та генерації завдань, і сприяють швидкій розробці в рамках обмеженого обсягу бакалаврської роботи.

2.3. Розробка архітектури системи

2.3.1. Загальна архітектура

Розроблена платформа має клієнт-серверну архітектуру з серверним рендерингом сторінок. Клієнтська частина представлена веб-браузером користувача, який надсилає HTTP-запити до сервера. Сервер на базі Flask обробляє запити, виконує бізнес-логіку, взаємодіє з базами даних та генерує HTML-сторінки за допомогою шаблонізатора Jinja2. Така архітектура забезпечує простоту розгортання, високу продуктивність для навчального навантаження та повний контроль над виконанням SQL-запитів у контрольованому середовищі.

Основні компоненти системи:

- Клієнтський рівень (веб-браузер користувача).
- Рівень представлення (шаблони Jinja2 з Bootstrap для інтерфейсу).
- Рівень додатка (Flask з маршрутами, сервісами та бізнес-логікою).
- Рівень даних (SQLite для метаданих платформи та файлові SQLite-бази для завдань).

Архітектура побудована за принципом поділу відповідальностей: маршрути обробляють запити, сервіси містять бізнес-логіку (перевірка запитів, генерація наборів), моделі даних забезпечують доступ до бази, а шаблони відповідають за відображення. Це дозволяє легко підтримувати та розширювати систему, наприклад, додавати нові модулі без зміни існуючого коду.

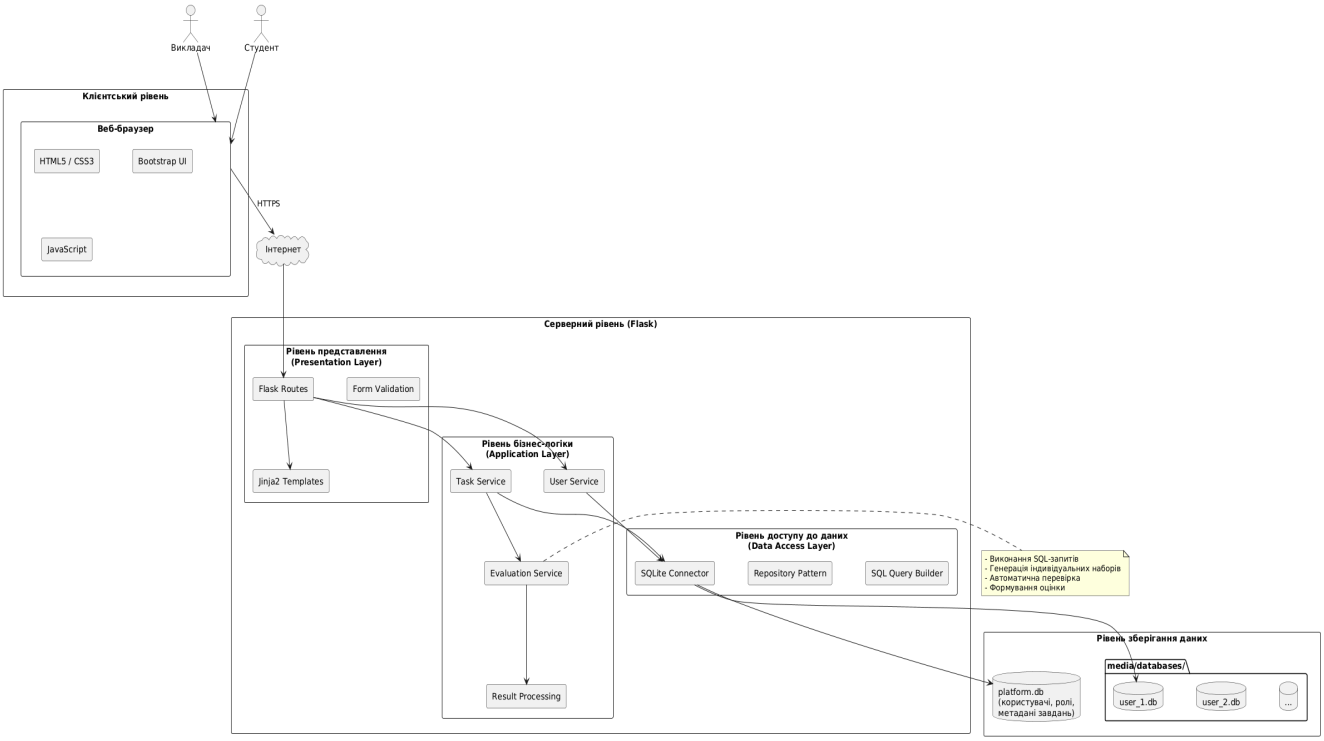

Рис. 2.1 Загальна архітектура системи

Запропонована архітектура є оптимальною для навчальної платформи з обмеженим навантаженням: вона не вимагає додаткових серверів баз даних чи кешування, забезпечує високу безпеку при роботі з користувацькими запитами та легко розгортається на одному хості. У майбутньому можливе розширення до мікросервісної архітектури або асинхронної обробки для більшого масштабу.

2.3.2. Структура проєкту та модулі

Структура проєкту побудована за принципом чіткого поділу відповідальностей, що забезпечує легкість підтримки, тестування та подальшого розширення. Усі компоненти організовані в логічні директорії та файли, з акцентом на модульність: основна логіка розміщена в app.py, конфігурація та допоміжні функції винесені в окремі модулі, а шаблони та статичні файли – у відповідні папки. Такий підхід відповідає рекомендованим практикам розробки Flask-додатків і дозволяє легко орієнтуватися в коді.

Основні директорії та файли проєкту:

- app.py – головний файл додатка, містить ініціалізацію Flask, маршрути (routes), представлення (views) та обробку запитів для всіх ролей користувачів.
- config.py – файл конфігурації, визначає секретний ключ, шляхи до баз даних та директорій для завантажень, а також функцію перевірки дозволених розширень файлів.
- database.py – модуль для ініціалізації бази даних платформи (створення таблиць) та функції отримання з'єднання з SQLite.
- services/ – директорія з сервісами бізнес –логіки:
  - query_executor.py – безпечне виконання SQL –запитів у read –only режимі та порівняння результатів.
  - assignment_generator.py – генерація індивідуальних наборів завдань з рандомізацією для зареєстрованих студентів.
- templates/ – директорія з Jinja2 –шаблонами:
  - base.html – базовий шаблон з навігацією.
  - teacher/ – шаблони для викладача (дашборд, керування базами, завданнями, наборами).
  - student/ – шаблони для студента (дашборд, виконання завдань).
- static/ – статичні файли (CSS, JavaScript, зображення), включаючи кастомні стилі на основі Bootstrap.
- media/databases/ – директорія для зберігання завантажених викладачами SQLite –файлів баз даних.
- platform.db – основна база даних платформи з метаданими (користувачі, завдання, набори, спроби).
- requirements.txt – список залежностей для розгортання.

Така структура забезпечує чіткий поділ на рівні: представлення (шаблонів), логіку додатка (маршрути та сервіси) та дані (SQLite –файли). Модульність дозволяє незалежно розвивати окремі компоненти, наприклад, додавати нові сервіси без зміни основного app.py. Проєкт легко розгортається на локальному

сервері або в Docker, а код залишається зрозумілим для аналізу та демонстрації в рамках бакалаврської роботи.

2.3.3. Схема взаємодії компонентів

Взаємодія компонентів системи побудована за принципом послідовної обробки запитів від користувача до отримання відповіді, з чітким розподілом обов'язків між рівнями. Користувач (викладач або студент) через веб –браузер надсилає HTTP –запит до Flask –додатка. Маршрут (view) обробляє запит, викликає необхідний сервіс бізнес –логіки, який, у свою чергу, взаємодіє з рівнем даних (SQLite –бази або файлові бази). Після отримання результатів сервіс повертає дані маршруту, який формує відповідь у вигляді HTML –шаблону Jinja2. Така схема забезпечує контрольоване виконання критичних операцій, таких як перевірка SQL –запитів чи генерація наборів, у ізольованому середовищі.

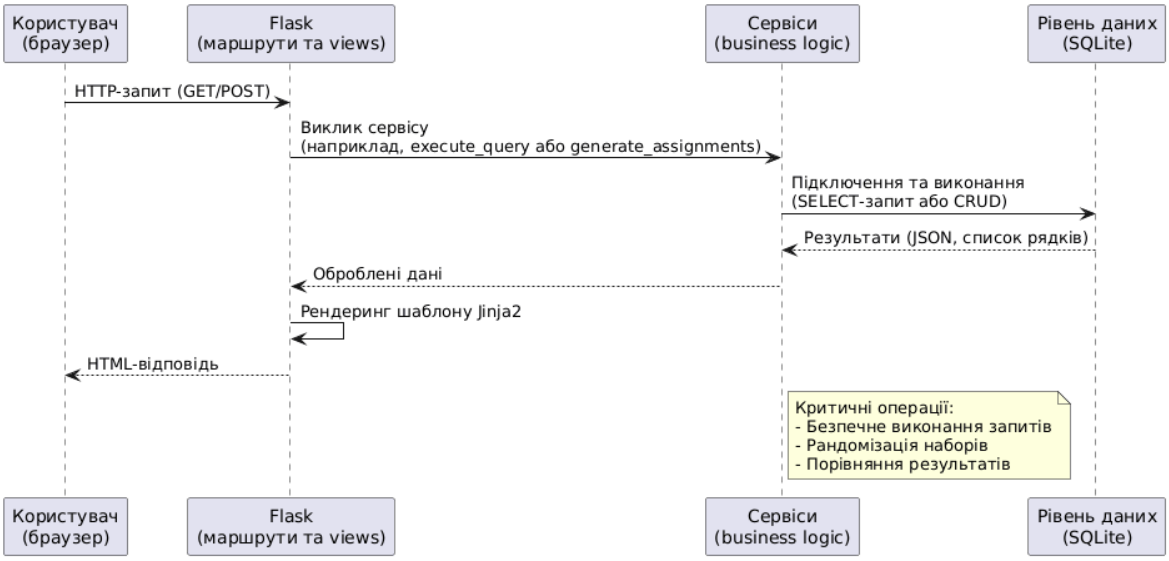

Рис. 2.2 Схема взаємодії компонентів

Наведена схема ілюструє типовий потік для ключових сценаріїв: створення завдання викладачем, генерації наборів або виконання запиту студентом. У разі POST –запитів (наприклад, надсилання SQL студента) додається етап валідації та збереження спроби в базі. Взаємодія є синхронною, що спрощує відладку та забезпечує послідовність операцій. Така організація мінімізує залежності між компонентами, полегшує тестування (наприклад, сервіси можна тестувати окремо) та відповідає вимогам безпеки при роботі з користувацькими запитами.

2.4. Проектування моделі даних

2.4.1. ER-діаграма

Модель даних платформи спроектована з урахуванням реляційного підходу та вимог до зберігання метаданих про користувачів, бази даних, завдання, набори завдань та спроби студентів. Основними сутностями є користувачі (з ролями викладач/студент), завантажені бази даних, завдання, набори завдань, індивідуальні набори для студентів, зв’язки між завданнями та наборами, а також спроби виконання. Зв’язки між сутностями відображають ієрархію: викладач завантажує бази та створює завдання і набори, студенти отримують індивідуальні набори та зберігають спроби.

ER-діаграма відображає ключові зв’язки один-до-багатьох (викладач створює багато баз, завдань та наборів; один набір прив’язаний до однієї бази; студент має багато індивідуальних наборів та спроб). Багато зв’язок між індивідуальними наборами та завданнями реалізовано через проміжну таблицю student_assignment_tasks. Така структура забезпечує нормалізацію даних, уникнення дублювання та ефективне виконання запитів для статистики прогресу студентів. Модель легко розширюється для майбутньої інтеграції з Google Classroom (додавання полів для course_id тощо).

