Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавра

зі спеціальності 121 – Інженерія програмного забезпечення

На тему: Розробка застосунку для структурованого документування результатів тестування програмного забезпечення

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.

Студент гр. ІПЗ-21-2

_____ /А. Є. Нехаєва /

Керівник кваліфікаційної роботи _____ / І. О. Доценко /

Завідувач кафедри _____ / А. М. Стрюк /

Кривий Ріг

2025

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Ступінь вищої освіти: бакалавр

Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри

_____ А. М. Стрюк

«___» _____ 2025 р.

ЗАВДАННЯ

на кваліфікаційну роботу

студенту групи ІПЗ-21-2 Нехаєвій Анастасії Євгенівній

1. Тема: Розробка застосунку для структурованого документування результатів тестування програмного забезпечення затверджено наказом по КНУ № 200с від «10» квітня 2025 р.
2. Термін подання студентом закінченої роботи: «15» червня 2025р.
3. Вихідні дані по роботі: Система повинна забезпечувати роботу з тест-кейсами, баг-репортами, завданнями та користувачами (адмін, викладач, студент). Передбачено базу даних, розмежування прав доступу, завантаження Excel-файлів.
4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): виконати аналіз подібних систем, спроєктувати структуру бази даних, реалізувати функціонал додавання, редагування, видалення звітів, авторизацію та розмежування доступу, завантаження студентів з Excel та тестування системи.
5. Перелік ілюстративного матеріалу: ER-діаграма бази даних, граф інтерфейсу користувача, блок-схеми логіки ролей і перевірки, скріншоти основних сторінок: панелі викладача, студента, адміністратора, зразки форм додавання/редагування звітів.

Календарний план:

№ з/п	Найменування етапів кваліфікаційної роботи	Термін виконання етапів робіт
1	Вивчення літератури за темою та аналіз аналогічних систем	10.04.2025 – 14.04.2025
2	Проєктування архітектури вебдодатку та структури бази даних	15.04.2025 – 19.04.2025
3	Реалізація функціоналу для студентів (додавання, редагування, видалення звітів)	20.04.2025 – 26.04.2025
4	Реалізація функціоналу для студентів (додавання, редагування, видалення звітів)	27.04.2025 – 02.05.2025
5	Реалізація функціоналу для адміністратора (керування користувачами, ролями)	03.05.2025 – 07.05.2025
6	Інтеграція завантаження студентів з файлу електронних таблиць MS Excel Excel	08.05.2025 – 10.05.2025
7	Тестування системи та виправлення помилок	11.05.2025 – 16.05.2025
8	Підготовка матеріалів пояснювальної записки (розділи 1-3)	17.05.2025 – 24.05.2025
9	Оформлення пояснювальної записки, додатків, списку джерел	25.05.2025 – 01.06.2025
10	Перевірка, підписання, подання завершеної кваліфікаційної роботи	02.06.2025 – 15.06.2025

Дата видачі завдання: «10» квітня 2025 р.

Студент _____ / А. Є. Нехаєва /

Керівник роботи _____ / І. О. Доценко /

РЕФЕРАТ

ТЕСТ-КЕЙС, БАГ-РЕПОРТ, ВЕБ-ДОДАТОК, FLASK, СИСТЕМА КЕРУВАННЯ ЗВІТАМИ, РОЛІ КОРИСТУВАЧІВ, ЗБЕРІГАННЯ ДАНИХ, ТЕСТУВАННЯ.

Пояснювальна записка: 73 с., 2 табл., 27 рис., 1 дод., 14 джерел.

Метою кваліфікаційної роботи є створення вебзастосунку для управління тест-кейсами та баг-репортами у процесі навчального тестування програмного забезпечення, з урахуванням ролей користувачів – студентів, викладачів та адміністратора.

У роботі проведено аналіз існуючих систем управління звітами про тестування, вивчено сучасні інструменти веброзробки та обґрунтовано вибір технологій для реалізації (Python, Flask, HTML/CSS, SQLite). Спроєктовано структуру бази даних та інтерфейс користувача.

Здійснено реалізацію основного функціоналу:

– Для студентів – додавання, редагування та видалення тест-кейсів і баг-репортів.

– Для викладачів – перевірка, коментування та оцінювання звітів та завантаження студентів із Excel-файлів.

– Для адміністратора – керування користувачами.

У додатку впроваджено AJAX-оновлення сторінки, обмеження за ролями, автоматичне збереження, перевірки на унікальність логінів.

Проведено тестування розробленої системи, оформлено результати, наведено приклади використання.

Основні положення кваліфікаційної роботи можуть бути використані для створення внутрішньої системи навчання та документування результатів тестування програмного забезпечення в університеті.

ABSTRACT

TEST CASE, BUG REPORT, WEB APPLICATION, FLASK, REPORT MANAGEMENT SYSTEM, USER ROLES, DATA STORAGE, TESTING.

Explanatory note: 73 pages, 2 tables, 27 figures, 1 appendix, 14 sources.

The purpose of this thesis is to create a web application for managing test cases and bug reports in the process of educational software testing, taking into account the roles of users – students, teachers and administrators.

The work analyses existing test report management systems, studies modern web development tools and justifies the choice of technologies for implementation (Python, Flask, HTML/CSS, SQLite). The database structure and user interface have been designed.

The main functionality has been implemented:

– For students – adding, editing and deleting test cases and bug reports.

– For teachers – checking, commenting on and evaluating reports and uploading students from Excel files.

– For the administrator – user management.

The application implements AJAX page refresh, role restrictions, automatic saving, and login uniqueness checks.

The developed system was tested, the results were documented, and examples of use were provided.

The main provisions of the qualification work can be used to create an internal training system and document the results of software testing at the university.

ЗМІСТ

ВСТУП

У сучасному світі розробка якісного програмного забезпечення неможлива без належного тестування. В умовах зростаючої складності інформаційних систем та високих вимог до їхньої надійності особливої актуальності набуває автоматизація процесів тестування. Важливо не лише виявити помилки, а й систематизувати, зберігати та оцінювати інформацію про них у зручному для аналізу вигляді.

Підготовка фахівців у сфері інформаційних технологій вимагає від студентів засвоєння як теоретичних основ, так і практичних навичок тестування програмного забезпечення. Саме тому виникає потреба у створенні вебзастосунку, який дозволить студентам створювати тест-кейси та баг-репорти, а викладачам – перевіряти, коментувати та оцінювати ці звіти.

Метою даної кваліфікаційної роботи є розробка вебдодатку з використанням мови програмування Python, фреймворку Flask та СУБД SQLite для підтримки процесу тестування в навчальному середовищі. У системі буде реалізовано підтримку ролей користувачів (студент, викладач, адміністратор), функціонал взаємодії з базами даних тест-кейсів і баг-репортів, можливість редагування даних та завантаження списків студентів з Excel-файлів.

1 СУЧАСНИЙ СТАН І АКТУАЛЬНІСЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1.1 Актуальність теми кваліфікаційної роботи

У сучасному світі галузь освіти має швидкий розвиток та цифрову трансформацію, що об’єднує усі навчальні процеси такі як організація практичних занять, контроль знань та оцінка досягнень студентів. Через високий попит виникає необхідність створення власних цифрових платформ для реалізації занять та навчання студентів. Дані платформи йдуть більш орієнтовані на потреби конкретного навчального закладу, які відповідають силабусу з відповідної дисципліни.

До минулого року, завдяки існуючому Меморандуму про співпрацю між ТОВ «Кьюатестлаб» та нашим університетом для вивчення дисципліни «Якість програмного забезпечення та тестування», ТОВ «Кьюатестлаб» надавало безкоштовний доступ до системи відслідковування дефектів Mantis Bug Tracker, системи управління тест-кейсами TestRail та доступ до WEB100 Platform: QATestLab Univer, призначеної для роботи викладача та особистого кабінету, призначеного для роботи студента.

Минулого року відбулось розірвання Меморандуму про співпрацю і постала проблема тотожної заміни платформ для вивчення дисципліни «Якість програмного забезпечення та тестування». Наразі дані системи (Mantis Bug Tracker та TestRail) мають обмежений функціонал та платний доступ. Це створює великі труднощі в навчальному процесі та гальмує розвиток освіти студентів.

Виходячи з цього, було прийнято рішення розробити вебсайт для документування результатів тестування програмного забезпечення, який дозволить створювати баг-репорти та тест-кейси. За допомогою даного вебресурсу буде можливість проводити навчання з дисципліни «Якість програмного забезпечення та тестування». Буде передбачено можливість створювати профілі студентів та викладачів. Дана платформа стане універсальним інструмен-

том для навчання, фіксації, перевірки та оцінки результатів виконання поставлених завдань. Дозволить зворотній зв’язок між студентом та викладачем. Даний вебсайт підвищить автономність університету.

1.2 Цілі та завдання кваліфікаційної роботи

Мета кваліфікаційної роботи – розробка застосунку для структурованого документування результатів тестування програмного забезпечення.

Об’єктом дослідження кваліфікаційної роботи є системи для відстеження помилок або ж багтрекінгові системи, а також системи для створення тест-кейсів для тестування програмного забезпечення.

Предметом розробки кваліфікаційної роботи є застосунок для структурованого документування результатів тестування програмного забезпечення.

Виходячи з поставленої мети маємо такі завдання:

а) Провести аналіз існуючих аналогів, які призначені для тестування програмного забезпечення. Визначити які вони мають переваги та недоліки.

б) Провести дослідження.

1) Зібрати потрібні рекомендації потенційних користувачів (викладачів, студентів).
2) Проаналізувати зібрану інформацію та створити план розробки.

в) Обрати мови програмування. Розробити логічну та фізичну бази даних. Розробити клієнтську та серверну частини вебсайту.

г) Написати програмний код до відповідного вебресурсу. Підв’язати базу даних та забезпечити відповідні функції для роботи з нею, а саме: збереження, читання та редагування даних.

Система має стати зручним ресурсом, як для викладачів так і для студентів, виходячи з цього маємо такі функціональні вимоги:

а) Реєстрація та авторизація.

б) Можливість публікування завдань.

в) Створення звітів тестування (форми створення тест-кейсів та баг-репортів).

г) Перевірка та оцінювання (перегляд звітів, додавання коментарів та оцінювання).

д) Зворотній зв’язок.

1.3 Аналіз існуючих аналогів

Дана кваліфікаційна робота позиціонує себе, як створення вебсайту, за допомогою якого будуть аналізуватися результати досліджень та тестувань програмного забезпечення. На сьогодні існує багато аналогів таких сайтів вони мають назву, як система для відстеження помилок або ж багтрекінгова система, які контролюють знайдені помилки та допомагають відстежувати процеси усунення цих помилок [1].

Для проведення аналізу було обрано такі варіанти систем: Jira Software, YouTrack JetBrains та MantisBT.

Сервіс Jira Software – це популярна багтрекінгова система, яка є найпопулярнішим інструментом для створення завдань та управління проєктами (див. рис. 1.1).

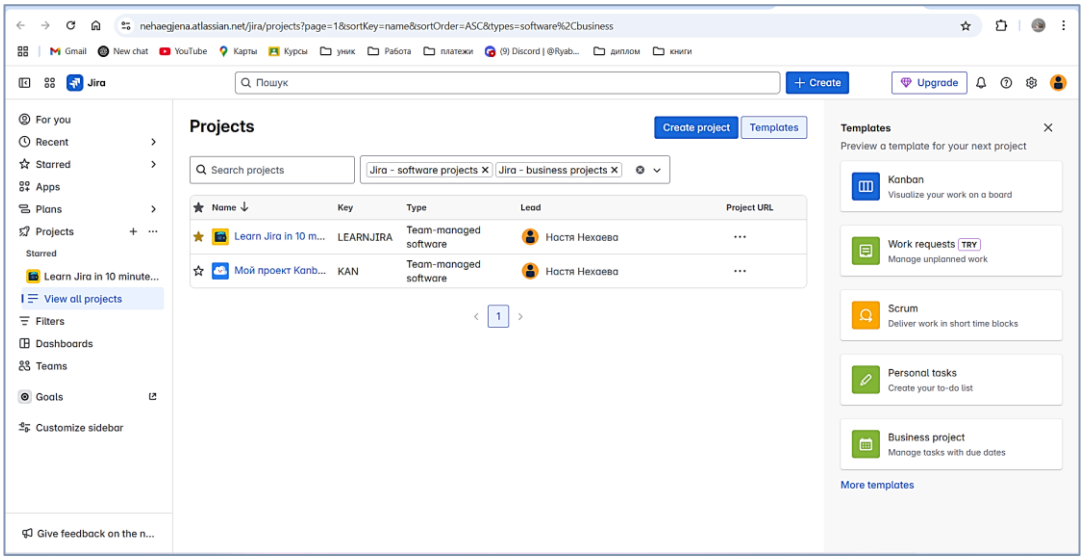

Рисунок 1.1 – Сервіс Jira Software

Найпопулярнішим функціоналом сервісу Jira Software є створення інтерактивних дошок, де можна контролювати процес виконання проєкту. Дана

платформа є платною, через це має дуже обмежений функціонал у безкоштовній версії. Виходячи з цього, маємо те, що даний ресурс обмежує наші можливості у використанні, що призводить до неможливості повноцінного використання даного ресурсу для навчання в університеті [2].

Система YouTrack JetBrains – одна з найвідоміших багтрекінкових систем, яка має такі функціональні можливості: створення баг-трекерів та створення дошок для контролю проєктів (див. рис. 1.2). Даний вебресурс також є платним, тому це ускладнює його використання в навчанні [3].