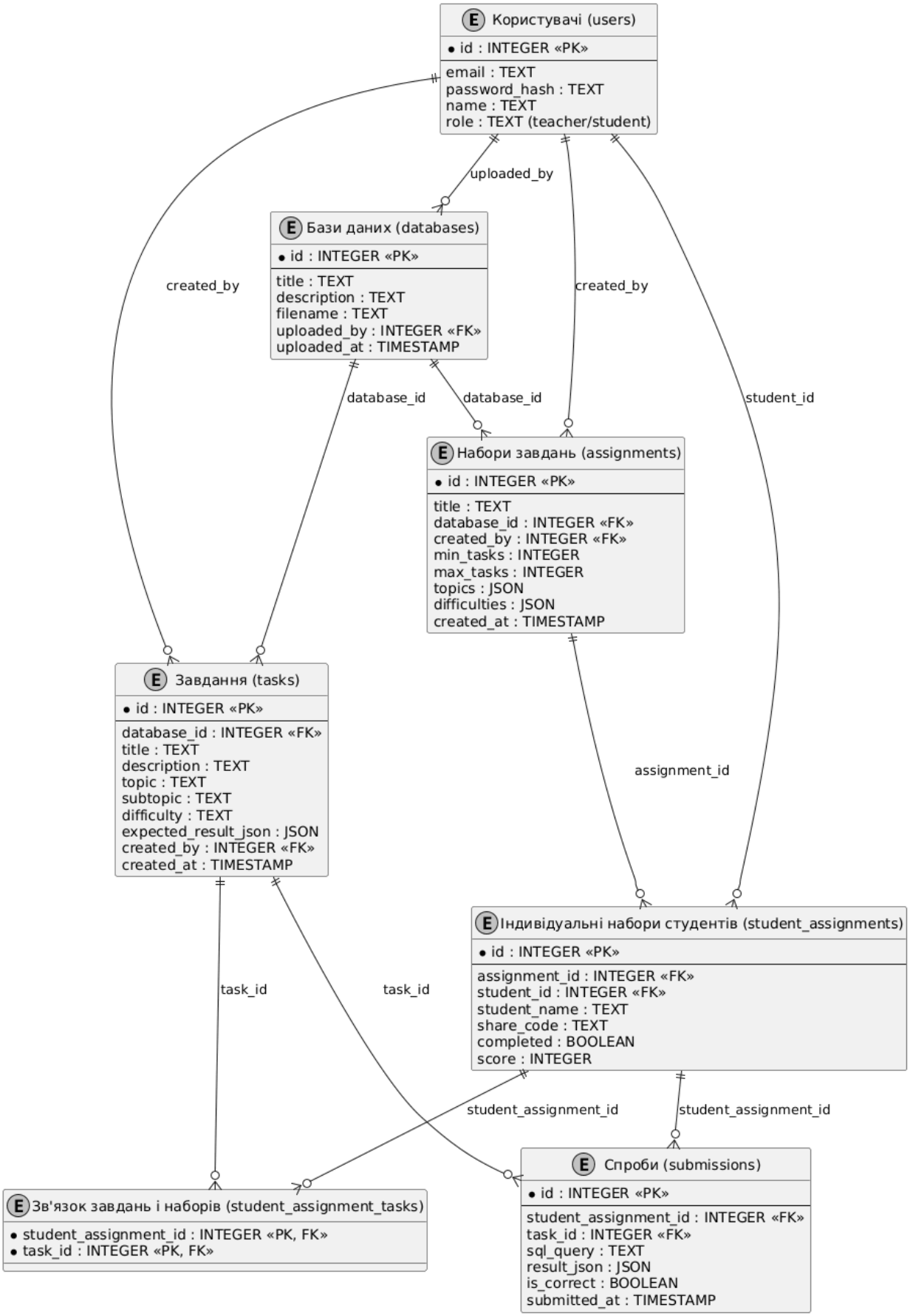

Рис. 2.3 ER-діаграма моделі даних

2.4.2. Опис таблиць та зв'язків

Модель даних платформи реалізовано у реляційній базі даних SQLite з використанням семи основних таблиць, які зберігають всю необхідну інформацію для функціонування системи. Кожна таблиця має чітко визначену структуру з первинними та зовнішніми ключами для забезпечення цілісності даних, а також поля типу JSON для гнучкого зберігання структурованих даних, таких як очікувані результати чи критерії фільтрації. Опис таблиць подано нижче з детальним поясненням їх ролі в системі.

Таблиця users є центральною для автентифікації та управління ролями. Вона зберігає дані всіх зареєстрованих користувачів платформи, розділяючи їх на викладачів та студентів. Поле role визначає права доступу: викладачі можуть створювати контент, а студенти – лише виконувати завдання. Хешування паролів забезпечує безпеку, а унікальність email запобігає дублюванню акаунтів. Ця таблиця є основою для всіх зовнішніх ключів, що пов'язують користувачів з створеним чи призначеним контентом.

Таблиця 2.3

Користувачі (users)

Поле	Тип даних	Опис	Обмеження
id	INTEGER	Унікальний ідентифікатор користувача	PRIMARY KEY, AUTOINCREMENT
email	TEXT	Електронна пошта (унікальна)	UNIQUE, NOT NULL
password_hash	TEXT	Хеш пароля	NOT NULL
name	TEXT	ПІБ користувача	NOT NULL
role	TEXT	Роль (teacher або student)	NOT NULL

Таблиця databases призначена для зберігання метаданих про завантажені викладачами SQLite –файли баз даних. Кожен запис відповідає одному файлу, який використовується як основа для завдань. Поле filename вказує на фізичний файл у директорії media/databases, а зовнішній ключ uploaded_by пов'язує базу з

викладачем. Це дозволяє викладачам мати кілька незалежних баз для різних курсів чи тем, забезпечуючи гнучкість у створенні завдань.

Таблиця 2.4

Завантажені бази даних (databases)

Поле	Тип даних	Опис	Обмеження
id	INTEGER	Унікальний ідентифікатор	PRIMARY KEY, AUTOINCREMENT
title	TEXT	Назва бази даних	NOT NULL
description	TEXT	Опис (опціонально)	
filename	TEXT	Ім’я файлу в директорії media/databases	NOT NULL
uploaded_by	INTEGER	ID викладача	FOREIGN KEY (users.id)
uploaded_at	TIMESTAMP	Дата завантаження	DEFAULT CURRENT_TIMESTAMP

Таблиця tasks містить усі створені викладачем завдання з SQL. Кожне завдання прив’язане до конкретної бази даних через database_id та містить повний опис: умова, тематика, рівень складності та еталонний результат у JSON –форматі для автоматичної перевірки. Поле expected_result_json генерується автоматично під час створення завдання шляхом виконання еталонного запиту. Це забезпечує точну та об’єктивну перевірку відповідей студентів.

Таблиця 2.5

Завдання (tasks)

Поле	Тип даних	Опис	Обмеження
id	INTEGER	Унікальний ідентифікатор	PRIMARY KEY, AUTOINCREMENT
database_id	INTEGER	ID бази даних	FOREIGN KEY (databases.id)
title	TEXT	Назва завдання	NOT NULL
description	TEXT	Умова завдання	NOT NULL
topic	TEXT	Тематика	NOT NULL
subtopic	TEXT	Підтема (опціонально)	
difficulty	TEXT	Рівень складності	

Продовження таблиці 2.5

expected_result_json	JSON	Очікуваний результат у JSON	NOT NULL
created_by	INTEGER	ID викладача	FOREIGN KEY (users.id)
created_at	TIMESTAMP	Дата створення	DEFAULT CURRENT_TIMESTAMP

Таблиця assignments описує набори завдань, створені викладачем для розподілу серед студентів. Вона містить критерії генерації (мінімальна та максимальна кількість завдань, теми та рівні складності у JSON), що дозволяють автоматизувати створення індивідуальних варіантів. Набір завжди прив’язаний до однієї бази даних, забезпечуючи єдність датасету для всіх завдань у наборі.

Таблиця 2.6 Набори завдань (assignments)

Поле	Тип даних	Опис	Обмеження
id	INTEGER	Унікальний ідентифікатор	PRIMARY KEY, AUTOINCREMENT
title	TEXT	Назва набору	NOT NULL
database_id	INTEGER	ID бази даних	FOREIGN KEY (databases.id)
created_by	INTEGER	ID викладача	FOREIGN KEY (users.id)
min_tasks	INTEGER	Мінімальна кількість завдань	DEFAULT 5
max_tasks	INTEGER	Максимальна кількість завдань	DEFAULT 10
topics	JSON	Список тем (JSON)	
difficulties	JSON	Список рівнів складності (JSON)	
created_at	TIMESTAMP	Дата створення	DEFAULT CURRENT_TIMESTAMP

Таблиця student_assignments зберігає згенеровані індивідуальні набори для кожного студента. Кожен запис містить унікальний share_code для доступу, посилання на студента та базовий набір. Поля completed та score дозволяють відстежувати прогрес та оцінку, хоча основний розрахунок відбувається динамічно на основі спроб.

Таблиця 2.7

Індивідуальні набори студентів (student_assignments)

Поле	Тип даних	Опис	Обмеження
id	INTEGER	Унікальний ідентифікатор	PRIMARY KEY, AUTOINCREMENT
assignment_id	INTEGER	ID набору завдань	FOREIGN KEY (assignments.id)
student_id	INTEGER	ID студента	FOREIGN KEY (users.id)

Продовження таблиці 2.7.

student_name	TEXT	Ім’я студента	NOT NULL
share_code	TEXT	Унікальний код доступу	UNIQUE
completed	BOOLEAN	Прапорець завершення	DEFAULT 0
score	INTEGER	Оцінка (опціонально)	DEFAULT 0

Таблиця student_assignment_tasks є проміжною для реалізації багатобагатозв’язку між індивідуальними наборами студентів та завданнями. Вона містить лише ключі, що дозволяють гнучко формувати різні комбінації завдань для кожного студента під час генерації.

Таблиця 2.8

Зв’язок завдань і індивідуальних наборів (student_assignment_tasks)

Поле	Тип даних	Опис	Обмеження
student_assignment_id	INTEGER	ID індивідуального набору	PRIMARY KEY, FOREIGN KEY
task_id	INTEGER	ID завдання	PRIMARY KEY, FOREIGN KEY

Таблиця submissions фіксує всі спроби виконання завдань студентами. Кожен запис містить надісланий запит, результат виконання у JSON, прапорець правильності та час надсилання. Це дозволяє зберігати історію спроб, відстежувати прогрес та аналізувати помилки.

Таблиця 2.9

Спроби виконання (submissions)

Поле	Тип даних	Опис	Обмеження
id	INTEGER	Унікальний ідентифікатор	PRIMARY KEY, AUTOINCREMENT
student_assignment_id	INTEGER	ID індивідуального набору	FOREIGN KEY
task_id	INTEGER	ID завдання	FOREIGN KEY
sql_query	TEXT	SQL –запит студента	NOT NULL
result_json	JSON	Результат виконання	
is_correct	BOOLEAN	Прапорець правильності	DEFAULT 0
submitted_at	TIMESTAMP	Дата надсилання	DEFAULT CURRENT_TIMESTAMP

Таблиця 2.10

Опис зв’язків між таблицями

Зв’язок	Тип зв’язку	Опис
users → databases	Один –до – багатьох	Один викладач завантажує багато баз даних
users → tasks	Один –до – багатьох	Один викладач створює багато завдань
users → assignments	Один –до – багатьох	Один викладач створює багато наборів завдань
users → student_assignments	Один –до – багатьох	Один студент має багато індивідуальних наборів
databases → tasks	Один –до – багатьох	Одна база даних використовується в багатьох завданнях
databases → assignments	Один –до – багатьох	Одна база даних прив’язана до багатьох наборів
assignments → student_assignments	Один –до – багатьох	Один набір генерує багато індивідуальних наборів для студентів
student_assignments ↔ tasks	Багато –до – багатьох	Через проміжну таблицю student_assignment_tasks
student_assignments → submissions	Один –до – багатьох	Один індивідуальний набір має багато спроб
tasks → submissions	Один –до – багатьох	Одне завдання має багато спроб від різних студентів

Описана модель даних забезпечує повну нормалізацію, ефективне зберігання та швидкий доступ до інформації для всіх функцій системи: від генерації наборів до розрахунку статистики прогресу. Вона гнучка для розширення (наприклад, додавання полів для інтеграції з Google Classroom) та відповідає принципам реляційного дизайну.

2.5. Проектування інтерфейсу користувача

2.5.1. Інтерфейс викладача

Інтерфейс викладача спроектовано з акцентом на зручність керування контентом та моніторингу прогресу студентів. Основна навігація реалізована через верхню панель з логотипом платформи, привітанням користувача, вказівкою ролі та кнопкою виходу. Всі сторінки використовують єдиний базовий шаблон з Bootstrap 5 для забезпечення адаптивності та сучасного вигляду. Колірна схема – світла з акцентами синього для кнопок дій, що полегшує сприйняття. Форми створення та редагування об’єктів містять обов’язкові поля з валідацією та підказками, а таблиці підтримують сортування та пагінацію для великих обсягів даних.

Головною сторінкою викладача являється дашборд , що надає швидкий доступ до ключових функцій: перехід до керування базами даних, створення наборів завдань та перегляду статистики. На дашборді відображається список завантажених баз даних з короткою інформацією та швидкими діями (додати завдання, переглянути задачі, створити набір).

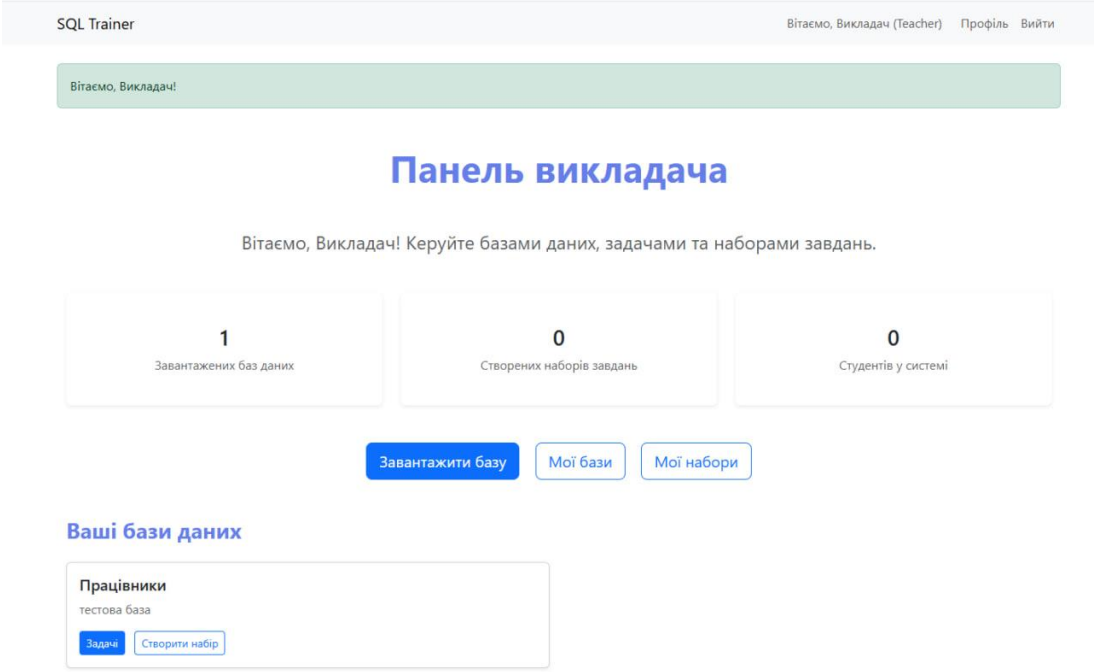

Рис. 2.4 Дашборд викладача

Сторінка керування базами даних дозволяє завантажувати нові SQLite – файли, переглядати список існуючих з описом та переходити до створення завдань для кожної бази. Форма завантаження містить поля для назви, опису та вибору файлу з перевіркою розширення. Таблиця баз даних включає колонки для назви, опису, дати завантаження та дій.