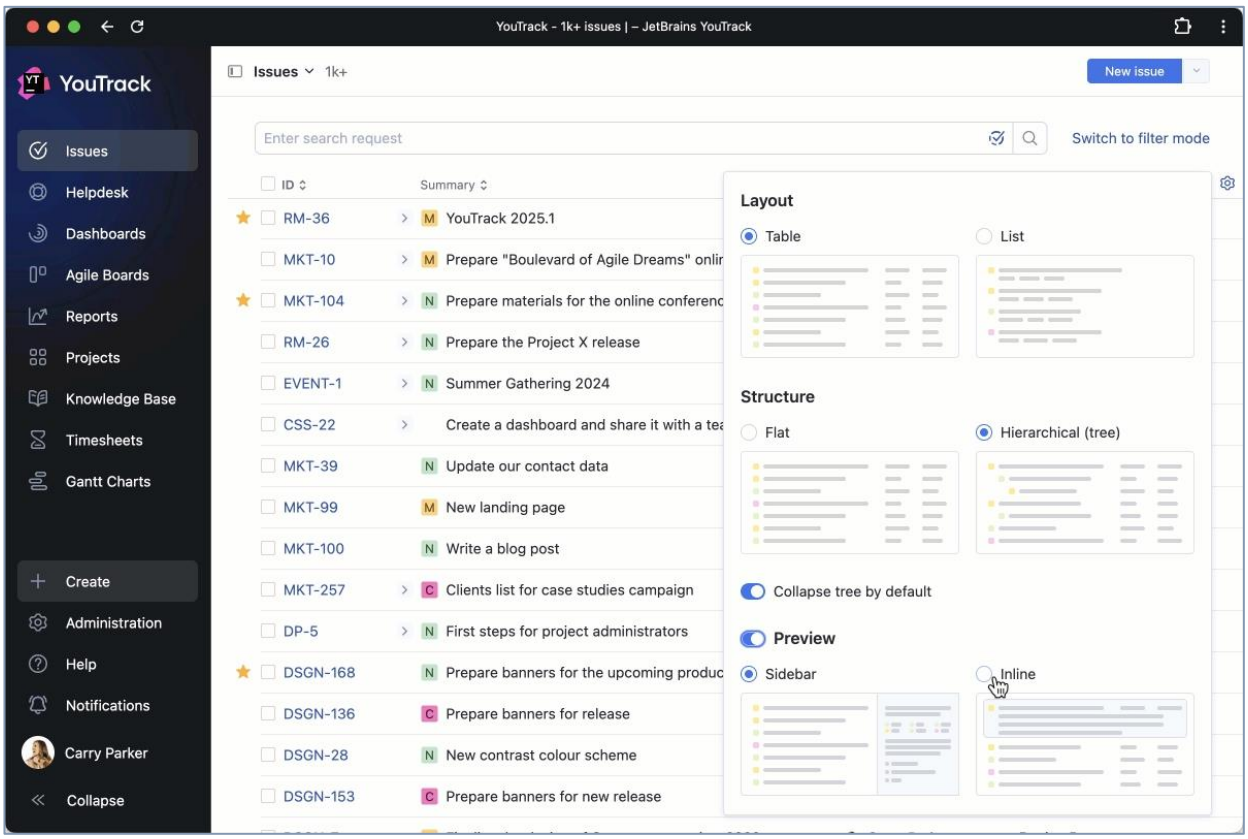

Рисунок 1.2 – Багтрекінкова система YouTrack JetBrains

Веб ресурс MantisBT – дана багтрекінкова система є одною з відомих (див. рис. 1.3). Вона використовується в основному для невеликих груп та індивідуальних проєктів. Вона є безкоштовною, що є значним плюсом у її використанні, але дана система має застарілий дизайн та погано скомпонований інтерфейс. Цим саме створюючи додаткові проблеми для вивчення та аналізу системи, перед початком роботи [4].

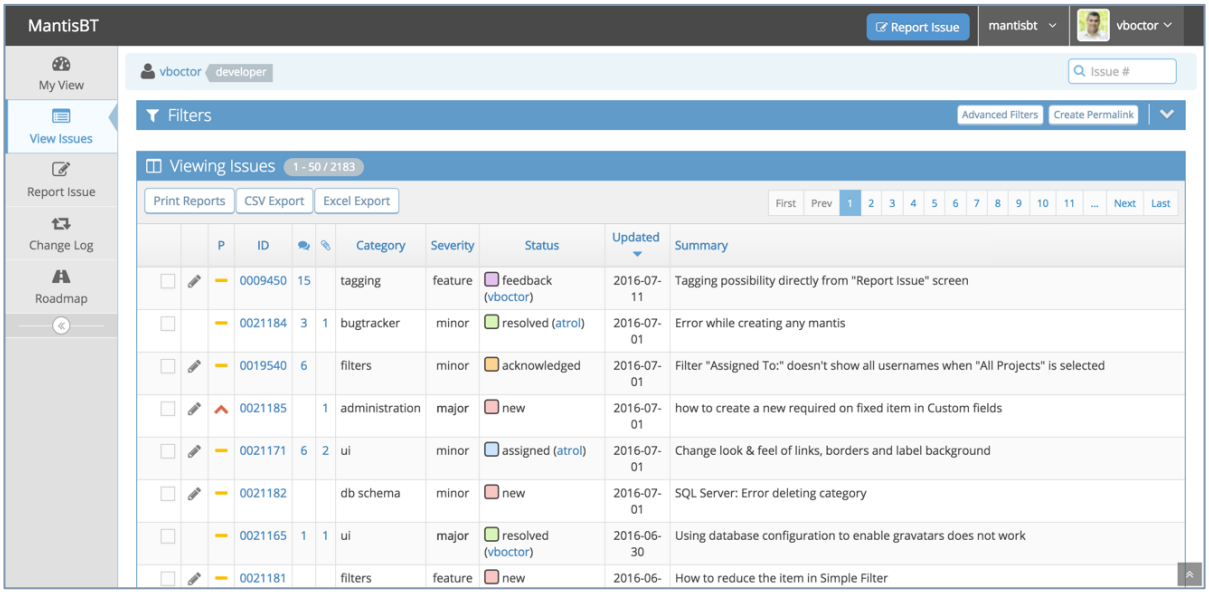

Рисунок 1.3 – Багтрекінкова система MantisBT

Зробивши короткий опис даних систем, можна зробити таку коротку порівняльну таблицю, яка об’єднає усі результати, та наглядно продемонструє різницю між трьома схожими аналогами (див. таб. 1.1).

Таблиця 1.1 – Порівняльна характеристика аналогів багтрекінкових систем

№	Система	Платформа	Ліцензія	Гнучкість	Основні задачі
1	Jira Software	Веб, хмара (SaaS), локально	Платна	Сучасний професійний інтерфейс	Задачі, баги, спринти, workflow, дашборди
2	YouTrack JetBrains	Веб, хмара (SaaS), локально	Платна	Сучасний адаптивний інтерфейс	Завдання, баги, борди, автоматизація
3	MantisBT	Локально, веб	Безкоштовна	Простий, але застарілий інтерфейс	Баги, коментарі, статуси, файли

Оскільки створюваний вебресурс призначений не лише для фіксації помилок, але й керуванні тестуванням, було доцільно включити аналіз аналогів з систем створення тест-кейсів. Обрано дві найпопулярніші системи TestRail та TestLink.

Система TestRail – один з популярних вебресурсів серед тестувальників, яка має відповідні функціональні можливості: створювати, організовувати та аналізувати тест-кейси (див. рис. 1.4). Серед переваг має інтуїтивний інтерфейс та широкі можливості інтеграції. Недоліком є платна підписка на використання [5].

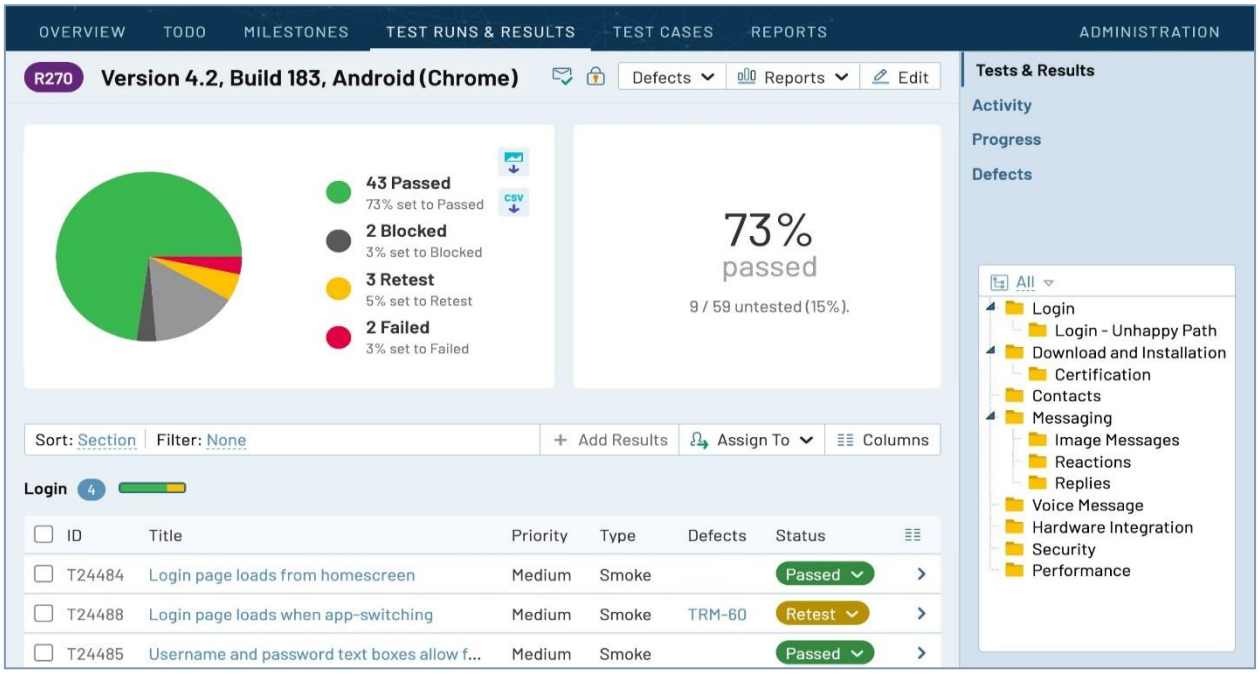

Рисунок 1.4 – Система TestRail

Система TestLink є також популярною системою, серед ресурсів для керування тестуванням (див. рис. 1.5). Має застарілий інтерфейс, але є безкоштовним ресурсом. Основним функціоналом є планування тестування, ведення тестової інформації та відстеження результатів виконання тестів. Дана система має складне початкове налаштування [6].

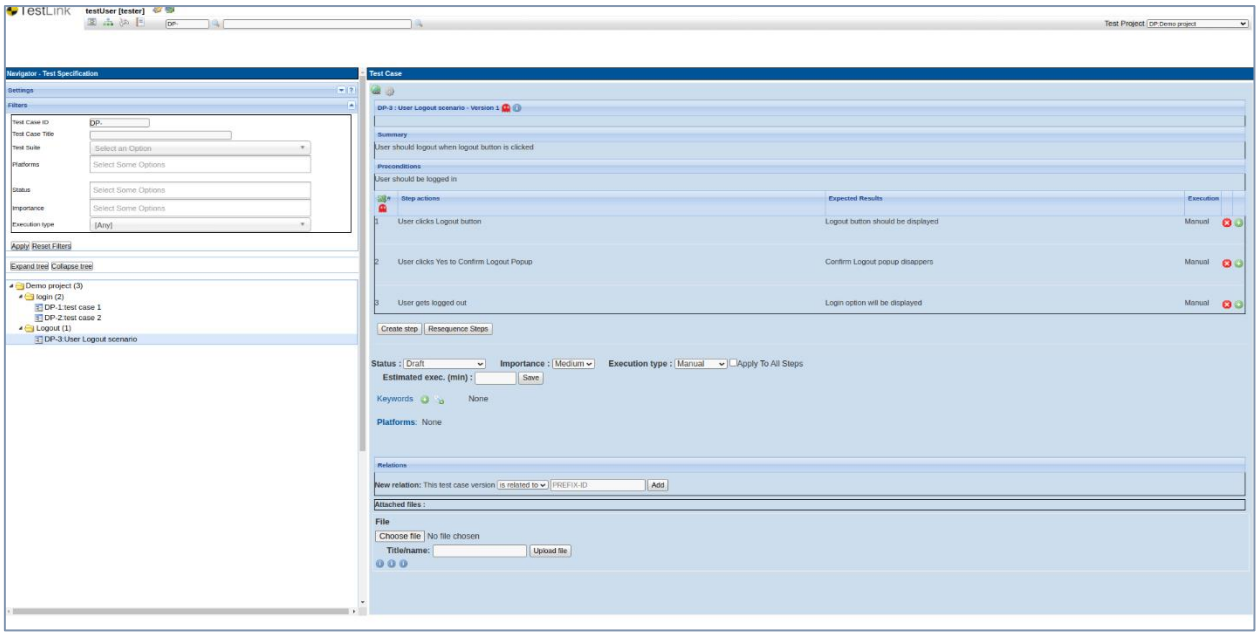

Рисунок 1.5 – Система TestLink

Роблячи висновок аналізу аналогів систем тест-кейсів, зробимо відповідну порівняльну таблицю (див. таб. 1.2).

Таблиця 1.2 – Порівняльна характеристика аналогів систем керування тестуванням

№	Система	Платформа	Ліцензія	Гнучкість	Основні задачі
1	TestRail	Веб (хмара / локальна)	Платна	Обмежена кастомізація	Створення, організація, запуск тест-кейсів, аналітика
2	TestLink	Веб (локальна)	Безкоштовна	Відкрите налаштування	Ведення тестової документації, планування, перевірка, мінімальна інтеграція

Підсумувавши результати проведеного аналізу, можна зробити такі висновки, що розроблювана система в даній кваліфікаційній роботі не є конку-

рентною. Це пояснюється тим, що, по-перше, вебресурс буде розроблятись на основі побажань викладачів. Необхідно врахувати весь потрібний функціонал для вивчення дисципліни «Якість програмного забезпечення та тестування», а саме: можливість створення профілів студентів та викладачів, можливість створення, перевірки та оцінювання звітів баг-репортів і тест-кейсів. Даний ресурс створюється виключно для внутрішніх потреб університету з метою підвищення автономності його роботи.

Крім того, система не передбачає широкого комерційного використання або інтеграції з іншими освітніми платформами, що також знижує її конкурентоспроможність на загальному ринку програмних продуктів. Основна мета розробки – забезпечення зручного інструменту для навчального процесу, який дозволить викладачам ефективно контролювати засвоєння матеріалу студентами, а студентам – краще розуміти принципи тестування програмного забезпечення.

2 РОЗРОБКА ФУНКЦІОНАЛЬНОЇ СХЕМИ І АЛГОРИТМУ

2.1 Діаграма варіантів використання системи

Для того щоб продемонструвати, можливості використання вебсайту користувачами було вирішено створити діаграму варіантів використання. Use Case Diagram або діаграма варіантів використання системи призначена для моделювання подій, які буде виконувати програма. Це взаємодія між акторами та функціями які будуть використовуватись в програмі. Дана діаграма особливо актуальна на ранніх стадіях проекту, так як вона дозволяє побудувати майже повне уявлення про проект, навіть якщо він ще не створений [7].

Діаграма використання особливо допоможе для проведення тестової частини проекту, так як за її схемою можна програти різні сценарії.

В даній діаграмі використано такі актори:

– Адміністратор – за його допомогою будуть контролюватись ролі користувачів, такі як викладач та студент. Саме тільки адміністратор може зареєструвати викладача.

– Викладач – дана роль дозволить реєструвати студентів, видаляти їх та редагувати. Також роль викладача передбачає перевірку робіт студентів, перегляд та можливість залишити коментар. Також дана роль має можливість оцінювати роботи.

– Студент – дана роль дозволить заходити за логіном та паролем який наддасть викладач. Студент буде мати можливість створювати тест-кейси та баг-репорти. Студент буде отримувати коментарі від викладача, а також буде мати можливість переглядати результати виконання робіт.

Розглянемо дану діаграму на рисунку 2.1.

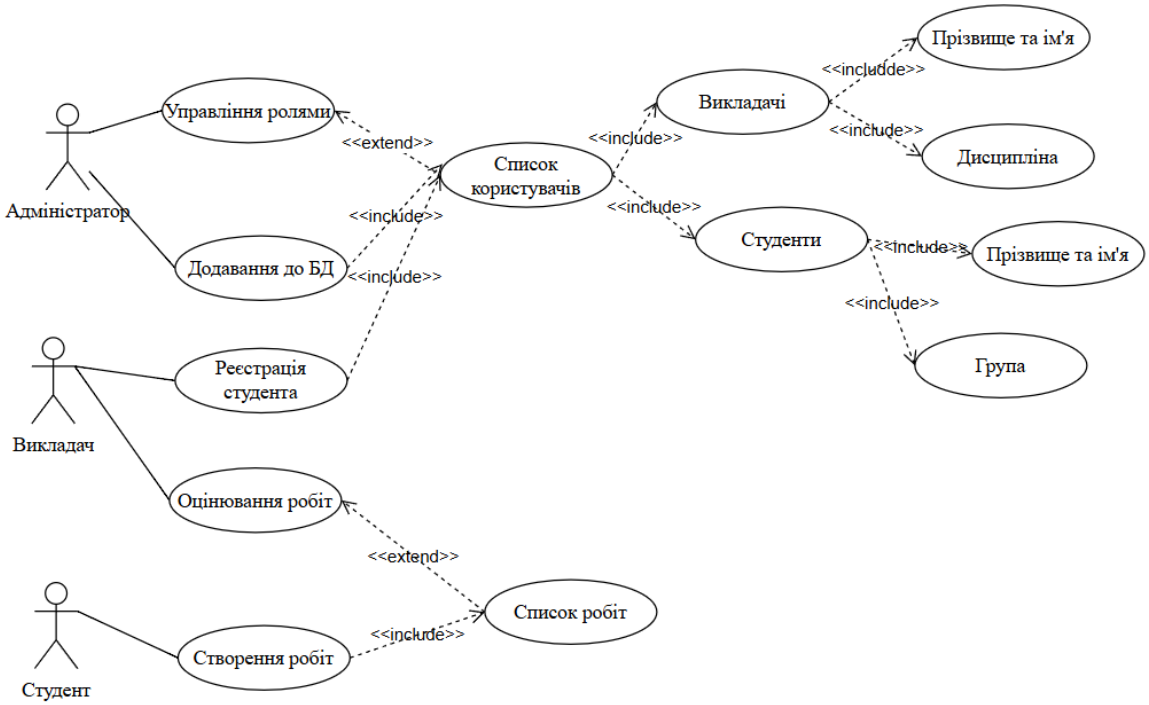

Рисунок 2.1 – Діаграма використання системи

2.2 Функціональна схема програми

Щоб детальніше проаналізувати процеси та, які відбуваються на кожному етапі вебсайту було створено функціональну діаграму. Дана схема дозволить наочно переглянути всю структуру вебзастосунку для фіксації результатів тестування програмного забезпечення. Вона показує основні компоненти системи та зв’язки між ними [8].

Розглянемо перший рівень даної функціональної схеми, який продемонстровано на рисунку 2.2.

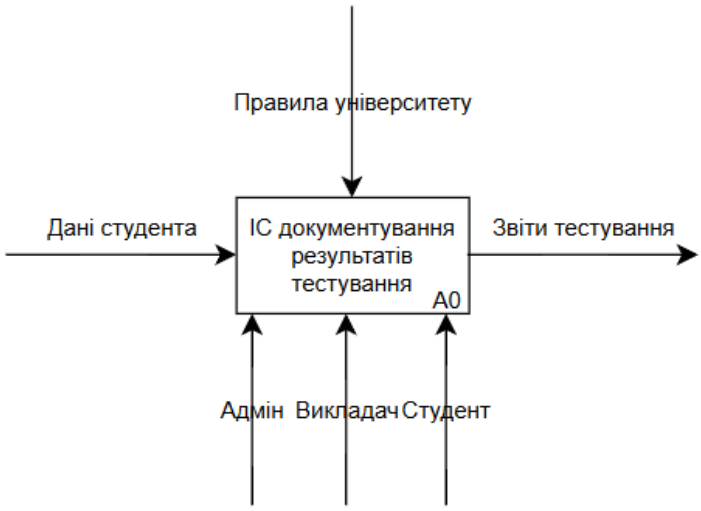

Рисунок 2.2 – Контекстна діаграма A0

На даній діаграмі показано такі елементи:

– Вхідні дані – дані студентів, а саме їх особиста інформація (Прізвище, ім’я та створені ними звіти: тест-кейси або баг-репорти)

– Керування – правила університету, а саме силабусу з відповідної дисципліни.

– Користувачами даної системи є: адміністратор (контролює та редагує усю інформацію пов’язану з базою даних), викладач (додає нових студентів, читає, аналізує, перевіряє, коментує та оцінюю роботи) та студент (створює роботи за завданням викладача).

– Вихідними даними можна вважати саме створені та оцінені тест-кейси або баг-репорти.

Далі розглянемо саме функціональну діаграму даної системи на рисунку 3.3.

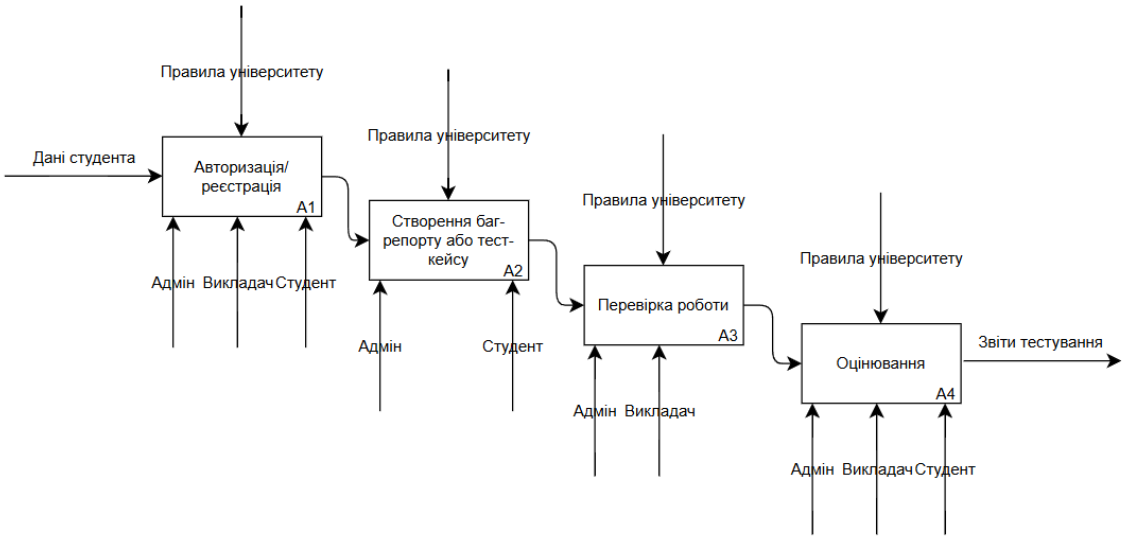

Рисунок 2.3 – Функціональна діаграма

На даній діаграмі продемонстровано основні функції системи.

Авторизація/реєстрація – на даному етапі студент авторизується в системі, або викладач його реєструє в системі.

Створення баг-репорту або тест-кейсу – в цій частині програми студент виконує завдання, яке йому надав викладач.

Перевірка роботи – викладач перевіряє заповнені роботи студентів, залишає коментарі на доопрацювання або переходить до наступного етапу.

Оцінювання – фінальний етап, в якому викладач оцінює роботу студента та отримує готовий звіт по завданню з тестування програмного забезпечення.

2.3 Граф діалогу інтерфейсу

Для створення інтерфейсу вебсайту та його кращого аналізу побудуємо граф діалогу інтерфейсу. В даній діаграмі показано процеси, в яких відбуваються дії користувача та перехід між вікнами [9]. Дана діаграма показує логіку дій користувача та інтерфейсу (див. рис. 2.4).

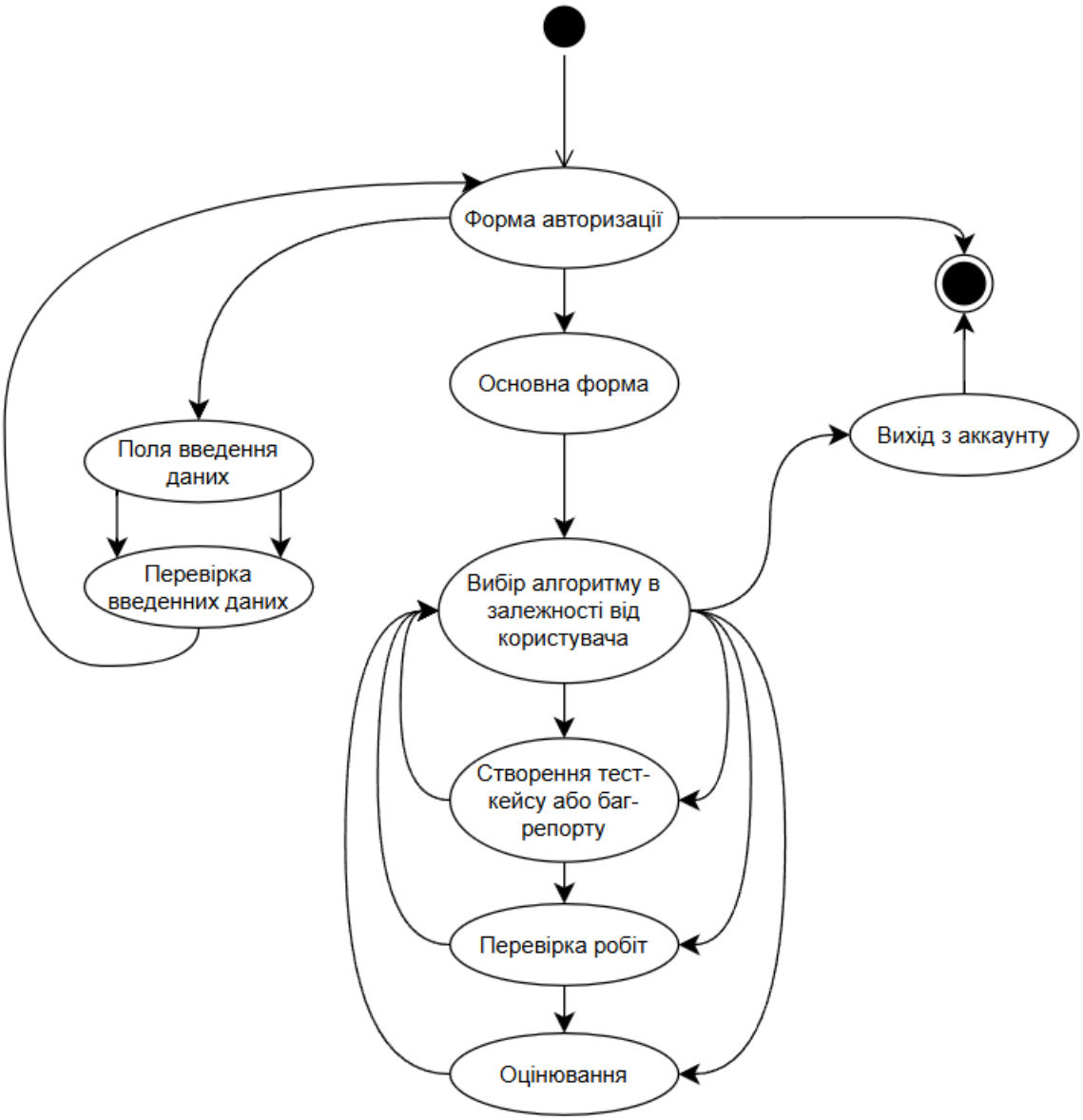

Рисунок 2.4 – Граф діалогу інтерфейсу

У даній діаграмі представлені основні сценарії використання: форма авторизації, вибір алгоритму в залежності від користувача (заповнення тест-кейсів/баг-репортів або перевірка та оцінювання робіт). Дана діаграма ілюструє перехід між екранами в залежності від дій користувачів, а саме: натискання кнопок, заповнення форм та збереження даних.

Кожний блок діаграми є вершиною графу, який представляє собою сторінку веб інтерфейсу, а дуги – логічні переходи між даними вікнами, які виникають через дії користувача.

2.4 Алгоритм створення, обробки та перевірки тестових звітів

Для розробки алгоритму повного циклу роботи з тестовими звітами, а саме відповідні функції: створення, збереження, перегляд, перевірка та оцінювання, важливо забезпечити чітку послідовність дій та виконання дій двох основних профілів – студента та викладача [10]. Тому, за допомогою діаграми діяльності (див. рис. 2.5) розглянемо основні дії кожного користувача.