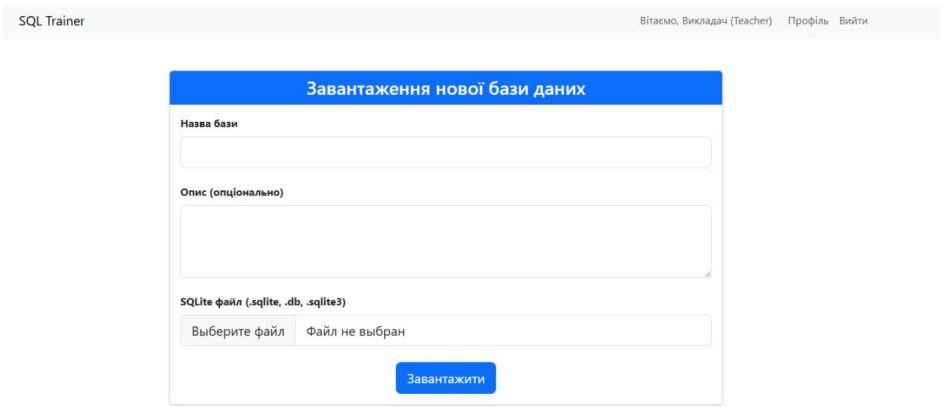

Рис. 2.5 Сторінка керування базами даних

Створення та редагування завдань відбувається на окремих сторінках з формою, що включає поля для назви, умови, тематики, підтеми, рівня складності та еталонного запиту. Після збереження система автоматично виконує еталонний

запит та зберігає результат. Список завдань для бази даних відображається у таблиці з можливістю редагування та видалення.

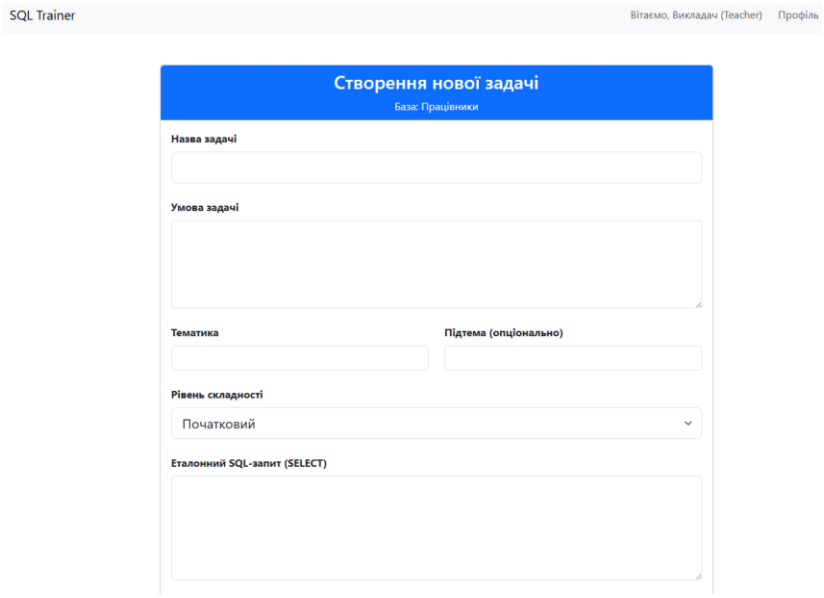

Рис. 2.6 Форма створення завдання

Сторінка створення набору завдань містить форму з назвою, критеріями кількості завдань та чекбоксами для вибору тем і рівнів складності. Після створення генеруються індивідуальні набори для всіх студентів, а викладач отримує перехід до перегляду кодів доступу та статистики.

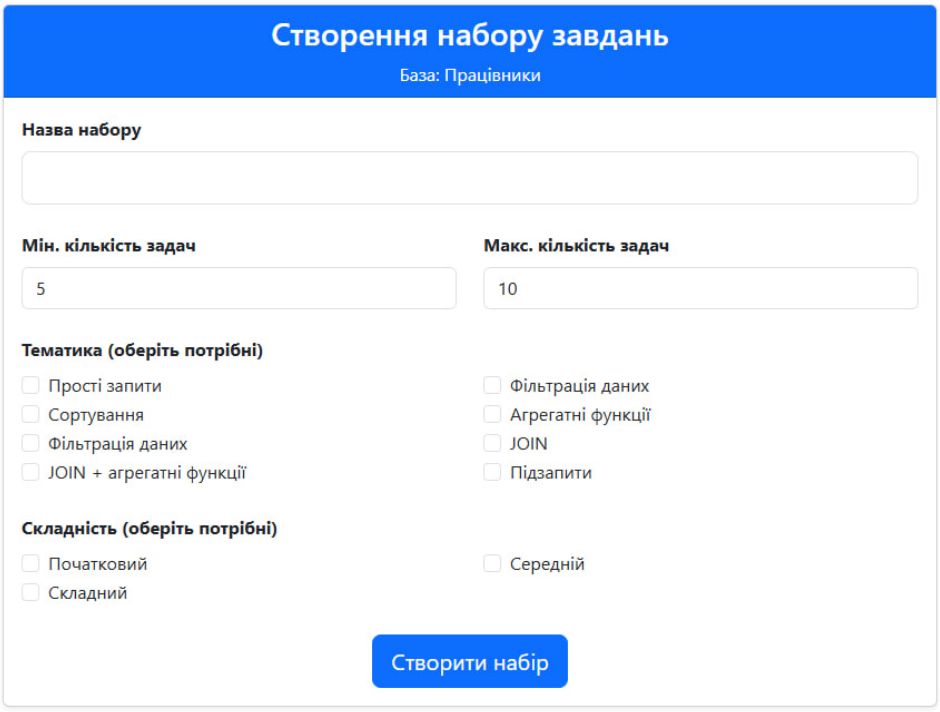

Рис. 2.7 Форма створення набору завдань

Перегляд результатів студентів реалізовано у таблиці з прогрес –барами, відсотком успішності та кількістю правильних відповідей. Кожний рядок містить ім’я студента, код доступу та посилання для перегляду детального виконання.

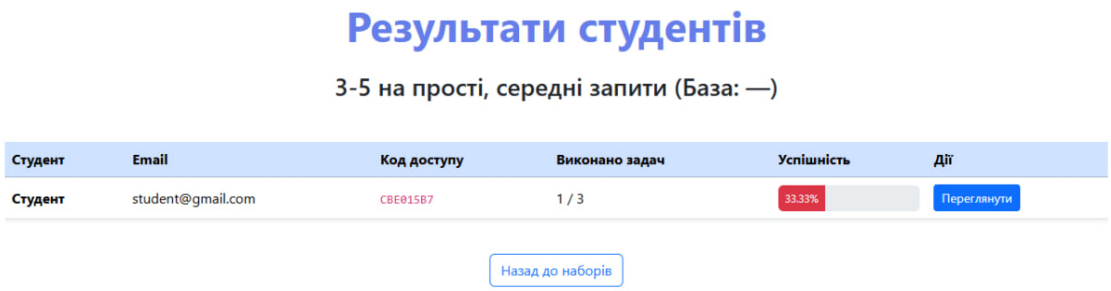

Рис. 2.8 Сторінка перегляду результатів студентів по набору

Інтерфейс викладача забезпечує повний контроль над навчальним контентом та студентами, з інтуїтивними формами та таблицями, що відповідає принципам usability та дозволяє ефективно керувати великими групами.

2.5.2. Інтерфейс студента

Інтерфейс студента спроектовано з акцентом на простоту та фокус на виконанні завдань, без зайвих елементів керування. Навігація аналогічна викладацькій: верхня панель з логотипом, привітанням, роллю та кнопкою виходу. Дизайн базується на Bootstrap 5 для адаптивності, з великими полями для введення SQL –запитів, чіткими індикаторами прогресу та кольоровим кодуванням результатів (зелений для правильних, червоний для неправильних). Форми надсилання запитів мінімалістичні, з автоматичним збереженням історії спроб та відображенням результатів у табличному форматі для зручного порівняння.

Для студента є окрема головна сторінка, подібна до дашборду, яка відображає список усіх призначених наборів завдань з назвою набору, базою даних та кнопкою переходу до виконання. Якщо наборів немає, показується відповідне повідомлення. Це дозволяє студентові швидко орієнтуватися в доступних завданнях та переходити до роботи без додаткових дій.

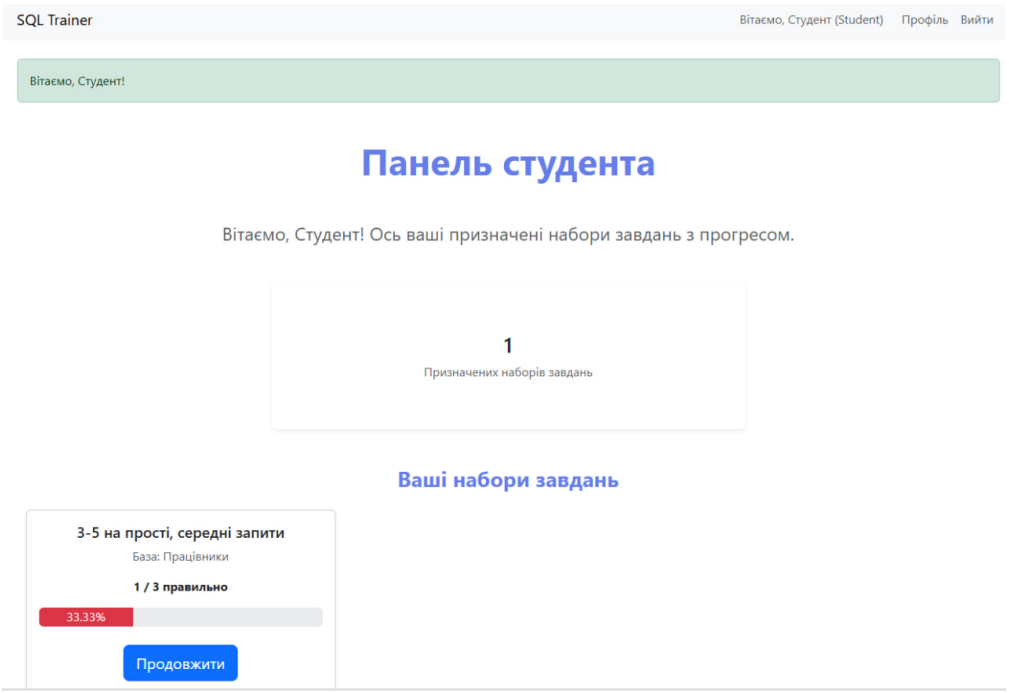

Рис. 2.9 Дашборд студента

Сторінка виконання набору завдань містить заголовок з назвою набору та бази даних, індикатор загального прогресу (кількість правильних завдань з загальної та прогрес –бар). Завдання відображаються у вигляді карток з номером, назвою, тематикою, рівнем складності та повною умовою. Під умовою розташована форма з великим текстовим полем для SQL –запиту та кнопкою перевірки.

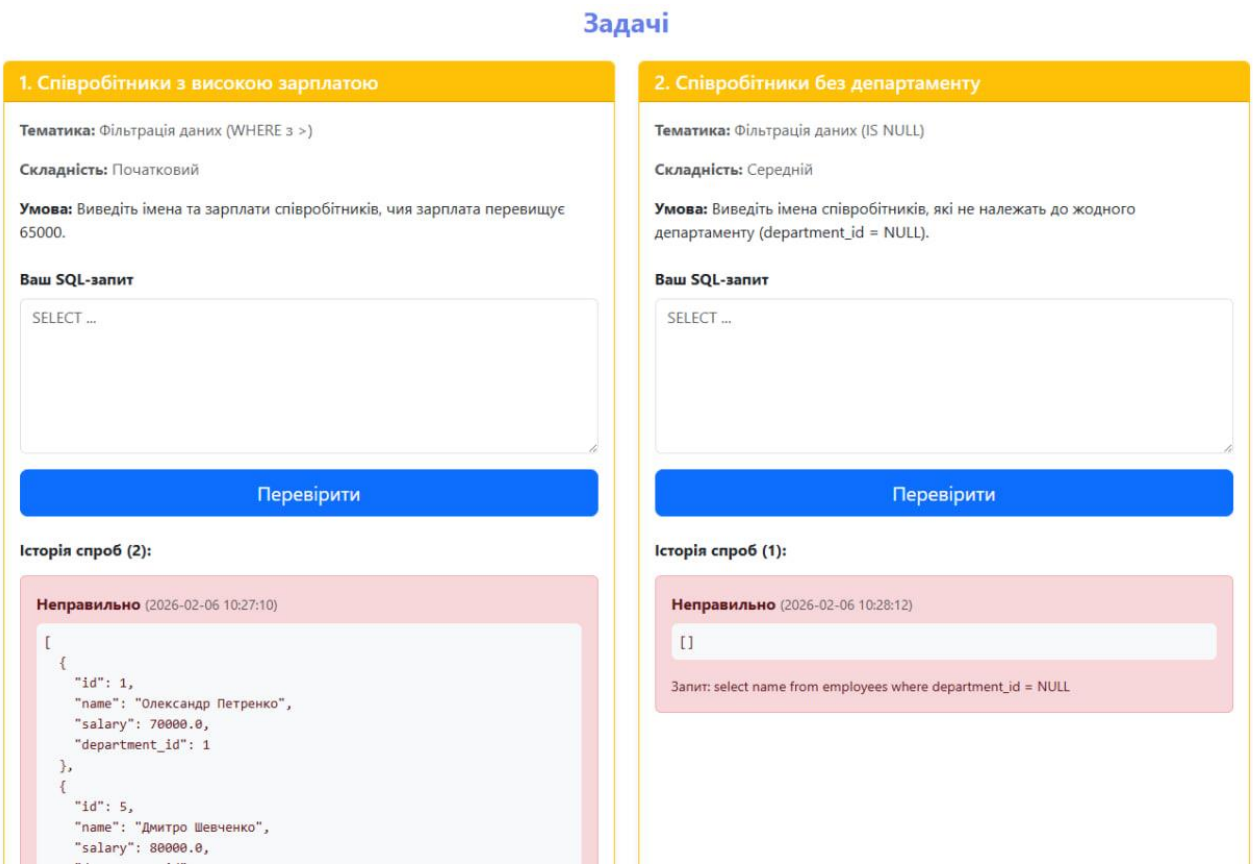

Рис. 2.10 Сторінка виконання набору завдань

Після надсилання запиту система відображає результат: повідомлення про правильність, таблицю з виведенням запиту та історію всіх спроб для цього завдання (з датою, запитом та результатом). Це дозволяє студентові аналізувати свої помилки та покращувати запити без перезавантаження сторінки.