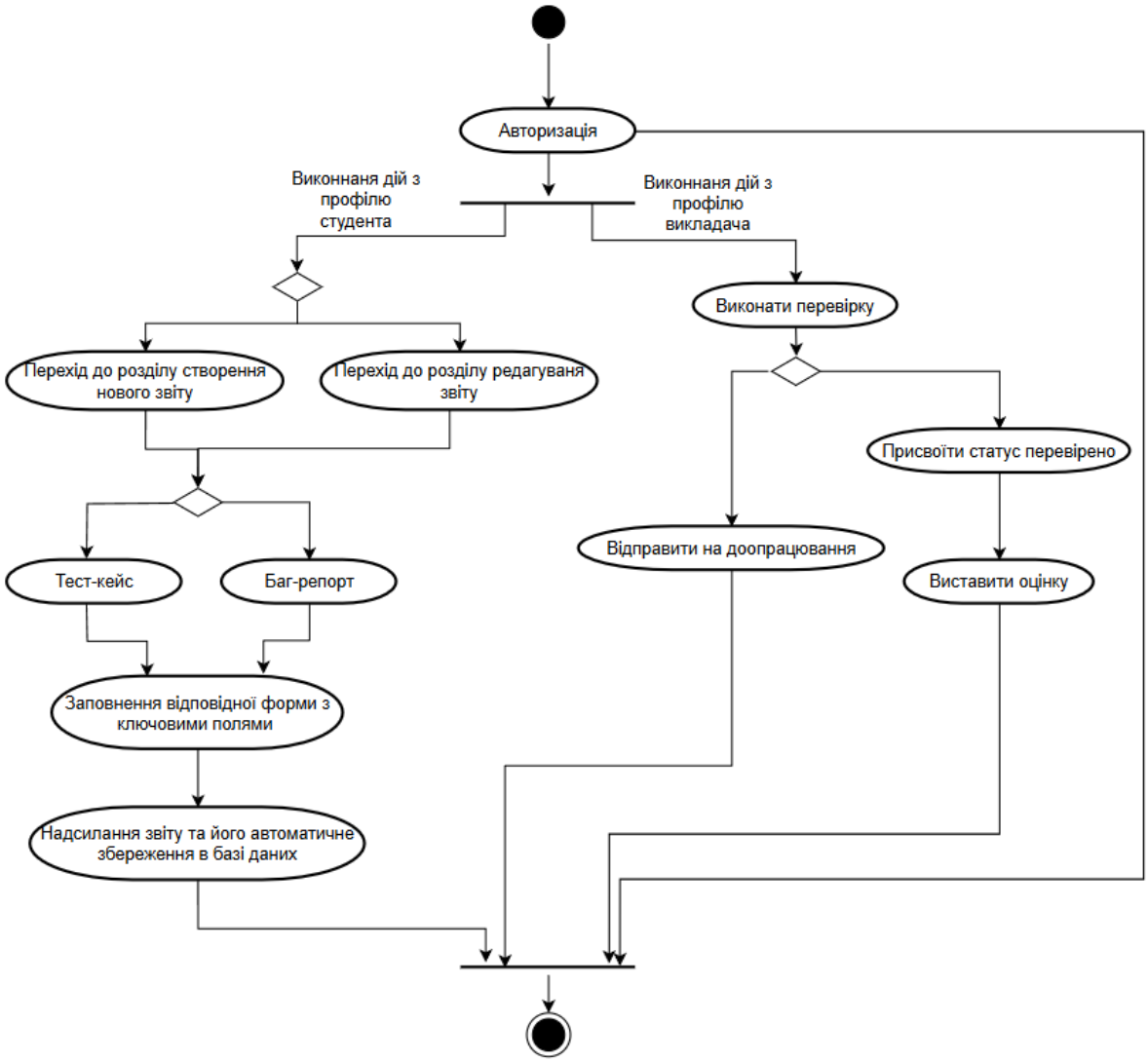

Рисунок 2.5 – Діаграма діяльності

З боку студента – можливість створювати звіти баг-репортів або тест-кейсів, а з боку викладача – це можливість перегляду, коментування та перевірка звітів.

Таким чином, можна розглянути, що дії користувачів активно взаємодіють між собою у процесі навчання тестування програмного забезпечення. Завдяки чітко структурованій логіці дій користувачів система дозволяє автоматизувати створення, обробку, перевірку та оцінювання тестових звітів.

2.5 Структура бази даних

ER-діаграма – це діаграма сутність-зв’язок, яка демонструє логічну модель бази даних системи документування результатів тестування програмного забезпечення [11]. Вона складається з п’яти основних сутностей: «Викладачі», «Студенти», «Завдання», «Баг-репорти» та «Тест-кейси», які пов’язані між собою через логічні зв’язки.

Сутність «Викладачі» містить основні атрибути, необхідні для авторизації та ідентифікації викладача в системі. Її унікальний ідентифікатор (ID) використовується як зовнішній ключ в інших таблицях.

Сутність «Студенти» включає інформацію про зареєстрованих студентів, зокрема дані авторизації та групову приналежність. Кожен студент має унікальний ідентифікатор, який використовується у зв’язках із баг-репортами та тест-кейсами.

Сутність «Завдання» створюється викладачами і асоціюється з ними через атрибут «Автор викладач», що дозволяє відстежувати, хто поставив завдання. Завдання є спільною основою для створення баг-репортів і тест-кейсів.

Сутність «Баг-репорти» описує знайдені студентами дефекти під час виконання завдань. Вона зберігає не тільки опис і параметри багів (серйозність, пріоритет, кроки відтворення тощо), але й вказує, хто створив звіт, для якого викладача і з яким завданням він пов'язаний.

Сутність «Тест-кейси» реалізує документацію перевірки функціоналу, містить очікувані результати та дані, що дозволяють відтворити тест. Як і баг-репорти, вона пов’язана із студентом, викладачем і конкретним завданням.

Сутність «Кроки тест-кейсу» це допоміжна таблиця, яка зберігає покрокову інформацію для виконання одного тест-кейсу.

Усі зв’язки між сутностями реалізовані через зовнішні ключі, що забезпечує цілісність та узгодженість даних (див. рис. 2.6).

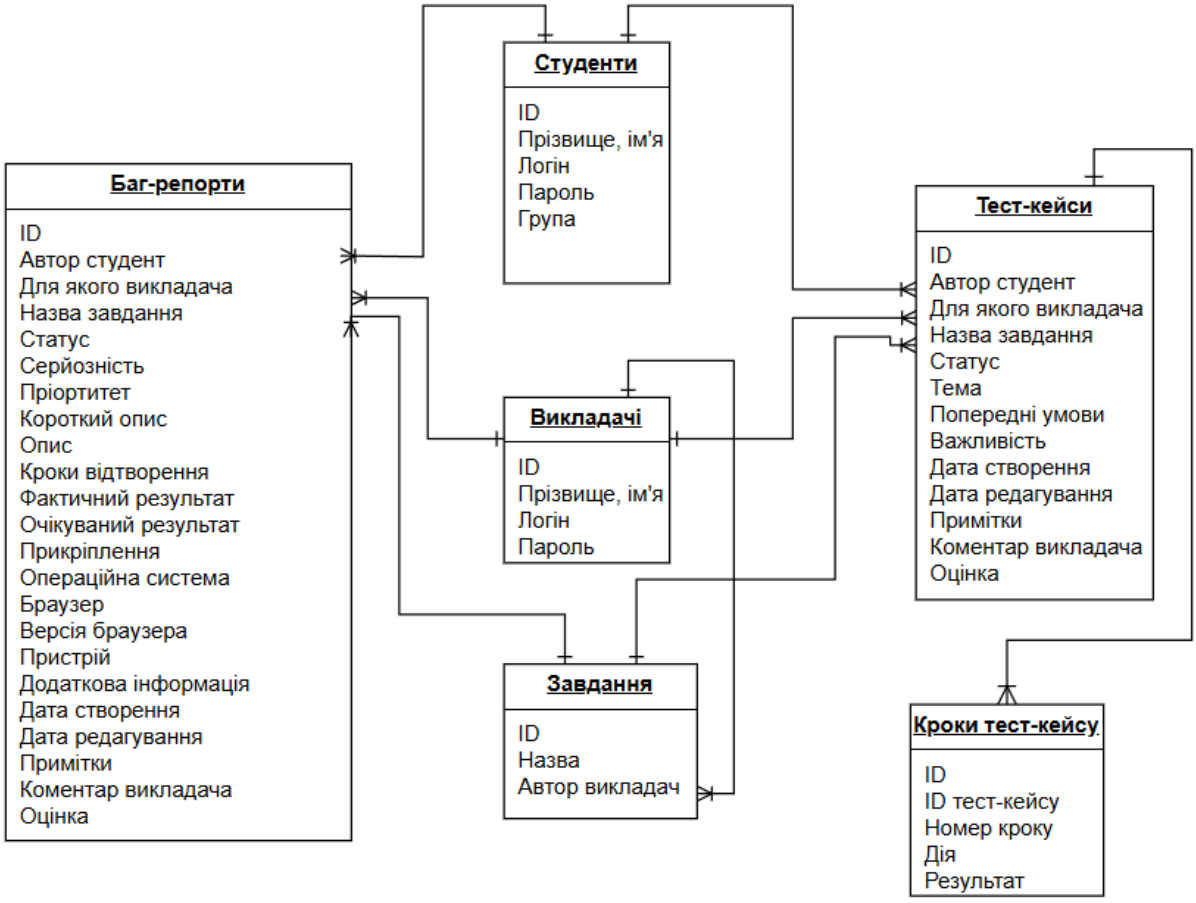

Рисунок 2.6 – ER-діаграма

Усі зв’язки між сутностями – «один до багатьох». Це означає, що кожна основна сутність (викладач, студент, завдання) може мати кілька пов’язаних записів у таблицях баг-репортів, тест-кейсів або кроків тест-кейсів.

Один викладач може створити багато завдань, але кожне завдання належить тільки одному викладачу.

Один викладач може отримувати багато баг-репортів від студентів, але кожен баг-репорт закріплений за одним викладачем.

Один викладач також може перевіряти багато тест-кейсів, але кожен тест-кейс виконується для одного конкретного викладача.

Один студент може створити багато баг-репортів, але кожен баг-репорт пов’язаний лише з одним студентом.

Один студент також може створити багато тест-кейсів, причому кожен тест-кейс належить тільки одному студенту.

Кожне завдання може мати багато пов’язаних баг-репортів, які відображають знайдені помилки у рамках цього завдання.

Також до кожного завдання може бути створено багато тест-кейсів, які перевіряють його коректність або функціональність.

Кожен тест-кейс може складатися з багатьох кроків, які описують покрокову інструкцію тестування.

Зв'язок між тест-кейсом і його кроками реалізовано через таблицю КрокиТестКейсу, яка включає порядковий номер кроку, дію і очікуваний результат.

Усі кроки, пов’язані з тест-кейсом через зовнішній ключ testcase_id, також видаляються автоматично завдяки каскадному видаленню.

3 РОЗРОБКА БАЗИ ДАНИХ І ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз обраної середи програмування

Для реалізації кваліфікаційної роботи, мета якої полягає у розробці застосунку для структурованого документування результатів тестування програмного забезпечення, було обрано комбінацію мов і технологій: HTML, CSS, JavaScript, Python та фреймворк Flask. Розробка ведеться у середовищі Visual Studio Code.

HTML, CSS та JavaScript забезпечують створення інтерфейсу користувача, що є необхідним для зручного введення, відображення та структурованого представлення тестових кейсів і результатів тестування. Ці технології є стандартом для веброзробки і дозволяють реалізувати адаптивний та інтерактивний фронтенд [12].

Python обрано як основну мову програмування через її простоту, широкі можливості та активну підтримку спільнотою. Вона добре підходить для бекенд-логіки, обробки даних і інтеграції з різними бібліотеками [13].

Flask – легкий мікрофреймворк для веб-розробки на Python, який дозволяє швидко створювати вебзастосунки з мінімальним накладним кодом. Flask підтримує роботу зі статичними файлами (HTML, CSS, JS), шаблонами та маршрутизацією, що робить його ідеальним вибором для розробки вебінтерфейсу для системи документування тестування [13].

Visual Studio Code (VS Code) було обрано для розробки кваліфікаційної роботи через низку вагомих переваг, які суттєво впливають на ефективність та якість процесу розробки.

VS Code є легким, швидким і кросплатформеним редактором коду, що працює на Windows, macOS та Linux, що забезпечує гнучкість у виборі операційної системи для розробника. Він має інтуїтивно зрозумілий інтерфейс, що дозволяє швидко почати роботу навіть новачкам, а також підтримує ши-

рокий спектр мов програмування, включаючи HTML, CSS, JavaScript, Python – саме ті, що використовуються у проєкті [14].

3.2 Програмна реалізація основних функцій

3.2.1 Авторизація користувачів

Для забезпечення доступу лише уповноваженим користувачам реалізована система входу з розмежуванням ролей: адміністратор, викладач, студент. Після успішного входу користувач потрапляє на відповідну панель.

На рисунку 3.1 можемо розглянути програмний код входу на панель адміністратора, він має сталий пароль та логін, який прописаний в коді.

```
if role == 'admin':
    if login == 'admin' and password == 'admin':
        session['role'] = 'admin'
        return redirect(url_for('admin_dashboard'))
    else:
        error = 'Невірний логін або пароль адміністратора.'
```

Рисунок 3.1 – Програмний код входу на панель адміністратора

На рисунку 3.2 зображено програмний код входу на сторінку викладача де вказана перевірка даних пароля то логіну в базі даних.

```
elif role == 'teacher':
    with sqlite3.connect('database.db') as con:
        cur = con.cursor()
        cur.execute('SELECT * FROM Викладачі WHERE login=? AND password=?', (login, password))
        user = cur.fetchone()
        if user:
            session['role'] = 'teacher'
            session['user_id'] = user[0]
            return redirect(url_for('teacher_dashboard'))
        else:
            error = 'Невірний логін або пароль викладача.'
```

Рисунок 3.2 – Програмний код входу на панель викладача

На рисунку 3.3 зображено програмний код входу на сторінку студента де вказана перевірка даних пароля то логіну в базі даних.

```
elif role == 'student':
    with sqlite3.connect('database.db') as con:
        cur = con.cursor()
        cur.execute('SELECT * FROM Студенти WHERE login=? AND password=?', (login, password))
        user = cur.fetchone()
        if user:
            session['role'] = 'student'
            session['user_id'] = user[0]
            return redirect(url_for('student_dashboard'))
        else:
            error = 'Невірний логін або пароль студента.'
```

Рисунок 3.3 – Програмний код входу на панель студента

Дані 3 лістинги коду показують як саме відбувається логіка переходу до панелей різних користувачів.

3.2.2 Робота з базою даних викладачів та студентів

Створено реалізацію додавання, редагування та видалення викладача. Всі функції можливо виконати через профіль адміністратора.

Створено функцію додавання та реалізовано відповідно форму. На даному рисунку 3.4 представлено код додавання викладача в базу даних, яка має відповідні поля для запису викладача, а саме: ПІБ (прізвище, ім’я та по-батькові), логін та пароль.

```
# === ДОДАВАННЯ ВИКЛАДАЧА ===
@app.route('/admin/add_teacher', methods=['GET', 'POST'])
def add_teacher():
    if session.get('role') != 'admin':
        return redirect(url_for('login'))
    if request.method == 'POST':
        name = request.form['name']
        login = request.form['login']
        password = request.form['password']
        with sqlite3.connect('database.db') as con:
            cur = con.cursor()
            cur.execute('INSERT INTO Викладачі (name, login, password) VALUES (?, ?, ?)', (name, login, password))
        return redirect(url_for('admin_dashboard'))
    return render_template('add_teacher.html')
```

Рисунок 3.4 – Програмний код додавання викладача

Також додано функціонал редагування та видалення, що дозволяє вносити відповідні зміни до бази даних, пов’язані з інформацією про викладача (див. рис. 3.5).

```python
# === РЕДАГУВАННЯ ВИКЛАДАЧА ===
@app.route('/admin/edit_teacher/<int:teacher_id>', methods=['GET', 'POST'])
def edit_teacher(teacher_id):
    if session.get('role') != 'admin':
        return redirect(url_for('login'))

    with sqlite3.connect('database.db') as con:
        cur = con.cursor()
        if request.method == 'POST':
            name = request.form['name']
            login = request.form['login']
            password = request.form['password']
            cur.execute('UPDATE Викладачі SET name=?, login=?, password=? WHERE id=?',
                        (name, login, password, teacher_id))
            return redirect(url_for('admin_dashboard'))
        else:
            cur.execute('SELECT * FROM Викладачі WHERE id=?', (teacher_id,))
            teacher = cur.fetchone()
            return render_template('edit_teacher.html', teacher=teacher)

# === ВИДАЛЕННЯ ВИКЛАДАЧА ===
@app.route('/admin/delete_teacher/<int:teacher_id>', methods=['POST'])
def admin_delete_teacher(teacher_id):
    if session.get('role') != 'admin':
        return redirect(url_for('login'))

    with sqlite3.connect('database.db') as con:
        cur = con.cursor()

        # Отримуємо всі testcases вчителя
        cur.execute('SELECT id FROM ТестКейси WHERE teacher_id=?', (teacher_id,))
        testcase_ids = [row[0] for row in cur.fetchall()]

        for tc_id in testcase_ids:
            cur.execute('DELETE FROM КрокиТестКейсу WHERE testcase_id=?', (tc_id,))

        cur.execute('DELETE FROM ТестКейси WHERE teacher_id=?', (teacher_id,))
        cur.execute('DELETE FROM Баги WHERE teacher_id=?', (teacher_id,))
        cur.execute('DELETE FROM Завдання WHERE teacher_id=?', (teacher_id,))
        cur.execute('DELETE FROM Викладачі WHERE id=?', (teacher_id,))

    return redirect(url_for('admin_dashboard'))
```

Рисунок 3.5 – Програмний код редагування та видалення викладача

На сторінці адміністратора відображається таблиця з усіма зареєстрованими викладачами, з кнопками для редагування та видалення. Цей функціонал дозволяє адміністратору ефективно керувати викладацьким складом, дотримуючись принципів безпеки та контролю доступу.

Додавання студента відбувається двома способами, через можливість завантаження файлу Excel (див. рис. 3.6) та через форму сайту. Студенти до-

даються у базу даних через форму, яка має поля: прізвище, ім’я, логін, пароль та група.

```
# === ЗАВАНТАЖЕННЯ СТУДЕНТА ===
@app.route('/teacher/upload_students', methods=['GET', 'POST'])
def upload_students():
    if session.get('role') != 'teacher':
        return redirect(url_for('login'))

    if request.method == 'POST':
        file = request.files.get('file')
        if file and allowed_file(file.filename):
            filename = secure_filename(file.filename)
            filepath = os.path.join(app.config['UPLOAD_FOLDER'], filename)
            file.save(filepath)

            try:
                df = pd.read_excel(filepath)

                required_columns = {'name', 'login', 'password', 'group_name'}
                if not required_columns.issubset(set(df.columns)):
                    flash("❌ Excel-файл повинен містити стовпці: name, login, password, group_name", "danger")
                    return redirect(request.url)

                with sqlite3.connect('database.db') as con:
                    cur = con.cursor()
                    for _, row in df.iterrows():
                        try:
                            cur.execute('''
                                INSERT INTO Студенти (name, login, password, group_name)
                                VALUES (?, ?, ?, ?)
                            ''', (row['name'], row['login'], row['password'], row['group_name']))
                        except sqlite3.IntegrityError:
                            flash(f"⚠ Студент з логіном '{row['login']}' вже існує і був пропущений.", "warning")

                    con.commit()
                flash("✅ Студенти успішно додані!", "success")
                return redirect(url_for('teacher_students'))

            except Exception as e:
                flash(f"❌ Помилка обробки файлу: {e}", "danger")
        else:
            flash("❌ Неправильний формат файлу. Завантажуйте лише .xlsx", "danger")

    # GET запит — показати студентів
    with sqlite3.connect('database.db') as con:
        cur = con.cursor()
        cur.execute('SELECT * FROM Студенти')
        students = cur.fetchall()

    return render_template('teacher_students.html', students=students)
```

Рисунок 3.6 – Програмний код додавання студентів через Excel таблицю

На рисунку 3.7 зображено лістинг програми додавання клієнта через форму на сайті. Дані 2 реалізації є доречними тим, що можна додати по одному студенту на платформу через сайт або ж завантажити файл для додавання одразу великої кількості студентів.

```
# === ДОДАВАННЯ СТУДЕНТА ===
@app.route('/teacher/add_student', methods=['GET', 'POST'])
def add_student():
    if session.get('role') != 'teacher':
        return redirect(url_for('login'))
    name = request.form.get('name')
    login = request.form.get('login')
    password = request.form.get('password')
    group_name = request.form.get('group')
    with sqlite3.connect('database.db') as con:
        cur = con.cursor()
        cur.execute('INSERT INTO Студенти (name, login, password, group_name) VALUES (?, ?, ?, ?)', (name, login, password,
    return redirect(url_for('teacher_students'))
```

Рисунок 3.7 – Програмний код додавання студента через форму сайту

Реалізовано редагування інформації про студента. Доступ на редагування має лише адміністратор зі свого профілю. Він може відредагувати всі поля в базі даних студента. Реалізацію програмного коду описано на рисунку 3.8.

```
# === РЕДАГУВАННЯ СТУДЕНТА ===
@app.route('/admin/edit_student/<int:student_id>', methods=['GET', 'POST'])
def edit_student(student_id):
    if session.get('role') != 'admin':
        return redirect(url_for('login'))

    with sqlite3.connect('database.db') as con:
        cur = con.cursor()
        if request.method == 'POST':
            name = request.form['name']
            login = request.form['login']
            password = request.form['password']
            group = request.form['group']
            cur.execute('UPDATE Студенти SET name=?, login=?, password=?, group_name=? WHERE id=?',
                        (name, login, password, group, student_id))
            return redirect(url_for('admin_dashboard'))
        else:
            cur.execute('SELECT * FROM Студенти WHERE id=?', (student_id,))
            student = cur.fetchone()
            return render_template('edit_student.html', student=student)
```

Рисунок 3.8 – Програмний код редагування студента

Також реалізовано видалення студента через панель адміністратора та панель викладача, при видаленні студента відповідно також видаляються всі пов’язані записи в базі даних. Реалізацію описано на рисунку 3.9.

```python
# === ВИДАЛЕННЯ СТУДЕНТА ===
@app.route('/admin/delete_student/<int:student_id>', methods=['POST'])
def admin_delete_student(student_id):
    if session.get('role') != 'admin':
        return redirect(url_for('login'))

    with sqlite3.connect('database.db') as con:
        cur = con.cursor()

        # Отримуємо всі testcases студента
        cur.execute('SELECT id FROM ТестКейси WHERE student_id=?', (student_id,))
        testcase_ids = [row[0] for row in cur.fetchall()]

        # Видаляємо кроки для кожного testcase
        for tc_id in testcase_ids:
            cur.execute('DELETE FROM КрокиТестКейсу WHERE testcase_id=?', (tc_id,))

        # Видаляємо зв'язані записи
        cur.execute('DELETE FROM ТестКейси WHERE student_id=?', (student_id,))
        cur.execute('DELETE FROM Баги WHERE student_id=?', (student_id,))
        cur.execute('DELETE FROM Студенти WHERE id=?', (student_id,))

    return redirect(url_for('admin_dashboard'))
```

Рисунок 3.9 – Програмний код видалення студента

Реалізовано основні модулі для роботи зі студентами та викладачами, а саме: додавання, перегляд, редагування та видалення.

3.2.3 Робота з базами даних тест-кейсів та баг-репортів

У рамках програмної реалізації вебдодатку було впроваджено повноцінну взаємодію з базами даних тест-кейсів і баг-репортів, що дозволяє ефективно управляти процесом тестування програмного забезпечення студентами та перевіркою викладачами.

Організовано реалізацію функцію додавання тест-кейсу через форму студента. На рисунку 3.10 зображено програмний код реалізації.

```python
# === ДОДАВАННЯ ТЕСТ КЕЙСУ ===
@app.route('/add_testcase', methods=['GET', 'POST'])
def add_testcase():
    if 'user_id' not in session or session['role'] != 'student':
        return redirect(url_for('login'))

    conn = sqlite3.connect('database.db')
    cur = conn.cursor()

    cur.execute('SELECT id, name FROM Викладачі')
    teachers = cur.fetchall()
    cur.execute('SELECT id, title FROM Завдання')
    tasks = cur.fetchall()

    if request.method == 'POST':
        student_id = session['user_id']
        teacher_id = request.form['teacher_id']
        task_id = request.form['task_id']
        topic = request.form['topic']
        preconditions = request.form['preconditions']
        importance = request.form['importance']
        status = 'на перевірку'
        notes = ''
        grade = ''
        created_at = updated_at = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

        cur.execute('''
            INSERT INTO ТестКейси (
                student_id, teacher_id, task_id, status, topic, preconditions,
                importance, created_at, updated_at, notes, grade
            ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
        ''', (
            student_id, teacher_id, task_id, status, topic, preconditions,
            importance, created_at, updated_at, notes, grade
        ))

        testcase_id = cur.lastrowid

        # Кроки тест-кейсу
        steps = request.form.getlist('steps[]')
        expected_results = request.form.getlist('expected_results[]')
        for i in range(len(steps)):
            if steps[i].strip():
                cur.execute('''
                    INSERT INTO КрокиТестКейсу (testcase_id, step_number, action, expected_result)
                    VALUES (?, ?, ?, ?)
                ''', (testcase_id, i + 1, steps[i], expected_results[i]))

        conn.commit()
        conn.close()
        flash("Тест-кейс додано!")
        return redirect(url_for('student_dashboard'))

    return render_template('add_testcase.html', teachers=teachers, tasks=tasks)
```

Рисунок 3.10 – Програмний код додавання тест-кейсу

Також реалізовано функцію додавання баг-репорту: з профілю студента відповідні дані вводяться у форму створення баг-репорту (див. рис. 3.11).

```python
@app.route('/add_bug', methods=['GET', 'POST'])
def add_bug():
    if 'user_id' not in session or session['role'] != 'student':
        return redirect(url_for('login'))

    conn = sqlite3.connect('database.db')
    cur = conn.cursor()

    # Дані для селекторів
    cur.execute('SELECT id, name FROM Викладачі')
    teachers = cur.fetchall()
    cur.execute('SELECT id, title FROM Завдання')
    tasks = cur.fetchall()

    if request.method == 'POST':
        student_id = session['user_id']
        teacher_id = request.form['teacher_id']
        task_id = request.form['task_id']
        severity = request.form['severity']
        priority = request.form['priority']
        short_desc = request.form['short_desc']
        description = request.form['description']
        steps = request.form['steps']
        operating_system = request.form.get('os', '')
        browser = request.form.get('browser', '')
        browser_version = request.form.get('browser_version', '')
        device = request.form.get('device', '')
        additional_info = ''  # Можна додати поле у форму за потреби
        notes = ''
        grade = ''
        status = 'на перевірку'
        created_at = updated_at = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

        # Завантаження файла
        attachment = ''
        file = request.files.get('attachment')
        if file and file.filename:
            filename = secure_filename(file.filename)
            upload_path = os.path.join('static/uploads', filename)
            file.save(upload_path)
            attachment = filename

        # Додавання в БД
        cur.execute('''
            INSERT INTO Баги (
                student_id, teacher_id, task_id, status, severity, priority,
                short_desc, description, steps, attachment, operating_system,
                browser, browser_version, device, additional_info,
                created_at, updated_at, notes, grade
            ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
        ''', (
            student_id, teacher_id, task_id, status, severity, priority,
            short_desc, description, steps, attachment, operating_system,
            browser, browser_version, device, additional_info,
            created_at, updated_at, notes, grade
        ))

        conn.commit()
        conn.close()
        flash("Баг-репорт додано!")
        return redirect(url_for('student_dashboard'))

    conn.close()
    return render_template('add_bug.html', teachers=teachers, tasks=tasks)
```

Рисунок 3.11 – Програмний код додавання баг-репорту

Окрім додавання баг-репортів та тест-кейсів, також було організовано функції редагування та видалення з профілю студента.

З профіля викладача маємо функції перевірки баг-репортів та тест-кейсів, а саме: зміну статусу з «на перевірку» на статус «доопрацювати» або «перевірено», додавання коментаря зі списку коментарів або додати свій коментар та також оцінки.