Рис. 2.11 Форма виконання завдання з історією спроб та результатом

Інтерфейс студента забезпечує інтуїтивність, мотивацію через візуальний прогрес та фокус на практичній роботі з SQL. Він мінімалістичний, адаптивний до мобільних пристроїв та сприяє самостійному навчанню з миттєвим зворотним зв’язком.

2.6. Проектування механізмів динамічної генерації завдань та автоматичної перевірки SQL –запитів

Механізми динамічної генерації завдань та автоматичної перевірки SQL – запитів є ключовими елементами платформи, що відрізняють її від існуючих аналогів. Динамічна генерація забезпечує індивідуалізацію наборів для кожного студента, запобігаючи списуванню та адаптуючи складність до критеріїв викладача. Автоматична перевірка базується на точному порівнянні результатів

виконання запитів, з урахуванням безпеки та нормалізації даних. Проектування цих механізмів виконано з акцентом на модульність, безпеку та ефективність, використовуючи сервісний рівень для ізоляції логіки від представлення.

Процес динамічної генерації завдань реалізовано в сервісі assignment_generator.py і включає такі етапи:

- Отримання критеріїв з набору завдань (мінімальна/максимальна кількість, теми, рівні складності).
- Фільтрація доступних завдань за заданими критеріями з пулу, прив’язаного до однієї бази даних.
- Вибір випадкової кількості завдань у заданому діапазоні для кожного зареєстрованого студента.
- Рандомний відбір завдань з відфільтрованого списку без повторів.
- Створення індивідуального запису в таблиці student_assignments з унікальним share_code та збереженням зв’язків у проміжній таблиці student_assignment_tasks.

Такий підхід гарантує унікальність наборів для кожного студента при збереженні тематичної та складнісної узгодженості, що сприяє об’єктивній оцінці знань.

Механізм автоматичної перевірки SQL-запитів реалізовано в сервісі query_executor.py з жорсткими обмеженнями безпеки. Перевірка відбувається шляхом порівняння результатів виконання запиту студента з еталонним результатом, збереженим у JSON під час створення завдання. Процес включає такі кроки:

- Перевірку синтаксису запиту (дозволено тільки SELECT).
- Підключення до користувацької бази даних у read-only режимі (file:...?mode=ro).
- Виконання запиту з обробкою помилок та таймаутом.
- Конвертацію результату в список словників (нормалізація порядку рядків та стовпців через сортування ключів).

– Точне порівняння з еталонним JSON.

– Збереження спроби з прапорцем правильності та результатом.

Це забезпечує об’єктивну перевірку без ручного втручання, з миттєвим зворотним зв’язком для студента та накопиченням історії для аналізу викладачем. Механізми спроектовані модульно для легкого тестування та розширення, наприклад, додавання нормалізації для агрегатних функцій чи підтримки інших діалектів SQL у майбутньому.

2.7. Аналіз вимог та проектування інтеграції з Google Classroom

Інтеграція з Google Classroom є важливим елементом платформи, оскільки ця система управління навчанням широко використовується в освітніх закладах України та світу для організації курсів, розподілу завдань та оцінки студентів. Аналіз вимог показав, що інтеграція повинна забезпечувати автентифікацію викладача через OAuth 2.0, отримання списку класів та студентів, публікацію завдань у вигляді посилань на індивідуальні набори та автоматичну передачу оцінок назад до журналу Classroom. Це дозволить викладачам працювати в єдиній екосистемі, уникнувши ручного копіювання посилань чи оцінок, та підвищить зручність використання платформи в реальному навчальному процесі.

Ключові етапи та компоненти проектування інтеграції:

1. Автентифікація викладача за допомогою Google OAuth 2.0 з використанням бібліотек google –auth –oauthlib та google –api –python –client для отримання токенів доступу.

2. Отримання списку курсів (classroom.courses.list) та студентів у кожному курсі (classroom.courses.students.list) для автоматичного зіставлення з зареєстрованими студентами платформи.

3. Створення завдань у Classroom (classroom.courses.courseWork.create) з типом "ASSIGNMENT" та вставкою унікального посилання на індивідуальний набір (share_code).

4. Автоматична передача оцінок (classroom.courses.courseWork.studentSubmissions.patch) на основі розрахунку прогресу (відсоток правильних завдань або ручне коригування викладачем).

5. Збереження мапінгу між наборами платформи та coursework_id у Classroom для синхронізації.

Проектування інтеграції передбачає модульний підхід: окремий пакет classroom_integration з функціями автентифікації, API –запитами та обробкою помилок. У поточній реалізації інтеграція підготовлена як заглушка (з плануванням повного впровадження), з використанням тестових credential –ів Google Cloud Console. Це дозволяє в майбутньому легко активувати функціонал без зміни основної логіки платформи. Інтеграція підвищить практичну цінність системи, зробивши її сумісною з існуючими освітніми інструментами та відповідаючи вимогам змішаного навчання.

2.8. Висновки

У другому розділі проведено комплексне проектування інформаційної системи інтерактивної навчальної платформи для практичного освоєння мови SQL. Визначено функціональні та нефункціональні вимоги, які охоплюють автентифікацію, керування базами даних і завданнями, динамічну генерацію індивідуальних наборів, автоматичну перевірку запитів, відстеження прогресу та інтеграцію з Google Classroom. Функціональні вимоги забезпечують повний цикл роботи для ролей викладача та студента, а нефункціональні – безпеку, продуктивність та зручність використання.

Обґрунтовано вибір технологічного стеку: Python з фреймворком Flask для backend, SQLite для бази даних платформи та користувацьких файлів, Jinja2 з Bootstrap 5 для фронтенду, а також допоміжні бібліотеки (Werkzeug, UUID, random, JSON). Такий стек гарантує простоту розробки, безпеку виконання запитів та швидке прототипування в рамках бакалаврської роботи.

Розроблено архітектуру системи (клієнт –серверну з серверним рендерингом), структуру проєкту з чітким поділом на модулі та схему взаємодії

компонентів. Спроектовано модель даних з ER –діаграмою та детальним описом таблиць і зв’язків, що забезпечує нормалізацію та ефективність запитів. Проект інтерфейсу користувача орієнтовано на зручність: для викладача – інструменти керування контентом та статистикою, для студента – просте виконання завдань з візуальним прогресом.

Окремо спроектовано механізми динамічної генерації наборів (з рандомізацією за критеріями) та автоматичної перевірки SQL –запитів (read –only режим, порівняння JSON –результатів). Проаналізовано вимоги та спроектовано інтеграцію з Google Classroom через OAuth 2.0 та API для автоматизації розподілу завдань та оцінок.

Розроблене проектування створює міцну основу для реалізації системи, усуваючи недоліки аналогів та забезпечуючи гнучкість, безпеку й адаптацію до освітніх потреб. Подальша реалізація дозволить перейти до практичного впровадження та тестування платформи.

РОЗДІЛ 3. РОЗДІЛ 3. РЕАЛІЗАЦІЯ СИСТЕМИ

3.1. Реалізація автентифікації та системи ролей користувачів

Реалізація автентифікації та системи ролей користувачів є базовим модулем платформи, що забезпечує безпеку доступу та розрізнення прав користувачів. Система побудована на механізмі сесій Flask з хешуванням паролів за допомогою бібліотеки Werkzeug. Користувачі поділяються на дві ролі: "teacher" (викладач) з повними правами на створення контенту та моніторинг, і "student" (студент) з доступом лише до призначених завдань. Реєстрація та вхід реалізовані через прості форми без зовнішніх провайдерів (наприклад, Google OAuth), що спрощує розгортання, але залишає можливість розширення в майбутньому.

Процес реєстрації дозволяє користувачеві вказати ПІБ, email, пароль та роль. Email є унікальним ідентифікатором, а пароль хешується перед збереженням у базі даних. Після успішної реєстрації користувач перенаправляється на сторінку входу.

Лістинг 3.1 Маршрут реєстрації (фрагмент app.py)

```
@app.route('/register', methods=['GET', 'POST'])
def register():
 if request.method == 'POST':
  email = request.form['email']
  password = request.form['password']
  name = request.form['name']
  role = request.form['role'] # 'teacher' або 'student'

  if role not in ['teacher', 'student']:
   flash('Невірна роль.', 'danger')
   return redirect(url_for('register'))

  conn = get_db_connection()
  existing = conn.execute('SELECT * FROM users WHERE email = ?', (email,)).fetchone()
  if existing:
```

```
            flash('Користувач з такою поштою вже існує.', 'danger')
            conn.close()
            return redirect(url_for('register'))

        password_hash = generate_password_hash(password)
        conn.execute('INSERT INTO users (email, password_hash, name, role) VALUES (?, ?, ?,
?)',
                (email, password_hash, name, role))
        conn.commit()
        conn.close()
        flash('Реєстрація успішна. Увійдіть в систему.', 'success')
        return redirect(url_for('login'))

    return render_template('register.html')
```

Вхід у систему перевіряє email та пароль, створює сесію з ID користувача, роллю та ім’ям, після чого перенаправляє на відповідний дашборд залежно від ролі.

Лістинг 3.2 Маршрут входу (фрагмент app.py)

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']

        conn = get_db_connection()
        user = conn.execute('SELECT * FROM users WHERE email = ?', (email,)).fetchone()
        conn.close()

        if user and check_password_hash(user['password_hash'], password):
            session['user_id'] = user['id']
            session['user_role'] = user['role']
            session['user_name'] = user['name']
```

```
            flash(f'Вітаємо, {user["name"]}!', 'success')
            if user['role'] == 'teacher':
              return redirect(url_for('teacher_dashboard'))
            else:
              return redirect(url_for('student_dashboard'))
          else:
            flash('Невірна пошта або пароль.', 'danger')

        return render_template('login.html')
```

Для захисту маршрутів реалізовано декоратор login_required, який перевіряє наявність сесії та відповідність ролі. Вихід очищає сесію.

Лістинг 3.3 Декоратор захисту маршрутів (фрагмент app.py)

```
def login_required(role=None):
  def decorator(f):
    def wrapper(*args, **kwargs):
      if 'user_id' not in session:
        flash('Будь ласка, увійдіть в систему.', 'warning')
        return redirect(url_for('login'))
      if role:
        conn = get_db_connection()
        user = conn.execute('SELECT * FROM users WHERE id = ?',
(session['user_id'],)).fetchone()
        conn.close()
        if user['role'] != role:
          abort(403) # Заборонено
      return f(*args, **kwargs)
    wrapper.__name__ = f.__name__
    return wrapper
  return decorator
```

Ця реалізація забезпечує базовий рівень безпеки (хешування паролів, захист сесій), простоту та надійність. У майбутньому можливе розширення до OAuth для інтеграції з Google, що спростить автентифікацію через існуючі акаунти освітніх

закладів. Система ролей чітко розділяє функціонал, запобігаючи несанкціонованому доступу до адміністративних функцій.

3.2. Реалізація модуля керування базами даних та задачами

Модуль керування базами даних та задачами є центральним для викладача, оскільки дозволяє створювати навчальний контент. Реалізація включає завантаження SQLite –файлів баз даних, їх зберігання в директорії media/databases з унікальними іменами, перегляд списку баз, а також створення, редагування та видалення завдань, прив’язаних до конкретної бази. Під час створення завдання викладач вказує еталонний SELECT –запит, який автоматично виконується для збереження очікуваного результату у JSON –форматі. Всі операції захищені декоратором login_required з роллю 'teacher'.

Завантаження бази даних реалізовано з перевіркою розширення файлу та генерацією унікального імені для запобігання колізій. Файл зберігається на сервері, а метадані – у таблиці databases.

Лістинг 3.4 Маршрут завантаження бази даних (фрагмент app.py)

```
@app.route('/teacher/databases/upload', methods=['GET', 'POST'])
@login_required(role='teacher')
def upload_database():
  if request.method == 'POST':
    title = request.form['title']
    description = request.form.get('description', '')
    file = request.files['file']

    if not file or file.filename == '':
      flash('Файл не вибрано.', 'danger')
      return redirect(url_for('upload_database'))

    if not allowed_file(file.filename):
      flash('Дозволено тільки SQLite файли (.sqlite, .db, .sqlite3).',
'danger')
      return redirect(url_for('upload_database'))

    filename = str(uuid.uuid4()) + '_' + secure_filename(file.filename)
    file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
```

```
            conn = get_db_connection()
            conn.execute('INSERT INTO databases (title, description, filename,
uploaded_by) VALUES (?, ?, ?, ?)',
                   (title, description, filename, session['user_id']))
            conn.commit()
            conn.close()
            flash('База даних успішно завантажена.', 'success')
            return redirect(url_for('teacher_databases'))

        return render_template('teacher/upload_database.html')
```

Перегляд списку баз даних та перехід до завдань реалізовано через запит до таблиці databases з фільтрацією за uploaded_by.

Лістинг 3.5 Маршрут перегляду баз даних (фрагмент app.py)

```
@app.route('/teacher/databases')
@login_required(role='teacher')
def teacher_databases():
    conn = get_db_connection()
    databases = conn.execute('SELECT * FROM databases WHERE uploaded_by = ?',
(session['user_id'],)).fetchall()
    conn.close()
    return render_template('teacher/databases.html', databases=databases)
```

Створення завдання включає форму з полями для умови, тематики, складності та еталонного запиту. Після надсилання запит виконується через сервіс query_executor, результат серіалізується в JSON та зберігається.

Лістинг 3.6 Маршрут створення завдання (фрагмент app.py)