Дані функції перевірки баг-репорту описано на рисунку 3.12.

```
# === ПЕРЕГЛЯД БАГУ(ВИКЛАДАЧ) ===
@app.route('/teacher/view_bug/<int:bug_id>', methods=['GET', 'POST'])
def view_bug_teacher(bug_id):
    if session.get('role') != 'teacher':
        return redirect(url_for('login'))
    with sqlite3.connect('database.db') as con:
        con.row_factory = sqlite3.Row
        cur = con.cursor()
        # Отримати баг
        cur.execute('''
            SELECT Баги.*, Студенти.name AS student_name, Завдання.title AS task_title, Викладачі.name AS teacher_name
            FROM Баги
            JOIN Студенти ON Баги.student_id = Студенти.id
            JOIN Завдання ON Баги.task_id = Завдання.id
            JOIN Викладачі ON Баги.teacher_id = Викладачі.id
            WHERE Баги.id=?
        ''', (bug_id,))
        bug = cur.fetchone()
        if not bug:
            return 'Баг не знайдено', 404
        # Отримати список коментарів
        cur.execute('SELECT * FROM КоментаріБаги')
        comment_options = cur.fetchall()
        # Обробка форми
        if request.method == 'POST':
            new_status = request.form.get('status')
            new_grade = request.form.get('grade')
            new_notes = request.form.get('notes')
            new_comment = request.form.get('comment_teacher')
            cur.execute('''
                UPDATE Баги
                SET status=?, grade=?, notes=?, comment_teacher=?, updated_at=datetime('now')
                WHERE id=?
            ''', (new_status, new_grade, new_notes, new_comment, bug_id))
            con.commit()
            return redirect(url_for('view_bug_teacher', bug_id=bug_id))
    return render_template('view_bug_teacher.html',
                           bug=bug,
                           comment_options=comment_options)
```

Рисунок 3.12 – Програмний код перевірки баг-репорту

Розглянемо реалізацію програмного коду для перевірки та роботи викладача з тест-кейсом (див. рис. 3.13).

```python
# === ПЕРЕГЛЯД ТЕСТ КЕЙСУ (ВИКЛАДАЧ) ===
@app.route('/teacher/view_testcase/<int:testcase_id>', methods=['GET', 'POST'])
def view_testcase_teacher(testcase_id):
    if session.get('role') != 'teacher':
        return redirect(url_for('login'))
    with sqlite3.connect('database.db') as con:
        con.row_factory = sqlite3.Row
        cur = con.cursor()
        # Отримати тест-кейс
        cur.execute('''
            SELECT ТестКейси.*, Студенти.name AS student_name, Завдання.title AS task_title, Викладачі.name AS teacher_name
            FROM ТестКейси
            JOIN Студенти ON ТестКейси.student_id = Студенти.id
            JOIN Завдання ON ТестКейси.task_id = Завдання.id
            JOIN Викладачі ON ТестКейси.teacher_id = Викладачі.id
            WHERE ТестКейси.id=?
        ''', (testcase_id,))
        testcase = cur.fetchone()

        if not testcase:
            return 'Тест-кейс не знайдено', 404
        # Отримати кроки
        cur.execute('SELECT * FROM КрокиТестКейсу WHERE testcase_id=? ORDER BY step_number', (testcase_id,))
        steps = cur.fetchall()

        # Отримати список доступних коментарів
        cur.execute('SELECT * FROM КоментаріТестКейси')
        comment_options = cur.fetchall()

        # Обробка оновлення
        if request.method == 'POST':
            new_status = request.form.get('status')
            new_grade = request.form.get('grade')
            new_notes = request.form.get('notes')
            new_comment = request.form.get('comment_teacher')
            cur.execute('''
                UPDATE ТестКейси
                SET status=?, grade=?, notes=?, comment_teacher=?, updated_at=datetime('now')
                WHERE id=?
            ''', (new_status, new_grade, new_notes, new_comment, testcase_id))
            con.commit()
            return redirect(url_for('view_testcase_teacher', testcase_id=testcase_id))
    return render_template('view_testcase_teacher.html',
                           testcase=testcase,
                           steps=steps,
                           comment_options=comment_options)
```

Рисунок 3.13 – Програмний код перевірки тест-кейсу

Дані функції дозволяють зручно перевіряти роботи виконанні студентами, викладач має можливість переглядати подані студентами матеріали, залишати коментарі та оцінки, здійснювати фільтрацію та контроль за якістю виконання завдань. Така реалізація дозволяє організувати ефективну взаємодію між учасниками освітнього процесу, забезпечити прозорість перевірки та зворотний зв’язок.

3.3 Інструкція з використання

В даному вебзастосунку створено 3 профілі: адміністратор, викладач та студент.

3.3.1 Загальний доступ до системи

Доступ до вебдодатку здійснюється через браузер за локальною адресою, наприклад http://127.0.0.1:5000. При вході користувач має вказати логін і пароль, після чого система автоматично перенаправляє його на відповідну панель згідно з роллю: адміністратор, викладач або студент (див. рис 3.14).

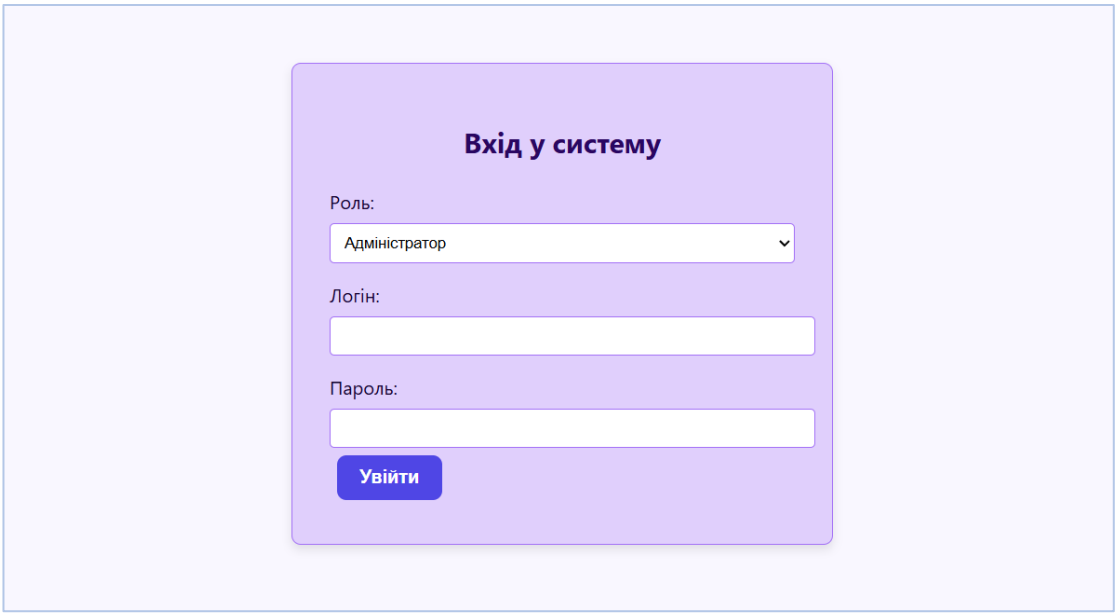

Рисунок 3.14 – Форма входу в систему

Поле «Роль» представлено у вигляді випадаючого списку, де можна обрати одну з трьох опцій: адміністратор, викладач або студент.

3.3.2 Роль адміністратора

Щоб увійти в роль адміністратора потрібно вибрати відповідну роль, а також ввести сталий логін (admin) та пароль (admin). Через дану панель є функції: додати викладача та перегляд списку викладачів та студентів, а також їх видалення або редагування(див. рис. 3.15).

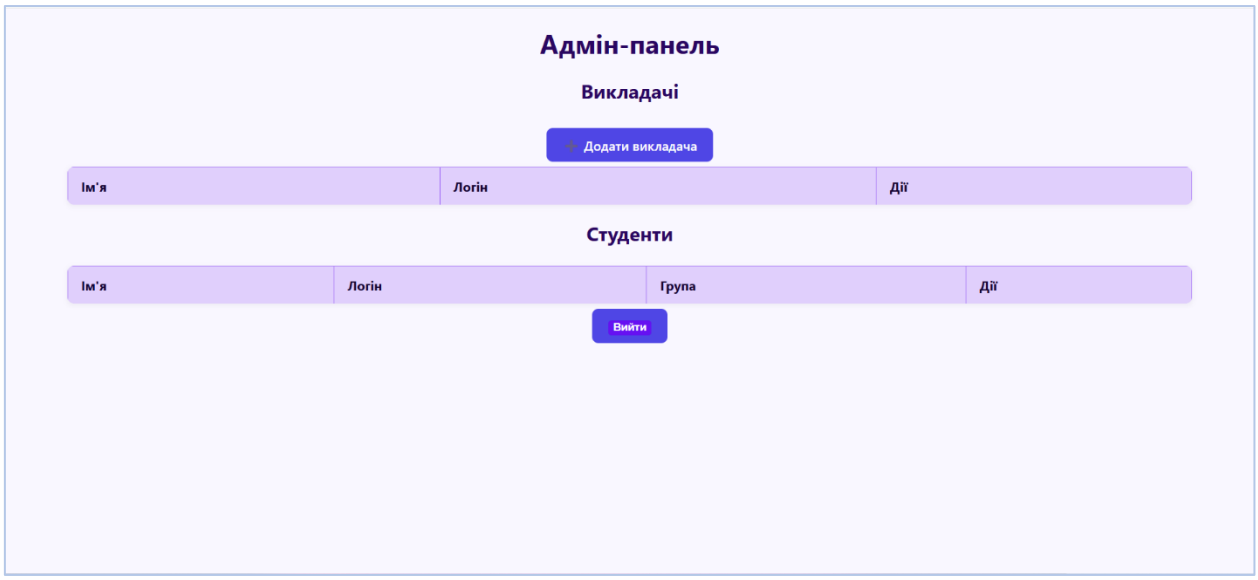

Рисунок 3.15 – Форма адмін-панелі

Щоб додати викладача потрібно заповнити відповідну форму зображено на рисунку 3.16. Потрібно ввести прізвище та ім’я, логін та пароль.

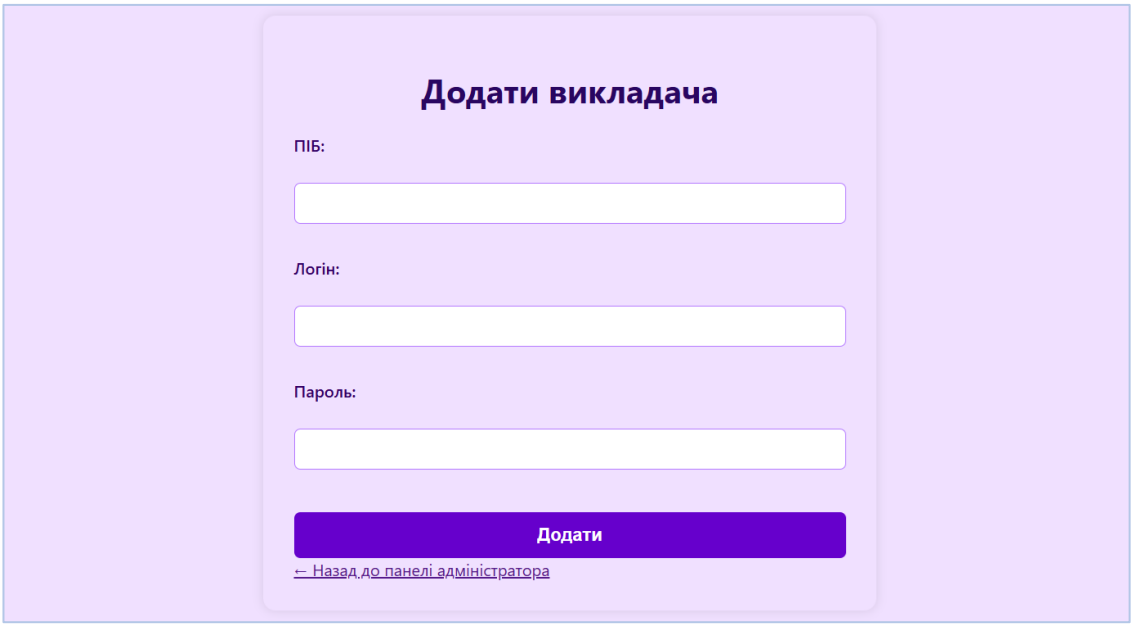

Рисунок 3.16 – Форма додавання викладача

Щоб відредагувати інформацію про викладача потрібно натиснути відповідну кнопку олівця біля інформації викладача, щоб видалити потрібно натиснути на хрестик (див. рис. 3.17).

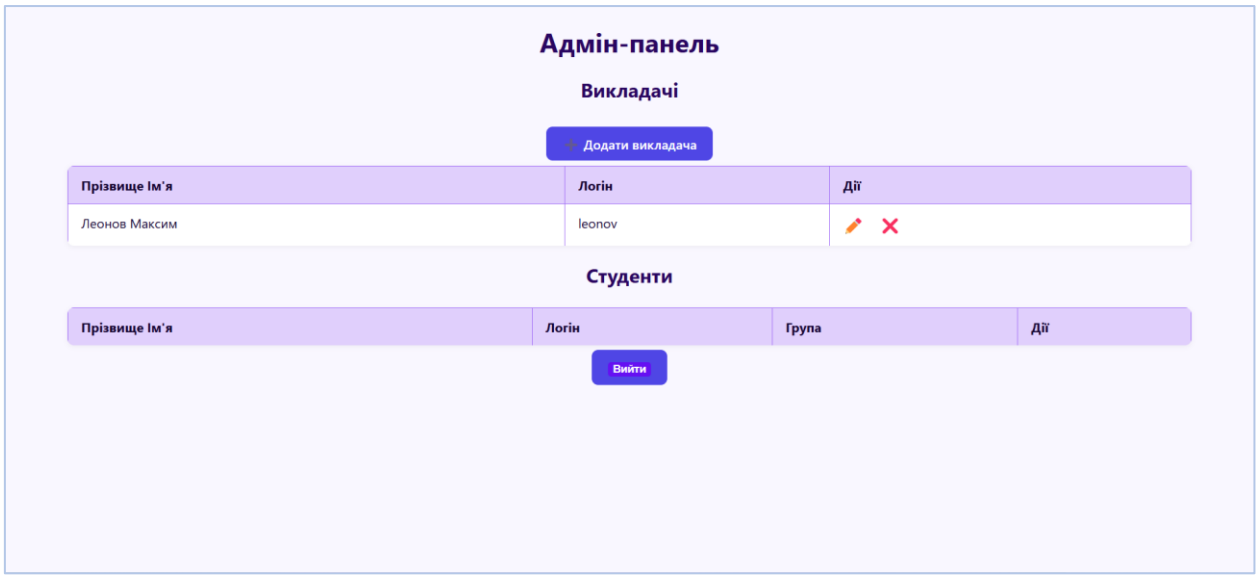

Рисунок 3.17 – Вигляд форми після додавання викладача

При редагуванні маємо відповідні поля, як при додаванні викладача. Не обов’язково вносити зміни в усі поля форми (див. рис. 3.18).

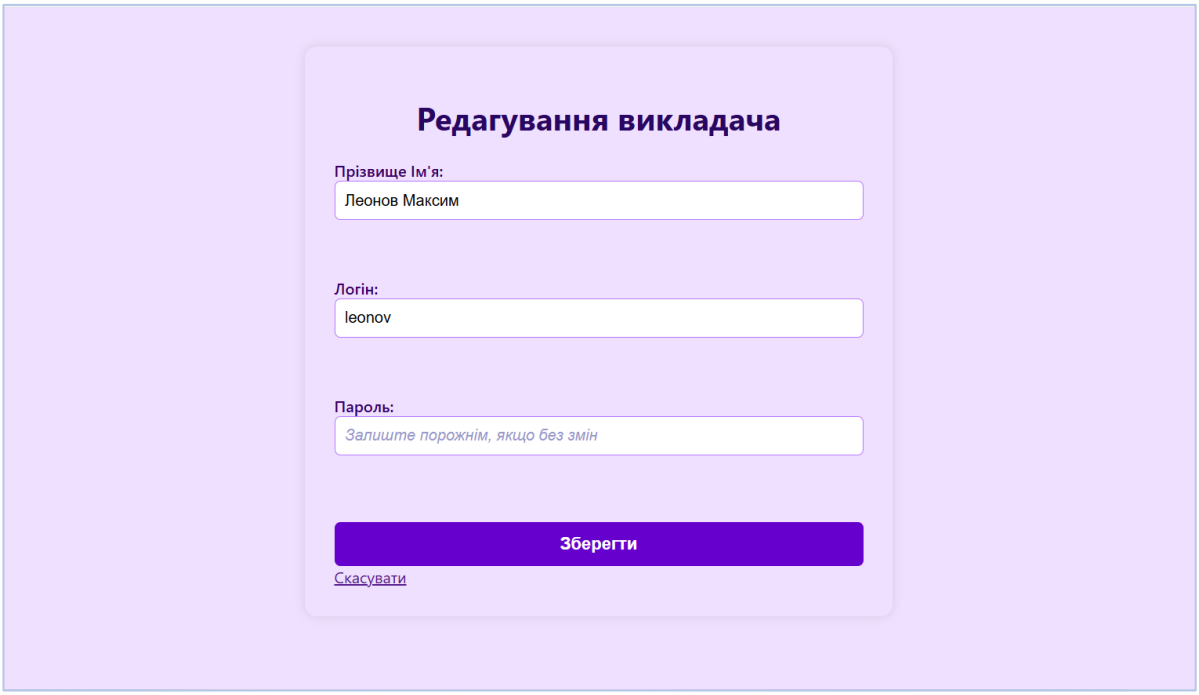

Рисунок 3.18 – Форма редагування викладача

Для редагування студента маємо відповідну форму зображено на рисунку 3.19.

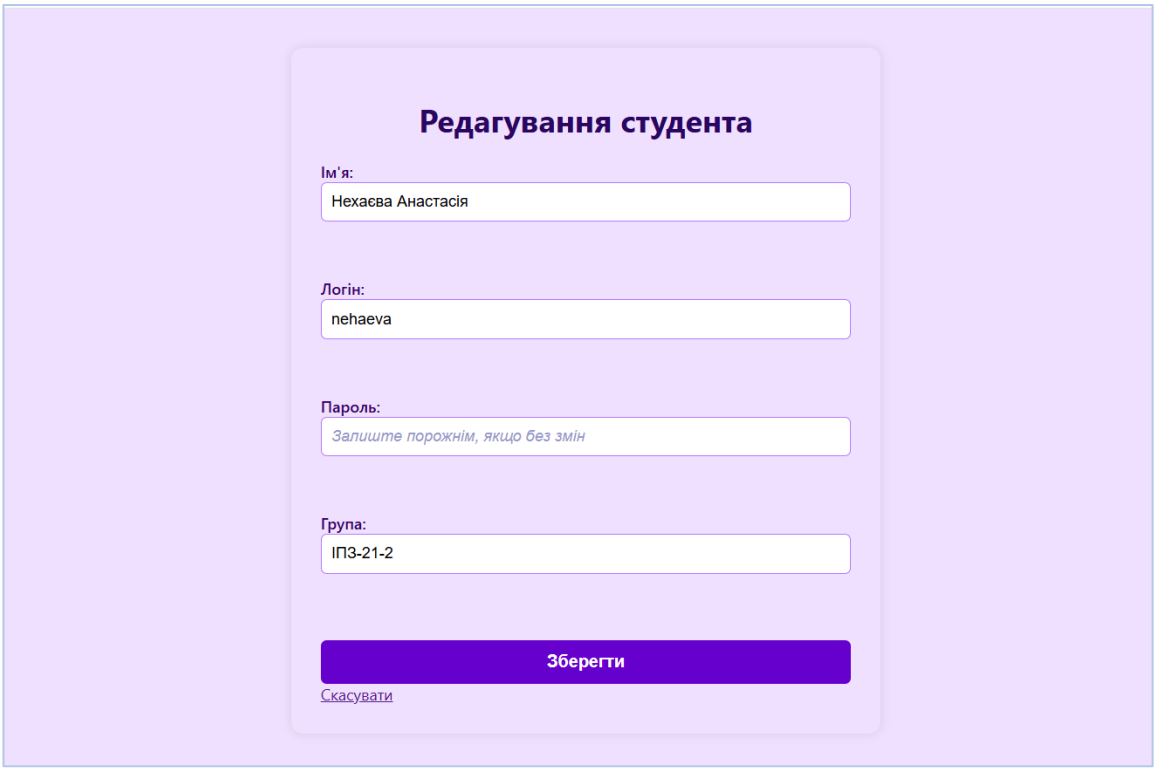

Рисунок 3.19 – Форма редагування студента

3.3.3 Роль викладача

Вхід в панель викладача передбачає через введення логіну та паролю, який створив адміністратор. На рисунку 3.20 зображено вигляд панелі після входу.

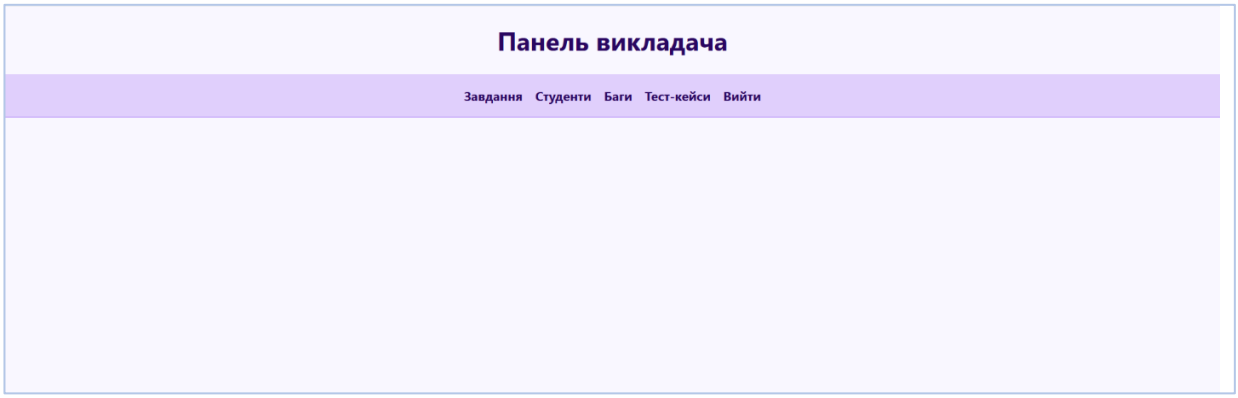

Рисунок 3.20 – Форма панелі викладача

Щоб додати завдання потрібно на відповідній сторінці, зображеній на рисунку 3.21 ввести усі необхідні дані в форму, також завдання можна видалити, натиснувши на іконку хрестика.

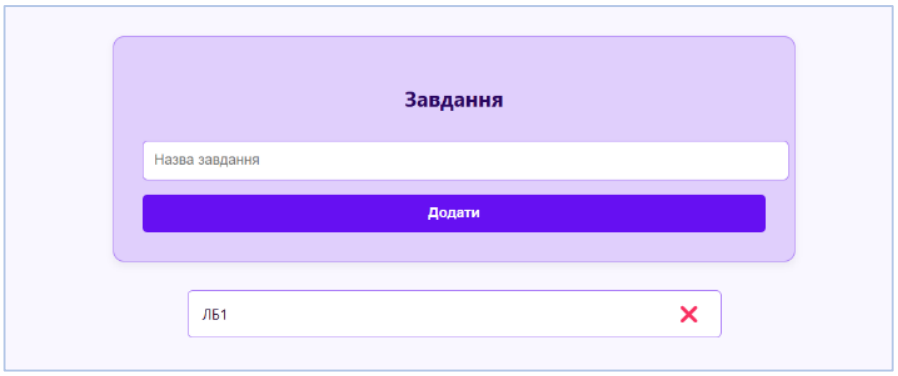

Рисунок 3.21 – Форма для роботи з завданнями

При роботі зі студентами маємо форму (див. рис. 3.22), яка розділена на блоки, які мають такі форми: додавання студента вручну додавання з файлу Excel та перегляд даних з бази даних.

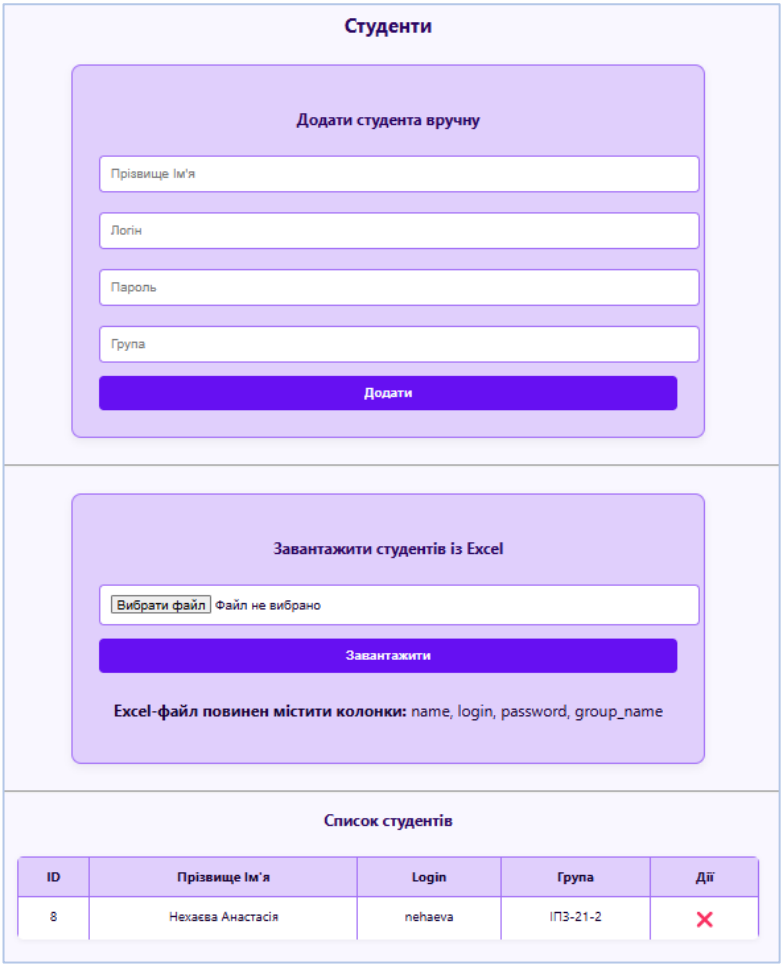

Рисунок 3.22 – Форма для роботи зі студентами

Щоб додати студента потрібно додати або через форму на сайті, або завантажити файл Excel.

Щоб перевірити тест-кейс або баг-репорт, потрібно перейти на відповідну сторінку рисунку 3.23.

Рисунок 3.23 – Форма перегляду списку тест-кейсів

Далі, натиснувши кнопку «Переглянути», відбувається перехід на сторінку для перегляду повної інформації про звіт (див. рис. 3.24).

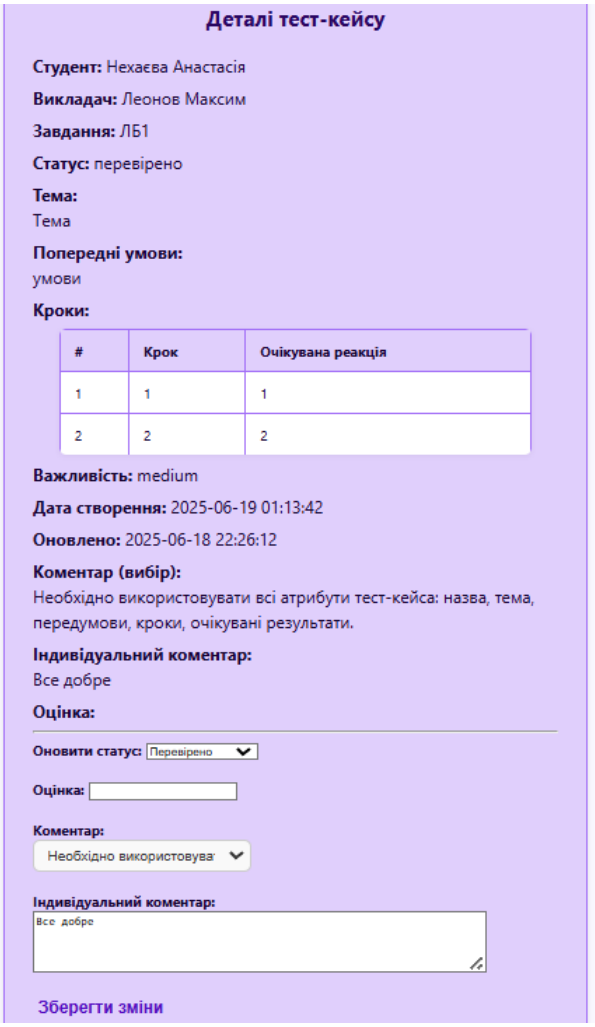

Рисунок 3.24 – Форма деталей тест-кейсу

Після збереження змінюється статус та відповідна оцінка чи коментар. А також змінюються статус.

3.3.4 Роль студента

Вхід в панель студента передбачає через введення логіну та паролю, який створив викладач. На рисунку 3.25 зображено вигляд панелі після входу.

Рисунок 3.25 – Форма панелі студента

Щоб додати баг-репорт або тест-кейс потрібно натиснути відповідну кнопку та заповнити форму редагування (див. рис. 3.26).

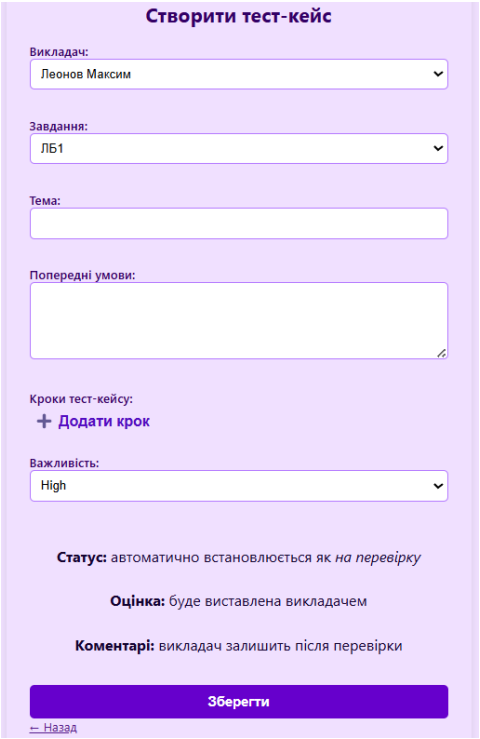

Рисунок 3.26 – Форма створення тест-кейсу

Також, щоб відредагувати звіт тест-кейсу або баг-репорту, потрібно на головній сторінці біля відповідного баг-репорту або тест-кейсу натиснути

кнопку «Переглянути». Буде відкрита відповідна форма для того щоб перейти на форму редагування або видалення, яка зображена на рисунку 3.27.