```
@app.route('/teacher/database/<int:db_id>/tasks/create',      methods=['GET',
'POST'])
@login_required(role='teacher')
def create_task(db_id):
    conn = get_db_connection()
    db = conn.execute('SELECT * FROM databases WHERE id = ? AND uploaded_by =
?', (db_id, session['user_id'])).fetchone()
    if not db:
        flash('База даних не знайдена або доступ заборонено.', 'danger')
        return redirect(url_for('teacher_databases'))

    if request.method == 'POST':
        # ... (отримання полів форми)
        reference_query = request.form['reference_query']
```

```
            results, error = execute_query_safely(db['filename'], reference_query)
            if error:
              flash(f'Помилка в еталонному запиті: {error}', 'danger')
              conn.close()
              return render_template('teacher/create_task.html', db=db)

            expected_json = json.dumps(results, ensure_ascii=False, indent=2)

            conn.execute('''INSERT INTO tasks
                    (database_id, title, description, topic, subtopic, difficulty,
                     expected_result_json, created_by)
                    VALUES (?, ?, ?, ?, ?, ?, ?, ?)''',
                   (db_id,   title,   description,   topic,   subtopic,   difficulty,
expected_json, session['user_id']))
            conn.commit()
            conn.close()
            flash('Задача успішно створена.', 'success')
            return redirect(url_for('teacher_databases'))

        conn.close()
        return render_template('teacher/create_task.html', db=db)
```

Редагування та видалення завдань реалізовано аналогічно, з повторним виконанням еталонного запиту під час оновлення. Модуль інтегровано з дашбордом викладача для швидкого доступу. Реалізація забезпечує безпеку (перевірка власника бази та завдань), зручність та автоматичне збереження еталонних результатів, що є основою для подальшої перевірки студентських відповідей.

3.3. Реалізація механізмів автоматичної перевірки SQL –запитів та динамічної генерації індивідуальних наборів завдань

Механізми автоматичної перевірки SQL –запитів та динамічної генерації індивідуальних наборів завдань є ключовими особливостями платформи, що забезпечують об’єктивність оцінки та персоналізацію навчання. Автоматична перевірка реалізована через окремий сервіс query_executor.py, який виконує запити студентів у безпечному режимі та порівнює результати з еталонними. Динамічна генерація виконується в сервісі assignment_generator.py під час створення набору

завдань викладачем, формуючи унікальні комбінації завдань для кожного зареєстрованого студента на основі заданих критеріїв.

Автоматична перевірка SQL –запитів відбувається в ізольованому середовищі: дозволено тільки SELECT –запити, підключення до бази даних у read –only режимі, з обробкою помилок та конвертацією результату в нормалізований список словників для точного порівняння з еталонним JSON.

Лістинг 3.7 Сервіс безпечного виконання та перевірки запитів (services/query_executor.py)

```
import sqlite3
import json
import os
from config import Config

def execute_query_safely(db_filename, query):
 """
 Безпечне виконання SELECT –запиту на SQLite файлі.
 Використовуємо read –only режим, дозволяємо тільки SELECT.
 Повертає: (results_as_list_of_dicts, error_message)
 """
 if not query.strip().upper().startswith('SELECT'):
  return None, "Дозволено тільки SELECT –запити."

 db_path = os.path.join(Config.UPLOAD_FOLDER, db_filename)
 if not os.path.exists(db_path):
  return None, "База даних не знайдена."

 try:
  # Read –only режим
  conn = sqlite3.connect(f"file:{db_path}?mode=ro", uri=True)
  conn.row_factory = sqlite3.Row
  cursor = conn.cursor()
  cursor.execute(query)
```

```
        rows = cursor.fetchall()
        # Конвертуємо в список словників
        results = [dict(row) for row in rows]
        conn.close()
        return results, None
    except sqlite3.Error as e:
        return None, f"Помилка виконання: {str(e)}"
    except Exception as e:
        return None, f"Невідома помилка: {str(e)}"
```

Цей сервіс використовується як під час створення завдання (для збереження еталонного результату), так і під час виконання студентом (для порівняння).

Динамічна генерація індивідуальних наборів реалізовано в сервісі assignment_generator.py. Після створення набору викладачем сервіс фільтрує завдання за критеріями, випадково обирає кількість та конкретні завдання для кожного зареєстрованого студента, генерує унікальний share_code та зберігає зв’язки.

Лістинг 3.8 Сервіс генерації індивідуальних наборів (services/assignment_generator.py)

```
import random
import uuid
import json

def generate_assignments(conn, assignment_id):
    assignment = conn.execute('SELECT * FROM assignments WHERE id = ?', (assignment_id,)).fetchone()
    if not assignment:
        return

    tasks = conn.execute('SELECT * FROM tasks WHERE database_id = ?', (assignment['database_id'],)).fetchall()

    topics = json.loads(assignment['topics']) if assignment['topics'] else []
```

```
        difficulties = json.loads(assignment['difficulties']) if assignment['difficulties'] else []

        filtered_tasks = [t for t in tasks if (not topics or t['topic'] in topics) and (not difficulties or t['difficulty'] in difficulties)]

        if not filtered_tasks:
          return

        # Отримуємо всіх реальних студентів
        students = conn.execute('SELECT id, name, email FROM users WHERE role = ?', ('student',)).fetchall()

        if not students:
          return

        for student in students:
          num_tasks = random.randint(assignment['min_tasks'], assignment['max_tasks'])
          selected_tasks = random.sample(filtered_tasks, min(num_tasks, len(filtered_tasks)))

          share_code = str(uuid.uuid4())[:8].upper()

          student_name = student['name'] or student['email']

          cur = conn.execute('''INSERT INTO student_assignments
                  (assignment_id, student_id, student_name, share_code)
                  VALUES (?, ?, ?, ?)''',
                 (assignment_id, student['id'], student_name, share_code))
          student_assign_id = cur.lastrowid

          for task in selected_tasks:
            conn.execute('''INSERT INTO student_assignment_tasks
                 (student_assignment_id, task_id)
                 VALUES (?, ?)''',
```

```
            (student_assign_id, task['id']))

    conn.commit()
```

Генерація викликається після створення набору в маршруті create_assignment.

Лістинг 3.9 Фрагмент маршруту створення набору завдань з викликом генерації (app.py)

```
        if request.method == 'POST':
            # ... (отримання даних форми)
            cur = conn.execute('''INSERT INTO assignments
                    (title, database_id, created_by, min_tasks, max_tasks, topics, difficulties)
                    VALUES (?, ?, ?, ?, ?, ?, ?)''',
                   (title, db_id, session['user_id'], min_tasks, max_tasks, topics, difficulties))
            conn.commit()
            assignment_id = cur.lastrowid

            generate_assignments(conn, assignment_id)
            conn.close()
            flash('Набір завдань створено та згенеровано для студентів.', 'success')
            return redirect(url_for('teacher_assignments'))
```

Реалізовані механізми забезпечують безпеку (обмеження на SELECT, read – only), об’єктивність перевірки та індивідуалізацію навчання (унікальні набори з рандомізацією). Вони інтегровані з іншими модулями та легко тестуються, що підтверджує відповідність проектним вимогам.

3.4. Реалізація функціоналу відстеження результатів (для викладача та студента)

Функціонал відстеження результатів забезпечує викладачу моніторинг прогресу всієї групи студентів по кожному набору завдань, а студентові – перегляд власних досягнень та історії спроб. Реалізація базується на динамічному розрахунку прогресу з таблиці submissions: враховується лише остання спроба для кожної задачі (правильна чи неправильна). Для викладача відображається таблиця з іменами студентів, кількістю правильних відповідей, відсотком успішності та

прогрес –баром. Для студента – загальний прогрес по набору та детальна історія спроб для кожного завдання з відображенням результату виконання.

Для викладача статистика реалізовано в маршруті assignment_students, де виконуються запити для розрахунку правильних відповідей на основі останньої спроби. Це дозволяє динамічно оновлювати дані без додаткового зберігання.

Лістинг 3.10 Фрагмент розрахунку прогресу для викладача (маршрут assignment_students в app.py)

```
# Для кожного студента розраховуємо прогрес
student_progress = []
for student in students:
 total_tasks = conn.execute('''
  SELECT COUNT(*) FROM student_assignment_tasks WHERE student_assignment_id = ?
 ''', (student['sa_id'],)).fetchone()[0]

 correct_tasks = conn.execute('''
  SELECT COUNT(DISTINCT s.task_id)
  FROM submissions s
  JOIN (
   SELECT task_id, MAX(submitted_at) AS last_submit
   FROM submissions
   WHERE student_assignment_id = ?
   GROUP BY task_id
  ) latest ON s.task_id = latest.task_id AND s.submitted_at = latest.last_submit
  WHERE s.student_assignment_id = ? AND s.is_correct = 1
 ''', (student['sa_id'], student['sa_id'])).fetchone()[0]

 progress = (correct_tasks / total_tasks * 100) if total_tasks > 0 else 0
 student_progress.append({
  'student': student,
  'total_tasks': total_tasks,
  'correct_tasks': correct_tasks,
  'progress': round(progress, 2)})
```

Шаблон відображає прогрес –бар з кольоровим кодуванням (зелений – високий прогрес, жовтий – середній, червоний – низький).

Для студента прогрес та історія спроб реалізовано в маршруті student_assignment та відповідному шаблоні. Загальний прогрес розраховується динамічно з урахуванням останньої спроби, а історія відображається для кожного завдання з результатом у табличному форматі.

Лістинг 3.11 Фрагмент маршруту student_assignment з отриманням спроб (app.py)

```
submissions = conn.execute('''
 SELECT *
 FROM submissions
 WHERE student_assignment_id = ?
 ORDER BY submitted_at DESC
''', (student_assign['id'],)).fetchall()
```

У шаблоні використовується Jinja2 для розрахунку прогресу та відображення історії:

Лістинг 3.12 Фрагмент шаблону student/assignment.html з розрахунком прогресу

```
{% set total_tasks = tasks|length %}
{% set correct_tasks = 0 %}
{% for task in tasks %}
 {% set latest_sub = submissions|selectattr('task_id', 'equalto', task.id)|list|sort(attribute='submitted_at', reverse=True)|first %}
 {% if latest_sub and latest_sub.is_correct %}
  {% set correct_tasks = correct_tasks + 1 %}
 {% endif %}
{% endfor %}
{% set progress = (correct_tasks / total_tasks * 100)|round(2) if total_tasks > 0 else 0 %}
<div class="alert alert –info">
 <strong>Загальний прогрес:</strong> {{ correct_tasks }} / {{ total_tasks }} правильно ({{ progress }}%)
 <div class="progress mt –2">
```

```
<div class="progress –bar bg –success" style="width: {{ progress }}%">{{ progress }}%</div>
</div>
</div>
```

Функціонал відстеження результатів інтегровано з іншими модулями, забезпечуючи реальний час оновлення даних. Для викладача це інструмент контролю та аналізу групи, для студента – мотивація через візуальний прогрес та детальний фідбек. Реалізація є ефективною, без зайвого навантаження на базу даних, та відповідає вимогам до персоналізованого навчання.

3.5. Реалізація інтеграції з Google Classroom

Повна інтеграція з Google Classroom на поточному етапі розробки не реалізована, оскільки вимагає реєстрації додатка в Google Cloud Console, отримання credential–файлів та тестування з реальними курсами. Однак, архітектура системи спроектована з урахуванням майбутньої інтеграції, а ключові функції підготовлені як заглушки. Інтеграція планується через Google Classroom API з використанням OAuth 2.0 для автентифікації викладача, отримання списку курсів та студентів, створення завдань з посиланнями на індивідуальні набори та автоматичну передачу оцінок. Для цього передбачено окремий модуль classroom_integration з функціями авторизації та API –запитами.

Псевдокод 3.13 Автентифікація викладача через OAuth 2.0 (планована реалізація)

```
Функція authenticate_teacher():
 flow = InstalledAppFlow.from_client_secrets_file(
  'credentials.json',
  scopes=['https://www.googleapis.com/auth/classroom.courses',
     'https://www.googleapis.com/auth/classroom.rosters',
     'https://www.googleapis.com/auth/classroom.coursework.students']
 )
 creds = flow.run_local_server(port=0)
 зберегти creds в сесію або базі для викладача повернути creds
```

Після автентифікації викладач зможе обрати курс, система отримає список студентів та зіставити їх з зареєстрованими в платформі (за email).

Псевдокод 3.14 Створення завдання в Classroom з посиланням на набір

```
Функція publish_assignment_to_classroom(assignment_id, classroom_course_id):
  creds = отримати creds викладача
  service = build('classroom', 'v1', credentials=creds)

  для кожного student_assignment в наборі assignment_id:
    share_link = "https://платформа/assignment/" + student_assignment.share_code

    coursework = {
      'title': назва набору,
      'description': "Індивідуальний набір завдань з SQL",
      'workType': 'ASSIGNMENT',
      'state': 'PUBLISHED',
      'materials': [{'link': {'url': share_link}}]
    }
    результат = service.courses().courseWork().create(courseId=classroom_course_id,
body=coursework).execute()
    зберегти coursework_id в базі для подальшої синхронізації оцінок
```

Псевдокод 3.15 Передача оцінок назад до Classroom

```
Функція sync_grades_to_classroom(assignment_id, classroom_coursework_id):
  creds = отримати creds викладача
  service = build('classroom', 'v1', credentials=creds)

  для кожного student_assignment в наборі:
    grade = розрахувати прогрес (відсоток правильних завдань)

    submission = {
      'assignedGrade': grade,
      'draftGrade': grade
    }
    service.courses().courseWork().studentSubmissions().patch(
```

```
    courseId=classroom_course_id,
    courseWorkId=classroom_coursework_id,
    id=student_google_id,
    updateMask='assignedGrade,draftGrade',
    body=submission
  ).execute()
```

На поточному етапі інтеграція реалізована частково: викладач може вручну копіювати share_code та посилання для роздачі студентам через Classroom. Повна реалізація запланована як розширення після реєстрації додатка в Google Cloud та тестування API. Це дозволить автоматизувати розподіл завдань та синхронізацію оцінок, роблячи платформу повноцінним доповненням до Google Classroom у освітньому процесі.

3.6. Висновки

У третьому розділі описано реалізацію основних модулів розробленої інтерактивної навчальної платформи для практичного освоєння мови SQL. Реалізовано систему автентифікації з розрізненням ролей користувачів (викладач/студент) на основі сесій Flask та хешування паролів, що забезпечує базовий рівень безпеки та контроль доступу.

Розроблено модуль керування базами даних та задачами: викладачі можуть завантажувати SQLite –файли, створювати, редагувати та видаляти завдання з автоматичним збереженням еталонних результатів у JSON –форматі. Реалізовано механізми автоматичної перевірки SQL –запитів у безпечному read –only режимі з обмеженням на SELECT та динамічної генерації індивідуальних наборів завдань з рандомізацією за критеріями (кількість, теми, складність) для зареєстрованих студентів.

Впроваджено функціонал відстеження результатів: для викладача – статистика прогресу групи з прогрес –барами та розрахунком на основі останньої спроби; для студента – загальний прогрес по набору, історія спроб та детальні результати виконання. Інтеграція з Google Classroom спроектована та підготовлена

як заглушка з псевдокодом для OAuth 2.0, створення завдань та синхронізації оцінок, з можливістю повної реалізації в майбутньому.

Розроблена система відповідає сформульованим вимогам, демонструє стабільну роботу модулів та усуває недоліки аналогів (індивідуалізація, кастомні бази, безпека перевірки). Проведене функціональне тестування підтвердило коректність ключових функцій: створення контенту, генерацію наборів, перевірку запитів та відстеження прогресу. Платформа готова до використання в освітньому процесі з потенціалом розширення, зокрема повної інтеграції з Google Classroom.

ВИСНОВКИ

У бакалаврській роботі розроблено інтерактивну веб-орієнтовану навчальну платформу для практичного освоєння мови SQL з можливістю динамічної генерації індивідуальних наборів завдань, автоматичної перевірки SQL-запитів та підготовленою інтеграцією з Google Classroom. Поставлена мета досягнута в повному обсязі шляхом виконання всіх визначених завдань.

Проведений аналіз предметної області та існуючих аналогів (HackerRank, LeetCode, StrataScratch, LearnSQL.com, SQLZoo, DataCamp) виявив їхні суттєві недоліки: відсутність підтримки завантаження власних баз даних викладачем, динамічної генерації індивідуальних завдань, повноцінної інтеграції з системами управління навчанням та зручних інструментів для групового моніторингу прогресу. Це обґрунтувало доцільність створення нової платформи, орієнтованої саме на потреби вищої освіти.

У роботі спроектовано та реалізовано систему на основі технологічного стеку Python + Flask з використанням SQLite та Jinja2. Розроблено архітектуру, модель даних та основні функціональні модулі: автентифікацію з розмежуванням ролей, керування базами даних і завданнями, безпечну автоматичну перевірку запитів у read-only режимі, динамічну генерацію індивідуальних наборів завдань з рандомізацією, а також систему відстеження результатів і статистики.

Практична цінність розробленої системи полягає у створенні повноцінного навчального інструменту, який дозволяє викладачу завантажувати власні бази даних, швидко формувати індивідуальні завдання для студентів та отримувати об’єктивну статистику виконання. Платформа значно зменшує час на підготовку та перевірку завдань, підвищує персоналізацію навчання та знижує ризик списування завдяки механізму динамічної генерації наборів.

Можливість масштабування системи забезпечується модульною архітектурою, використанням легковагового стеку технологій та файловою структурою баз даних. Платформа може бути легко розгорнута як у локальному середовищі навчального закладу, так і в хмарі, а також адаптована для роботи з більшими групами студентів без суттєвих змін у коді.

Перспективи інтеграції з LMS/API є одним із ключових напрямків подальшого розвитку. У роботі спроектовано механізм інтеграції з Google Classroom на основі OAuth 2.0 та Classroom API. У майбутньому це дозволить реалізувати повноцінну двосторонню синхронізацію: автоматичну публікацію завдань, передачу оцінок у журнал Classroom, отримання списку студентів та зворотний зв'язок про виконання. Крім того, архітектура платформи допускає інтеграцію з іншими системами управління навчанням та зовнішніми API.

Таким чином, у бакалаврській роботі створено сучасну, функціонально повну та практично орієнтовану платформу, яка вирішує актуальні проблеми практичного навчання SQL у вищій школі. Розроблена система має високу практичну цінність, потенціал для масштабування та широкі перспективи інтеграції з освітніми платформами та сервісами.

Основні положення та результати кваліфікаційної роботи були апробовані на науково-практичних конференціях, зокрема:

1. Ступак Н.О. Доцільність проєктування системи практичного засвоєння мови SQL. // Матеріали Всеукраїнської науково-технічної конференції «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. К.: ДУІКТ, 1–3 травня 2026 року м. Київ. Збірник тез. (подано до друку).
2. Ступак Н.О. Методи інтеграції цифрової системи засвоєння мови SQL з Google Workspace. // Матеріали VI Всеукраїнської науково-практичної конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті». Збірник тез. К.: ДУІКТ, 15 травня 2026 року м. Київ. Збірник тез. (подано до друку).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Gaffney K. P. SQLite: Past, Present, and Future / K. P. Gaffney, M. Prammer, L. Brasfield [та ін.] // Proceedings of the VLDB Endowment. – 2022. – Vol. 15, No. 12. – P. 3535–3547.

2. Grinberg M. Flask Web Development: Developing Web Applications with Python. – Sebastopol : O'Reilly Media, 2014. – 258 p.

3. Grinberg M. The New and Improved Flask Mega –Tutorial. – Raleigh : The Pragmatic Bookshelf, 2025. – 327 p.

4. Relan K. Building REST APIs with Flask. – Berkeley : Apress, 2019. – 199 p.

5. Izaguirre E. SQLite in Modern Web Production: Dreams Becoming Reality [Електронний ресурс] // Los Angeles AI Apps. – Режим доступу: https://losangelesaiapps.com/sqlite –in –production –dreams –becoming –reality. – Дата доступу: 11.02.2026.

6. Hipp D. R. SQLite Documentation [Електронний ресурс] / D. R. Hipp, D. Kennedy // SQLite.org. – Режим доступу: https://www.sqlite.org/docs.html. – Дата доступу: 11.02.2026.

7. Anagu E. Top 5 Platforms to Learn SQL [Електронний ресурс] // LinkedIn. – Режим доступу: https://www.linkedin.com/posts/esther –anagu_top –5 –platforms –to –learn –sql –activity –7290300043901726720 – –7xl. – Дата доступу: 11.02.2026.

8. Milne J. The top SQL resources to up your skills and get that raise in 2019 [Електронний ресурс] // Klipfolio. – Режим доступу: https://www.klipfolio.com/blog/best –books –courses –resources –to –learn –sql. – Дата доступу: 11.02.2026.

9. Carrillo A. N. Where to learn SQL. My personal recommendations [Електронний ресурс] // Medium. – Режим доступу: https://ancarrillo964.medium.com/where –to –learn –sql –d5fdc8e69bb2. – Дата доступу: 11.02.2026.

10. Vertabelo Academy. 18+ Best Online Resources for Learning SQL and Database Concepts (2020 revision) [Електронний ресурс] // Vertabelo Academy Blog.

– Режим доступу: https://academy.vertabelo.com/blog/18 –best –online –resources –for –learning –sql –and –database –concepts. – Дата доступу: 11.02.2026.

11. Rivery Team. 10 Best Resources To Learn SQL For Free (2026) [Електронний ресурс] // Rivery. – Режим доступу: https://rivery.io/blog/6 –best –free –resources –for –learning –sql. – Дата доступу: 11.02.2026.

12. Google for Education. Apps that work with Google Classroom [Електронний ресурс] // Google Education. – Режим доступу: https://edu.google.com/intl/ALL_us/workspace –for –education/products/classroom/apps. – Дата доступу: 11.02.2026.

13. Southeast Missouri State University. Benefits of Integrating Technology in the Classroom [Електронний ресурс] // SEMO Blog. – Режим доступу: https://semo.edu/blog/blog –posts/integrating –technology –in –the –classroom. – Дата доступу: 11.02.2026.

14. eLearning Industry. Google Classroom Review: Its Pros And Cons In eLearning [Електронний ресурс] // eLearningIndustry. – Режим доступу: https://elearningindustry.com/google –classroom –review –pros –and –cons –of –using –google –classroom –in –elearning. – Дата доступу: 11.02.2026.

15. Branches Learning. Empowering Learning Everywhere: The Advantage of Google Classroom for TK –12 Online Education [Електронний ресурс] // Branches Learning. – Режим доступу: https://brancheslearning.org/empowering –learning –everywhere –the –advantage –of –google –classroom –for –tk –12 –online –education. – Дата доступу: 11.02.2026.

16. Research.com. How Google Conquered the Classroom: The Googlification of Schools Worldwide for 2026 [Електронний ресурс] // Research.com. – Режим доступу: https://research.com/education/how –google –conquered –the –classroom. – Дата доступу: 11.02.2026.

17. School Pathways. School Pathways' Integration with Google Classroom [Електронний ресурс] // School Pathways. – Режим доступу: https://schoolpathways.com/school –pathways –expands –access –to –educators –

favorite –online –learning –tool –with –google –classroom –integration. – Дата доступу: 11.02.2026.

18. Shake Up Learning. 85+ Awesome Apps that Integrate with Google Classroom [Електронний ресурс] // Shake Up Learning. – Режим доступу: https://shakeuplearning.com/blog/20 –awesome –apps –that –integrate –with –google –classroom. – Дата доступу: 11.02.2026.

19. StrataScratch. A Comprehensive Review of Online Platforms for SQL Practice [Електронний ресурс] // StrataScratch. – Режим доступу: https://www.stratascratch.com/blog/a –comprehensive –review –of –online –platforms –for –sql –practice. – Дата доступу: 11.02.2026.

20. LearnSQL.com. Top 3 Platforms to Learn SQL Online: A Complete Comparison [Електронний ресурс] // LearnSQL.com. – Режим доступу: https://learnsql.com/blog/top –platforms –learn –sql –online –comparison. – Дата доступу: 11.02.2026.

21. EPAM Careers. Мікросервіси на Python: як створити ефективні сучасні додатки [Електронний ресурс] // EPAM Careers. – Режим доступу: https://careers.epam.ua/blog/microservices –in –python –how –to –build –efficient –modern –apps. – Дата доступу: 11.02.2026.

22. Webcase. Розробка веб додатків з використанням Python і Django [Електронний ресурс] // Webcase. – Режим доступу: https://webcase.com.ua/uk/blog/razrabotka –veb –prilozhenij –s –ispolzovaniem –python –i –django. – Дата доступу: 11.02.2026.

23. FoxmindEd. Архітектурні шаблони на Python: приклади реалізації [Електронний ресурс] // FoxmindEd. – Режим доступу: https://foxminded.ua/realizatsiia –arkhitekturnykh –shabloniv –na –python. – Дата доступу: 11.02.2026.

24. Pidhoretskyy R. Розробка веб –застосунку засобами Python та фреймворку Django [Електронний ресурс] // Львівський національний університет імені Івана Франка. – Режим доступу: http://eprints.zu.edu.ua/42377/1/63.pdf. – Дата доступу: 11.02.2026.

25. Ситник Н. В. Сучасні бази даних NoSQL у підготовці бакалаврів спеціальності "Комп'ютерні науки" [Електронний ресурс] // ResearchGate. – Режим доступу: https://www.researchgate.net/publication/349652906_SUCASNI_BAZI_DANIH_NoSQL_U_PIDGOTOVCI_BAKALAVRIV_SPECIALNOSTI_KOMP'UTERNI_NAUKI. – Дата доступу: 11.02.2026.

26. Семенюк О. Інтелектуальна технологія перетворення природної мови в SQL –запити [Електронний ресурс] // Вісник ВНТУ. – Режим доступу: https://visnyk.vntu.edu.ua/index.php/visnyk/article/view/3226. – Дата доступу: 11.02.2026.

27. NURE. Розробка та дослідження методів автоматичного оцінювання SQL скриптів з використанням великих мовних моделей для системи тестування знань студентів [Електронний ресурс] // NURE Open Archive. – Режим доступу: https://openarchive.nure.ua/entities/publication/451d5d90 –d7ef –4941 –ac53 –c8163a3f8f05. – Дата доступу: 11.02.2026.

28. Ситник Н. В. Information Technologies and Learning Tools [Електронний ресурс] // Journal IITTA. – Режим доступу: https://journal.iitta.gov.ua/index.php/itlt/article/view/3098/1772.pdf. – Дата доступу: 11.02.2026.

29. Epic Web Dev. Why you should probably be using SQLite [Електронний ресурс] // Epic Web Dev. – Режим доступу: https://www.epicweb.dev/why –you –should –probably –be –using –sqlite. – Дата доступу: 11.02.2026.

30. Stack Overflow. SQLite as a production database for a low –traffic site [Електронний ресурс] // Stack Overflow. – Режим доступу: https://stackoverflow.com/questions/913067/sqlite –as –a –production –database –for –a –low –traffic –site. – Дата доступу: 11.02.2026.

31. Reddit. SQLite in Production? [Електронний ресурс] // Reddit. – Режим доступу: https://www.reddit.com/r/dotnet/comments/1bulzdi/sqlite_in_production. – Дата доступу: 11.02.2026.

32. Canadian Journal of Learning and Technology. Using Google Classroom as Assistive Technology in Universally Designed Classrooms [Електронний ресурс] // CJLT. – Режим доступу: https://cjlt.ca/index.php/cjlt/article/view/28456/20708. – Дата доступу: 11.02.2026.

33. ResearchGate. Using Google Classroom in Education: A New Tool to Facilitate Blended Learning and Improve Student Outcomes: A Systematic Review [Електронний ресурс] // ResearchGate. – Режим доступу: https://www.researchgate.net/publication/376076851_Using_Google_Classroom_in_Education_A_New_Tool_to_Facilitate_Blended_Learning_and_Improve_Student_Outcomes_A_Systematic_Review. – Дата доступу: 11.02.2026.

34. ERIC. The Preferred Use of Google Classroom Features for Online Learning in Indonesian EFL Classes [Електронний ресурс] // ERIC. – Режим доступу: https://files.eric.ed.gov/fulltext/EJ1434236.pdf. – Дата доступу: 11.02.2026.

35. Quora. Which books/sites are the best for learning complex (advanced) SQL queries? [Електронний ресурс] // Quora. – Режим доступу: https://www.quora.com/Which –books –sites –are –the –best –for –learning –complex –advanced –SQL –queries. – Дата доступу: 11.02.2026.

36. Reddit. Best resources to learn SQL and what should I focus on to get a job? [Електронний ресурс] // Reddit. – Режим доступу: https://www.reddit.com/r/SQL/comments/16uk93q/best_resources_to_learn_sql_and_what_should_i. – Дата доступу: 11.02.2026.

37. Facebook. What are good resources for SQL query training? [Електронний ресурс] // Facebook. – Режим доступу: https://www.facebook.com/groups/santeehappenings/posts/2558841994289551. – Дата доступу: 11.02.2026.

38. Packt. Mastering Flask Web Development [Електронний ресурс] // Packt. – Режим доступу: https://www.packtpub.com/en –us/product/mastering –flask –web –development –9781788999557. – Дата доступу: 11.02.2026.

39. Reddit. Any good tutorials on learning flask? Also looking into making APIs to use with React [Електронний ресурс] // Reddit. – Режим доступу:

https://www.reddit.com/r/flask/comments/17kympx/any_good_tutorials_on_learning_flask_also_looking. – Дата доступу: 11.02.2026.

ДОДАТОК А

Основний код зостосунку

```
from flask import Flask, render_template, request, redirect, url_for, flash, session, abort
from config import Config, allowed_file # якщо allowed_file у config.py
from database import init_db, get_db_connection
from werkzeug.security import generate_password_hash, check_password_hash
from flask_bootstrap import Bootstrap5
from werkzeug.utils import secure_filename
from services.query_executor import execute_query_safely
from services.query_executor import get_db_schema
import os
import uuid
import json # Новий імпорт

app = Flask(__name__)
app.config.from_object(Config)
bootstrap = Bootstrap5(app)

# Створюємо папку для завантажень
os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)

# Ініціалізуємо БД
init_db()

def login_required(role=None):
  def decorator(f):
    def wrapper(*args, **kwargs):
      if 'user_id' not in session:
```

```
            flash('Будь ласка, увійдіть в систему.', 'warning')
            return redirect(url_for('login'))
          if role:
            conn = get_db_connection()
            user = conn.execute('SELECT * FROM users WHERE id = ?',
(session['user_id'],)).fetchone()
            conn.close()
            if user['role'] != role:
              abort(403) # Заборонено
          return f(*args, **kwargs)
        wrapper.__name__ = f.__name__ # для url_for
        return wrapper
      return decorator

    @app.route('/')
    def index():
      if 'user_id' in session:
        conn = get_db_connection()
        user = conn.execute('SELECT role FROM users WHERE id = ?',
(session['user_id'],)).fetchone()
        conn.close()
        if user['role'] == 'teacher':
          return redirect(url_for('teacher_dashboard'))
        else:
          return redirect(url_for('student_dashboard'))
      # Для незареєстрованих – лендінг
      return render_template('index.html')

    # – – – – – – – – – – – – – – – – – – – – Автентифікація – – – – – – – – – – – – – – – – – – – –
```

```
@app.route('/register', methods=['GET', 'POST'])
def register():
  if request.method == 'POST':
    email = request.form['email']
    password = request.form['password']
    name = request.form['name']
    role = request.form['role'] # 'teacher' або 'student'

    if role not in ['teacher', 'student']:
      flash('Невірна роль.', 'danger')
      return redirect(url_for('register'))

    conn = get_db_connection()
    existing = conn.execute('SELECT * FROM users WHERE email = ?', (email,)).fetchone()
    if existing:
      flash('Користувач з такою поштою вже існує.', 'danger')
      conn.close()
      return redirect(url_for('register'))

    password_hash = generate_password_hash(password)
    conn.execute('INSERT INTO users (email, password_hash, name, role) VALUES (?, ?, ?, ?)',
          (email, password_hash, name, role))
    conn.commit()
    conn.close()
    flash('Реєстрація успішна. Увійдіть в систему.', 'success')
    return redirect(url_for('login'))
```

```
    return render_template('register.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']

        conn = get_db_connection()
        user = conn.execute('SELECT * FROM users WHERE email = ?', (email,)).fetchone()
        conn.close()

        if user and check_password_hash(user['password_hash'], password):
            session['user_id'] = user['id']
            session['user_role'] = user['role']
            session['user_name'] = user['name']
            flash(f'Вітаємо, {user["name"]}!', 'success')
            if user['role'] == 'teacher':
                return redirect(url_for('teacher_dashboard'))
            else:
                return redirect(url_for('student_dashboard'))
        else:
            flash('Невірна пошта або пароль.', 'danger')

    return render_template('login.html')

@app.route('/logout')
def logout():
    session.clear()
```

```
    flash('Ви вийшли з системи.', 'info')
    return redirect(url_for('index'))

# – – – – – – – – – – – – – – – – – – – – Заглушки дашбордів – – – – – – – – – – – – – – – – – – – –

@app.route('/student/dashboard')
@login_required(role='student')
def student_dashboard():
    conn = get_db_connection()

    # Отримуємо призначені набори з базовими даними
    raw_assignments = conn.execute('''
        SELECT sa.share_code, sa.id AS sa_id, a.title, a.id AS assignment_id, d.title AS db_title
        FROM student_assignments sa
        JOIN assignments a ON sa.assignment_id = a.id
        JOIN databases d ON a.database_id = d.id
        WHERE sa.student_id = ?
    ''', (session['user_id'],)).fetchall()

    # Розраховуємо прогрес для кожного набору
    assignments_with_progress = []
    for assign in raw_assignments:
        # Задачі для цього набору
        tasks = conn.execute('''
            SELECT COUNT(*) FROM student_assignment_tasks WHERE student_assignment_id = ?
        ''', (assign['sa_id'],)).fetchone()[0]
```

```
        if tasks == 0:
          progress = 0
        else:
          # Submissions для цього набору
          submissions = conn.execute('''
            SELECT * FROM submissions WHERE student_assignment_id = ? ORDER BY submitted_at DESC
          ''', (assign['sa_id'],)).fetchall()

          latest_submissions = {}
          for sub in submissions:
            task_id = sub['task_id']
            if task_id not in latest_submissions or sub['submitted_at'] > latest_submissions[task_id]['submitted_at']:
              latest_submissions[task_id] = sub

          correct_tasks = sum(1 for sub in latest_submissions.values() if sub['is_correct'])

          progress = round(correct_tasks / tasks * 100, 2)

        assignments_with_progress.append({
          'share_code': assign['share_code'],
          'title': assign['title'],
          'db_title': assign['db_title'],
          'total_tasks': tasks,
          'correct_tasks': correct_tasks if tasks > 0 else 0,
          'progress': progress
        })
```

```
    conn.close()
    return render_template('student/dashboard.html', assignments=assignments_with_progress)

import uuid
from flask import send_from_directory, render_template_string
from werkzeug.utils import secure_filename
from services.query_executor import execute_query_safely

# Список баз даних викладача
@app.route('/teacher/databases')
@login_required(role='teacher')
def teacher_databases():
    conn = get_db_connection()
    databases = conn.execute('SELECT * FROM databases WHERE uploaded_by = ?', (session['user_id'],)).fetchall()
    conn.close()
    return render_template('teacher/databases.html', databases=databases)

# Завантаження нової бази даних
@app.route('/teacher/databases/upload', methods=['GET', 'POST'])
@login_required(role='teacher')
def upload_database():
    if request.method == 'POST':
        title = request.form['title']
        description = request.form.get('description', '')
        file = request.files['file']

        if not file or file.filename == '':
            flash('Файл не вибрано.', 'danger')
```

```
            return redirect(url_for('upload_database'))

        if not allowed_file(file.filename):
            flash('Дозволено тільки SQLite файли (.sqlite, .db, .sqlite3).', 'danger')
            return redirect(url_for('upload_database'))

        filename = str(uuid.uuid4()) + '_' + secure_filename(file.filename)
        file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))

        conn = get_db_connection()
        conn.execute('INSERT INTO databases (title, description, filename,
uploaded_by) VALUES (?, ?, ?, ?)',
                     (title, description, filename, session['user_id']))
        conn.commit()
        conn.close()
        flash('База даних успішно завантажена.', 'success')
        return redirect(url_for('teacher_databases'))

      return render_template('teacher/upload_database.html')

    # Створення задачі для конкретної бази
    @app.route('/teacher/database/<int:db_id>/tasks/create',
methods=['GET', 'POST'])
    @login_required(role='teacher')
    def create_task(db_id):
      conn = get_db_connection()
      db = conn.execute('SELECT * FROM databases WHERE id = ? AND
uploaded_by = ?', (db_id, session['user_id'])).fetchone()
      if not db:
        flash('База даних не знайдена або доступ заборонено.', 'danger')
```

```
            return redirect(url_for('teacher_databases'))

        if request.method == 'POST':
            title = request.form['title']
            description = request.form['description']
            topic = request.form['topic']
            subtopic = request.form.get('subtopic', '')
            difficulty = request.form['difficulty']
            reference_query = request.form['reference_query']

            results, error = execute_query_safely(db['filename'], reference_query)
            if error:
                flash(f'Помилка в еталонному запиті: {error}', 'danger')
                conn.close()
                return render_template('teacher/create_task.html', db=db)

            expected_json = json.dumps(results, ensure_ascii=False, indent=2)

            conn.execute('''INSERT INTO tasks
                    (database_id, title, description, topic, subtopic, difficulty,
                     expected_result_json, created_by)
                    VALUES (?, ?, ?, ?, ?, ?, ?, ?)''',
                    (db_id, title, description, topic, subtopic, difficulty, expected_json,
session['user_id']))
            conn.commit()
            conn.close()
            flash('Задача успішно створена.', 'success')
            return redirect(url_for('teacher_databases'))

        conn.close()
```

```
    return render_template('teacher/create_task.html', db=db)

# Оновлення дашборду викладача
@app.route('/teacher/dashboard')
@login_required(role='teacher')
def teacher_dashboard():
    conn = get_db_connection()
    databases = conn.execute('SELECT * FROM databases WHERE uploaded_by = ?', (session['user_id'],)).fetchall()
    conn.close()
    return render_template('teacher/dashboard.html', databases=databases)

import random # Для рандомізації (додайте до імпортів, якщо відсутній)
from services.query_executor import get_db_schema
# Перегляд задач для бази даних

@app.route('/teacher/database/<int:db_id>/tasks')
@login_required(role='teacher')
def teacher_tasks(db_id):
    conn = get_db_connection()
    db = conn.execute('SELECT * FROM databases WHERE id = ? AND uploaded_by = ?', (db_id, session['user_id'])).fetchone()
    if not db:
        flash('База даних не знайдена або доступ заборонено.', 'danger')
        conn.close()
        return redirect(url_for('teacher_databases'))

    tasks = conn.execute('SELECT * FROM tasks WHERE database_id = ?', (db_id,)).fetchall()
```

```
        # Отримуємо схему БД (після визначення db!)
        schema = get_db_schema(db['filename'])

        conn.close()
        return render_template('teacher/tasks_list.html', db=db, tasks=tasks, schema=schema)

    # Редагування задачі
    @app.route('/teacher/task/<int:task_id>/edit', methods=['GET', 'POST'])
    @login_required(role='teacher')
    def edit_task(task_id):
      conn = get_db_connection()
      task = conn.execute('SELECT * FROM tasks WHERE id = ? AND created_by = ?', (task_id, session['user_id'])).fetchone()
      if not task:
        flash('Задача не знайдена або доступ заборонено.', 'danger')
        return redirect(url_for('teacher_databases'))

      db = conn.execute('SELECT * FROM databases WHERE id = ?', (task['database_id'],)).fetchone()

      if request.method == 'POST':
        title = request.form['title']
        description = request.form['description']
        topic = request.form['topic']
        subtopic = request.form.get('subtopic', '')
        difficulty = request.form['difficulty']
        reference_query = request.form['reference_query']

        results, error = execute_query_safely(db['filename'], reference_query)
```

```
            if error:
              flash(f'Помилка в еталонному запиті: {error}', 'danger')
              conn.close()
              return render_template('teacher/edit_task.html', task=task, db=db)

            expected_json = json.dumps(results, ensure_ascii=False, indent=2)

            conn.execute('''UPDATE tasks SET title=?, description=?, topic=?, subtopic=?, difficulty=?,
                    expected_result_json=?, reference_query=? WHERE id=?''',
                   (title, description, topic, subtopic, difficulty, expected_json, reference_query, task_id))
            conn.commit()
            conn.close()
            flash('Задача успішно оновлена.', 'success')
            return redirect(url_for('teacher_tasks', db_id=db['id']))

          conn.close()
          return render_template('teacher/edit_task.html', task=task, db=db)

        # Видалення задачі
        @app.route('/teacher/task/<int:task_id>/delete', methods=['POST'])
        @login_required(role='teacher')
        def delete_task(task_id):
          conn = get_db_connection()
          task = conn.execute('SELECT * FROM tasks WHERE id = ? AND created_by = ?', (task_id, session['user_id'])).fetchone()
          if task:
            db_id = task['database_id']
            conn.execute('DELETE FROM tasks WHERE id = ?', (task_id,))
```

```
    conn.commit()
    flash('Задача видалена.', 'info')
  else:
    flash('Задача не знайдена.', 'danger')
  conn.close()
  return redirect(url_for('teacher_tasks', db_id=db_id))


from services.assignment_generator import generate_assignments
from services.query_executor import get_db_schema

@app.route('/assignment/<share_code>', methods=['GET', 'POST'])
def student_assignment(share_code):
  conn = get_db_connection()


  student_assign = conn.execute(
    'SELECT * FROM student_assignments WHERE share_code = ?',
    (share_code,)
  ).fetchone()

  if not student_assign:
    flash('Набір завдань не знайдений або код недійсний.', 'danger')
    conn.close()
    return redirect(url_for('index'))

  assignment = conn.execute('''
    SELECT a.*, d.title AS db_title
    FROM assignments a
    JOIN databases d ON a.database_id = d.id
```

```
  WHERE a.id = ?
''', (student_assign['assignment_id'],)).fetchone()

tasks = conn.execute('''
  SELECT t.*
  FROM tasks t
  JOIN student_assignment_tasks sat ON t.id = sat.task_id
  WHERE sat.student_assignment_id = ?
''', (student_assign['id'],)).fetchall()

db_filename = conn.execute('''
  SELECT d.filename
  FROM databases d
  JOIN assignments a ON d.id = a.database_id
  WHERE a.id = ?
''', (student_assign['assignment_id'],)).fetchone()['filename']

submissions = conn.execute('''
  SELECT *
  FROM submissions
  WHERE student_assignment_id = ?
  ORDER BY submitted_at DESC
''', (student_assign['id'],)).fetchall()

# Додаємо схему (після db_filename!)
schema = get_db_schema(db_filename)

# Розрахунок прогресу (остання спроба для кожної задачі)
total_tasks = len(tasks)
correct_tasks = 0
```

```
        latest_submissions = {}

        for sub in submissions:
          task_id = sub['task_id']
          if task_id not in latest_submissions or sub['submitted_at'] > latest_submissions[task_id]['submitted_at']:
            latest_submissions[task_id] = sub

        for task in tasks:
          latest = latest_submissions.get(task['id'])
          if latest and latest['is_correct']:
            correct_tasks += 1

        progress = round(correct_tasks / total_tasks * 100, 2) if total_tasks > 0 else 0

        if request.method == 'POST':
          if 'user_id' not in session:
            flash('Увійдіть в систему для виконання завдань.', 'warning')
            conn.close()
            return redirect(url_for('login'))

          task_id = int(request.form['task_id'])
          sql_query = request.form['sql_query'].strip()

          results, error = execute_query_safely(db_filename, sql_query)

          if error:
            flash(f'Помилка виконання: {error}', 'danger')
            result_json = None
            is_correct = False
```

```
  else:
    expected_json = conn.execute(
      'SELECT expected_result_json FROM tasks WHERE id = ?',
      (task_id,)
    ).fetchone()['expected_result_json']
    expected = json.loads(expected_json)
    is_correct = (results == expected)
    result_json = json.dumps(results, ensure_ascii=False, indent=2)
    flash('Запит виконано: ' + ('Правильно!' if is_correct else 'Неправильно.'),
       'success' if is_correct else 'danger')

  conn.execute('''
    INSERT INTO submissions
    (student_assignment_id, task_id, sql_query, result_json, is_correct)
    VALUES (?, ?, ?, ?, ?)
  ''', (student_assign['id'], task_id, sql_query, result_json, int(is_correct)))
  conn.commit()
  conn.close()

  return redirect(url_for('student_assignment', share_code=share_code))

conn.close()
return render_template(
  'student/assignment.html',
  student_assign=student_assign,
  assignment=assignment,
  tasks=tasks,
  submissions=submissions,
  total_tasks=total_tasks,
  correct_tasks=correct_tasks,
```

```
    progress=progress,
    schema=schema # Нова змінна
  )

@app.route('/teacher/assignments')
@login_required(role='teacher')
def teacher_assignments():
  conn = get_db_connection()
  assignments = conn.execute('''
    SELECT a.*, d.title AS db_title
    FROM assignments a
    JOIN databases d ON a.database_id = d.id
    WHERE a.created_by = ?
  ''', (session['user_id'],)).fetchall()
  conn.close()
  return render_template('teacher/assignments_list.html', assignments=assignments)

@app.route('/teacher/assignment/<int:assignment_id>/students')
@login_required(role='teacher')
def assignment_students(assignment_id):
  conn = get_db_connection()
  assignment = conn.execute('SELECT * FROM assignments WHERE id = ? AND created_by = ?', (assignment_id, session['user_id'])).fetchone()
  if not assignment:
    flash('Набір не знайдений або доступ заборонено.', 'danger')
    return redirect(url_for('teacher_assignments'))

  # Отримуємо студентів з набором
  students = conn.execute('''
```

```
            SELECT sa.id AS sa_id, sa.student_name, sa.share_code, sa.student_id,
                u.name AS real_name, u.email
            FROM student_assignments sa
            LEFT JOIN users u ON sa.student_id = u.id
            WHERE sa.assignment_id = ?
        ''', (assignment_id,)).fetchall()

        # Для кожного студента розраховуємо прогрес
        student_progress = []
        for student in students:
            # Загальна кількість задач
            total_tasks = conn.execute('''
                SELECT COUNT(*) FROM student_assignment_tasks WHERE
student_assignment_id = ?
            ''', (student['sa_id'],)).fetchone()[0]

            # Кількість правильно виконаних (остання спроба)
            correct_tasks = conn.execute('''
                SELECT COUNT(DISTINCT s.task_id)
                FROM submissions s
                JOIN (
                    SELECT task_id, MAX(submitted_at) AS last_submit
                    FROM submissions
                    WHERE student_assignment_id = ?
                    GROUP BY task_id
                ) latest ON s.task_id = latest.task_id AND s.submitted_at = latest.last_submit
                WHERE s.student_assignment_id = ? AND s.is_correct = 1
            ''', (student['sa_id'], student['sa_id'])).fetchone()[0]

            progress = (correct_tasks / total_tasks * 100) if total_tasks > 0 else 0
```

```
            student_progress.append({
              'student': student,
              'total_tasks': total_tasks,
              'correct_tasks': correct_tasks,
              'progress': round(progress, 2)
            })

        conn.close()
        return render_template('teacher/assignment_students.html', assignment=assignment, student_progress=student_progress)

    @app.route('/teacher/database/<int:db_id>/assignment/create', methods=['GET', 'POST'])
    @login_required(role='teacher')
    def create_assignment(db_id):
        conn = get_db_connection()
        db = conn.execute('SELECT * FROM databases WHERE id = ? AND uploaded_by = ?', (db_id, session['user_id'])).fetchone()
        if not db:
            flash('База не знайдена.', 'danger')
            return redirect(url_for('teacher_databases'))

        if request.method == 'POST':
            title = request.form['title']
            min_tasks = int(request.form['min_tasks'])
            max_tasks = int(request.form['max_tasks'])
            topics = json.dumps(request.form.getlist('topics') or [])
            difficulties = json.dumps(request.form.getlist('difficulties') or [])

            # Виконуємо INSERT та отримуємо курсор
```

```
            cur = conn.execute('''INSERT INTO assignments
                    (title, database_id, created_by, min_tasks, max_tasks, topics, difficulties)
                    VALUES (?, ?, ?, ?, ?, ?, ?)''',
                   (title, db_id, session['user_id'], min_tasks, max_tasks, topics, difficulties))
            conn.commit()

            assignment_id = cur.lastrowid # Правильно: з курсора

            generate_assignments(conn, assignment_id) # Генерація кодів
            conn.close()

            flash('Набір завдань створено. Коди доступу згенеровано.', 'success')
            return redirect(url_for('teacher_assignments'))

        # Унікальні теми та складності для форми
        unique_topics = [row['topic'] for row in conn.execute('SELECT DISTINCT topic FROM tasks WHERE database_id = ?', (db_id,)).fetchall()]
        unique_diff = [row['difficulty'] for row in conn.execute('SELECT DISTINCT difficulty FROM tasks WHERE database_id = ?', (db_id,)).fetchall()]
        conn.close()

        return render_template('teacher/create_assignment.html', db=db, unique_topics=unique_topics, unique_diff=unique_diff)

    from werkzeug.security import generate_password_hash, check_password_hash

    @app.route('/profile', methods=['GET', 'POST'])
    @login_required()
    def profile():
        conn = get_db_connection()
```

```
    user = conn.execute('SELECT * FROM users WHERE id = ?', (session['user_id'],)).fetchone()

    if request.method == 'POST':
      name = request.form['name'].strip()
      email = request.form['email'].strip().lower()
      new_password = request.form.get('new_password', '').strip()
      confirm_password = request.form.get('confirm_password', '').strip()

      # Валідація
      if not name or not email:
        flash('ПІБ та email обов’язкові.', 'danger')
      elif email != user['email']:
        existing = conn.execute('SELECT id FROM users WHERE email = ? AND id != ?', (email, session['user_id'])).fetchone()
        if existing:
          flash('Цей email вже використовується.', 'danger')
          conn.close()
          return render_template('profile.html', user=user)

      if new_password:
        if len(new_password) < 6:
          flash('Пароль має бути не менше 6 символів.', 'danger')
        elif new_password != confirm_password:
          flash('Паролі не співпадають.', 'danger')
        else:
          password_hash = generate_password_hash(new_password)
          conn.execute('UPDATE users SET password_hash = ? WHERE id = ?', (password_hash, session['user_id']))
```

```
        # Оновлення основних даних
        conn.execute('UPDATE users SET name = ?, email = ? WHERE id = ?',
                     (name, email, session['user_id']))
        conn.commit()
        conn.close()

        # Оновлення сесії
        session['user_name'] = name

        flash('Дані успішно оновлено!', 'success')
        return redirect(url_for('profile'))

    conn.close()
    return render_template('profile.html', user=user)

if __name__ == '__main__':
    app.run(debug=True)
```

ДОДАТОК Б

Код генерації завдань

```
import random
import uuid
import json

def generate_assignments(conn, assignment_id):
  assignment = conn.execute('SELECT * FROM assignments WHERE id = ?', (assignment_id,)).fetchone()
  if not assignment:
    return

  tasks = conn.execute('SELECT * FROM tasks WHERE database_id = ?', (assignment['database_id'],)).fetchall()

  topics = json.loads(assignment['topics']) if assignment['topics'] else []
  difficulties = json.loads(assignment['difficulties']) if assignment['difficulties'] else []

  filtered_tasks = [t for t in tasks if (not topics or t['topic'] in topics) and (not difficulties or t['difficulty'] in difficulties)]

  if not filtered_tasks:
    return

  # Отримуємо всіх реальних студентів з бази даних
  students = conn.execute('SELECT id, name, email FROM users WHERE role = ?', ('student',)).fetchall()
```

```
        if not students:
          return # Якщо немає студентів – нічого не генеруємо

        for student in students:
          num_tasks = random.randint(assignment['min_tasks'], assignment['max_tasks'])
          selected_tasks = random.sample(filtered_tasks, min(num_tasks, len(filtered_tasks)))

          share_code = str(uuid.uuid4())[:8].upper() # Короткий код, наприклад A1B2C3D4

          student_name = student['name'] or student['email']

          # Вставляємо з посиланням на реального студента (student_id)
          cur = conn.execute('''INSERT INTO student_assignments
                  (assignment_id, student_id, student_name, share_code)
                  VALUES (?, ?, ?, ?)''',
                 (assignment_id, student['id'], student_name, share_code))
          student_assign_id = cur.lastrowid

          # Зв'язуємо вибрані задачі
          for task in selected_tasks:
            conn.execute('''INSERT INTO student_assignment_tasks
                 (student_assignment_id, task_id)
                 VALUES (?, ?)''',
                (student_assign_id, task['id']))

        conn.commit()
```

ДОДАТОК В

Код інтерфейсу адміністратора

```
{% extends "base.html" %}
{% block content %}
<div class="container mt –5">
 <h1 class="display –5 fw –bold text –center mb –5" style="color: #667eea;">Панель викладача</h1>
 <p class="text –center fs –4 text –muted mb –5">Вітаємо, {{ session.user_name }}! Керуйте базами даних, задачами та наборами завдань.</p>

 <! – – Статистика – –>
 <div class="row mb –5">
  <div class="col –md –4">
   <div class="card shadow –sm border –0 text –center p –4">
    <i class="bi bi –database fs –1 text –primary mb –3"></i>
    <h3>{{ databases|length }}</h3>
    <p class="text –muted">Завантажених баз даних</p>
   </div>
  </div>
  <div class="col –md –4">
   <div class="card shadow –sm border –0 text –center p –4">
    <i class="bi bi –list –task fs –1 text –primary mb –3"></i>
    <h3>{{ assignments|length if assignments else 0 }}</h3>
    <p class="text –muted">Створених наборів завдань</p>
   </div>
  </div>
  <div class="col –md –4">
   <div class="card shadow –sm border –0 text –center p –4">
    <i class="bi bi –people fs –1 text –primary mb –3"></i>
```

```
            <h3>{{ total_students|default(0) }}</h3>
            <p class="text –muted">Студентів у системі</p>
          </div>
        </div>
      </div>

      <! – – Дії – –>
      <div class="text –center mb –5">
        <a href="{{ url_for('upload_database') }}" class="btn btn –primary btn –lg me –3"><i class="bi bi –upload"></i> Завантажити базу</a>
        <a href="{{ url_for('teacher_databases') }}" class="btn btn –outline –primary btn –lg me –3"><i class="bi bi –database"></i> Мої бази</a>
        <a href="{{ url_for('teacher_assignments') }}" class="btn btn –outline –primary btn –lg"><i class="bi bi –collection"></i> Мої набори</a>
      </div>

      <! – – Список баз даних – –>
      {% if databases %}
      <h3 class="fw –bold mb –4" style="color: #667eea;">Ваші бази даних</h3>
      <div class="row">
        {% for db in databases %}
        <div class="col –md –6 mb –4">
          <div class="card shadow –sm h –100">
            <div class="card –body">
              <h5 class="card –title"><i class="bi bi –database –fill"></i> {{ db.title }}</h5>
              <p class="card –text text –muted">{{ db.description|default('Без опису') }}</p>
              <div class="mt –3">
```

 Задачі

 Створити набір

{% endfor %}

{% else %}

Ви ще не завантажили жодної бази даних. Почніть з завантаження!

{% endif %}

{% endblock %}