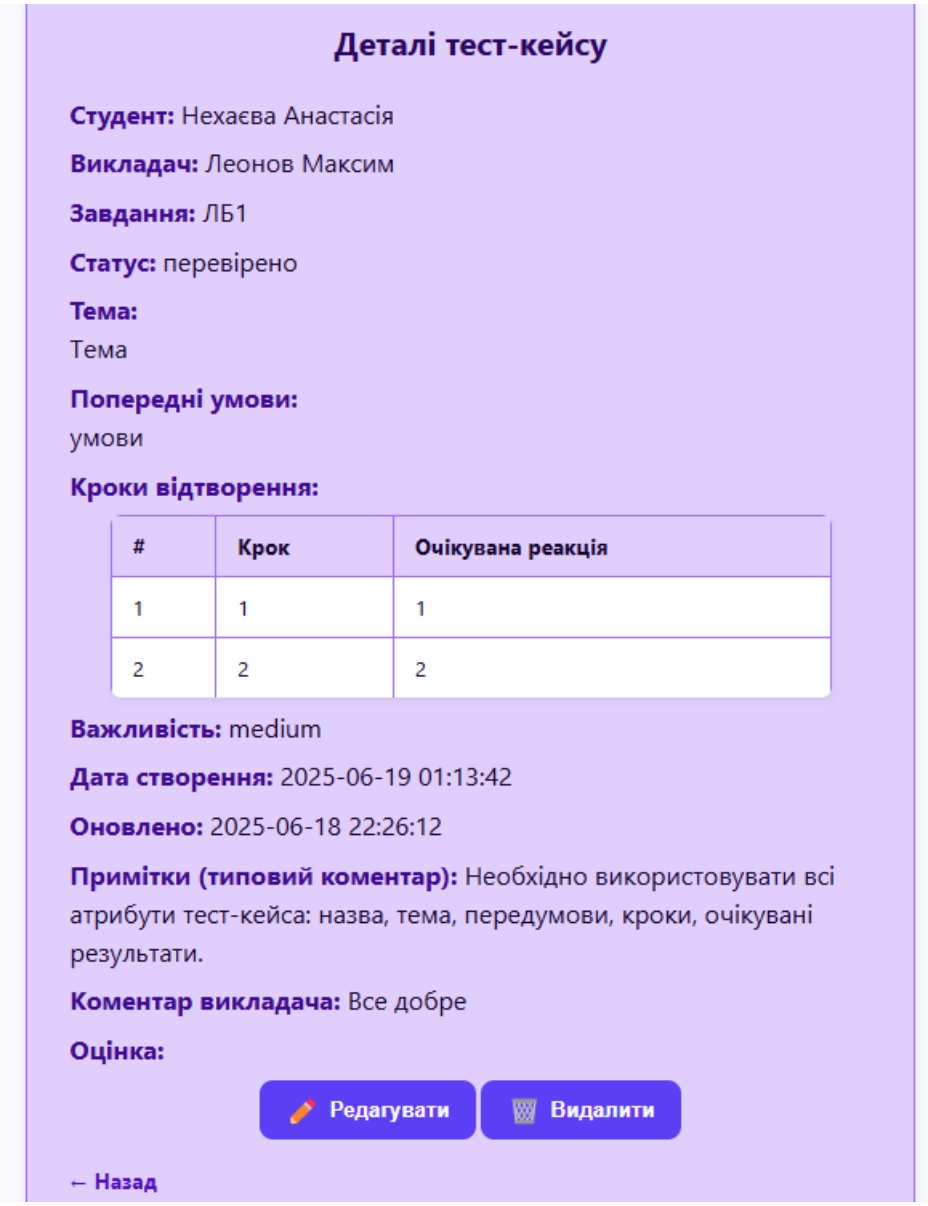

Рисунок 3.27 – Форма перегляду тест-кейсу

Відповідно до форм для роботи з тест-кейсами створено відповідні форми для роботи з баг-репортами.

ВИСНОВОК

У рамках кваліфікаційної роботи було реалізовано вебзастосунок, призначений для підтримки процесу тестування програмного забезпечення у навчальному середовищі. Основною метою проєкту було створення інструменту, який дозволить студентам створювати та редагувати тест-кейси й баг-репорти, а викладачам – перевіряти, коментувати й оцінювати подані звіти.

У процесі реалізації було виконано:

– проєктування архітектури бази даних із урахуванням ролей користувачів;

– розробка інтерфейсу з урахуванням розмежування прав доступу (студент, викладач, адміністратор);

– впровадження функціоналу для роботи з тест-кейсами та баг-репортами: створення, редагування, видалення, коментування та виставлення оцінок;

– реалізацію імпорту студентів з Excel-файлу;

– додано автоматичне оновлення даних на сторінках з використанням AJAX;

– забезпечення зручного адміністрування системи.

Для перевірки коректності та стабільності роботи вебзастосунку було проведено всебічне функціональне тестування. Перевірка охоплювала всі основні сценарії взаємодії користувачів із системою: реєстрацію та автентифікацію з урахуванням ролей, створення й редагування тест-кейсів і баг-репортів, зміну статусів, виставлення оцінок, додавання коментарів, імпорт даних із таблиць Excel, а також одночасну роботу декількох користувачів. Було протестовано коректну обробку помилок при введенні даних та цілісність збереженої інформації в базі даних. У результаті тестування не було виявлено критичних помилок або збоїв. Система продемонструвала стабільну роботу та повну відповідність функціональним вимогам.

Розроблений вебзастосунок може ефективно використовуватися як навчальний інструмент для студентів спеціальності «Інженерія програмного забезпечення» у межах вивчення дисциплін, пов’язаних із тестуванням ПЗ. Його впровадження полегшує організацію навчального процесу, забезпечує зворотний зв’язок між викладачами та студентами, а також сприяє формуванню практичних навичок тестування у майбутніх фахівців.

Отримані результати підтверджують доцільність використання сучасних вебтехнологій для автоматизації освітніх процесів у галузі програмної інженерії, а сам проєкт може слугувати прикладом ефективного поєднання технологічних рішень із методикою професійної підготовки студентів.

ПЕРЕЛІК ПОСИЛАНЬ

1. QALearning. Test Cases та Bug Tracking Systems: Severity & Priority [Електронний ресурс]. – Режим доступу: https://qalearning.com.ua/theory/lectures/material/test-cases-bugtracking-systems-severity-priority/.
2. Atlassian Jira. Software for teams [Електронний ресурс]. – Режим доступу: https://www.atlassian.com/ru/software/jira.
3. JetBrains. Development Tools [Електронний ресурс]. – Режим доступу: https://www.jetbrains.com/.
4. MantisBT – Bug Tracker [Електронний ресурс]. – Режим доступу: https://mantisbt.org/.
5. TestRail – Test Management Tool [Електронний ресурс]. – Режим доступу: https://www.testrail.com/.
6. TestLink – Test Management System [Електронний ресурс]. – Режим доступу: https://testlink.org/.
7. What is a UML Use Case Diagram [Електронний ресурс] // MindOnMap. – Режим доступу: https://www.mindonmap.com/uk/blog/what-is-a-uml-use-case-diagram/.
8. Функціональна схема [Електронний ресурс] // Вікіпедія. – Режим доступу: https://uk.wikipedia.org/wiki/Функціональна_схема
9. Основи побудови функціональних схем [Електронний ресурс] // Studfile. – Режим доступу: https://studfile.net/preview/10261984/page:10/.
10. UML Activity Diagram – Приклади та опис [Електронний ресурс] // Guru99. – Режим доступу: https://www.guru99.com/uk/uml-activity-diagram.html.
11. ER-модель: сутності, зв’язки, атрибути [Електронний ресурс] // BestProg. – Режим доступу: https://www.bestprog.net/uk/2019/01/24/the-concept-of-er-model-the-concept-of-essence-and-communication-attributes-attribute-types-ua/.

12. Автоматизовані системи тестування ПЗ [Електронний ресурс] // Науково-технічна бібліотека КПІ. – Режим доступу: https://ela.kpi.ua/server/api/core/bitstreams/ed9f65b1-0af7-42da-b6a7-c12862080053/content.

13. Інструменти для управління тестуванням [Електронний ресурс] // Наукова бібліотека НаУКМА. – Режим доступу: https://ekmair.ukma.edu.ua/server/api/core/bitstreams/8a2cc4c0-01e6-4052-856e-0376c0083966/content.

14. Visual Studio Code [Електронний ресурс] // Вікіпедія. – Режим доступу: https://uk.wikipedia.org/wiki/Visual_Studio_Code.

Додаток А – Програмний код

```
from flask import Flask, render_template, request, redirect,
url_for, session, flash
import sqlite3
from datetime import datetime
from werkzeug.utils import secure_filename
import os
from flask import request
import pandas as pd

app = Flask(__name__)
app.secret_key = 'your_secret_key'
UPLOAD_FOLDER = 'uploads'
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.',
1)[1].lower() in {'xlsx'}

# === БАЗА ДАНИХ ===
def init_db():
    with sqlite3.connect('database.db') as con:
        cur = con.cursor()
        # Таблиці користувачів
        cur.execute('''CREATE TABLE IF NOT EXISTS Викладачі (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT,
            login TEXT UNIQUE,
            password TEXT
        )''')

        cur.execute('''CREATE TABLE IF NOT EXISTS Студенти (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT,
            login TEXT UNIQUE,
            password TEXT,
            group_name TEXT
        )''')

        cur.execute('''CREATE TABLE IF NOT EXISTS Завдання (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            title TEXT,
            teacher_id INTEGER,
            FOREIGN KEY (teacher_id) REFERENCES Викладачі(id)
        )''')

        cur.execute('''CREATE TABLE IF NOT EXISTS Баги (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            student_id INTEGER,
            teacher_id INTEGER,
            task_id INTEGER,
```

```
            status TEXT,
            severity TEXT,
            priority TEXT,
            short_desc TEXT,
            description TEXT,
            steps TEXT,
            expected_res TEXT,
            actual_res TEXT,
            attachment TEXT,
            operating_system TEXT,
            browser TEXT,
            browser_version TEXT,
            device TEXT,
            additional_info TEXT,
            created_at TEXT,
            updated_at TEXT,
            notes TEXT,
            comment_teacher TEXT,
            grade TEXT,
            FOREIGN KEY (student_id) REFERENCES Студенти(id),
            FOREIGN KEY (teacher_id) REFERENCES Викладачі(id),
            FOREIGN KEY (task_id) REFERENCES Завдання(id)
        )''')

        cur.execute('''CREATE TABLE IF NOT EXISTS ТестКейси (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            student_id INTEGER,
            teacher_id INTEGER,
            task_id INTEGER,
            status TEXT,
            topic TEXT,
            preconditions TEXT,
            importance TEXT,
            created_at TEXT,
            updated_at TEXT,
            notes TEXT,
            comment_teacher TEXT,
            grade TEXT,
            FOREIGN KEY (student_id) REFERENCES Студенти(id),
            FOREIGN KEY (teacher_id) REFERENCES Викладачі(id),
            FOREIGN KEY (task_id) REFERENCES Завдання(id)
        )''')

        cur.execute('''CREATE TABLE IF NOT EXISTS КрокиТестКейсу
(
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            testcase_id INTEGER,
            step_number INTEGER,
            action TEXT,
            expected_result TEXT,
            FOREIGN KEY (testcase_id) REFERENCES ТестКейси(id)
ON DELETE CASCADE
        )''')
```

```
        cur.execute('''CREATE TABLE IF NOT EXISTS КоментаріБаги
(
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT
        )''')

        cur.execute('''CREATE TABLE IF NOT EXISTS КоментаріТест-
Кейси (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT
        )''')

init_db()

# === ГОЛОВНА СТОРІНКА ===
@app.route('/', methods=['GET', 'POST'])
def login():
    error = ''
    if request.method == 'POST':
        login = request.form['login']
        password = request.form['password']
        role = request.form['role']

        if role == 'admin':
            if login == 'admin' and password == 'admin':
                session['role'] = 'admin'
                return redirect(url_for('admin_dashboard'))
            else:
                error = 'Невірний логін або пароль адміністрато-
ра.'
        elif role == 'teacher':
            with sqlite3.connect('database.db') as con:
                cur = con.cursor()
                cur.execute('SELECT * FROM Викладачі WHERE
login=? AND password=?', (login, password))
                user = cur.fetchone()
                if user:
                    session['role'] = 'teacher'
                    session['user_id'] = user[0]
                    return
redirect(url_for('teacher_dashboard'))
                else:
                    error = 'Невірний логін або пароль виклада-
ча.'
        elif role == 'student':
            with sqlite3.connect('database.db') as con:
                cur = con.cursor()
                cur.execute('SELECT * FROM Студенти WHERE
login=? AND password=?', (login, password))
                user = cur.fetchone()
                if user:
                    session['role'] = 'student'
```

```
                    session['user_id'] = user[0]
                    return
redirect(url_for('student_dashboard'))
                else:
                    error = 'Невірний логін або пароль студен-
та.'

    return render_template('login.html', error=error)

# === ФУНКЦІЇ АДМІНА ===
@app.route('/admin')
def admin_dashboard():
    if session.get('role') != 'admin':
        return redirect(url_for('login'))
    with sqlite3.connect('database.db') as con:
        cur = con.cursor()
        students = cur.execute('SELECT * FROM Студен-
ти').fetchall()
        teachers = cur.execute('SELECT * FROM Виклада-
чі').fetchall()
    return render_template('admin_dashboard.html',
students=students, teachers=teachers)
# === РЕДАГУВАННЯ СТУДЕНТА ===
@app.route('/admin/edit_student/<int:student_id>',
methods=['GET', 'POST'])
def edit_student(student_id):
    if session.get('role') != 'admin':
        return redirect(url_for('login'))

    with sqlite3.connect('database.db') as con:
        cur = con.cursor()
        if request.method == 'POST':
            name = request.form['name']
            login = request.form['login']
            password = request.form['password']
            group = request.form['group']
            cur.execute('UPDATE Студенти SET name=?, login=?,
password=?, group_name=? WHERE id=?',
                        (name, login, password, group,
student_id))
            return redirect(url_for('admin_dashboard'))
        else:
            cur.execute('SELECT * FROM Студенти WHERE id=?',
(student_id,))
            student = cur.fetchone()
            return render_template('edit_student.html',
student=student)

# === ВИДАЛЕННЯ СТУДЕНТА ===
@app.route('/admin/delete_student/<int:student_id>',
methods=['POST'])
def admin_delete_student(student_id):
    if session.get('role') != 'admin':
```

```
        return redirect(url_for('login'))

    with sqlite3.connect('database.db') as con:
        cur = con.cursor()

        # Отримуємо всі testcases студента
        cur.execute('SELECT id FROM ТестКейси WHERE
student_id=?', (student_id,))
        testcase_ids = [row[0] for row in cur.fetchall()]

        # Видаляємо кроки для кожного testcase
        for tc_id in testcase_ids:
            cur.execute('DELETE FROM КрокиТестКейсу WHERE
testcase_id=?', (tc_id,))

        # Видаляємо зв’язані записи
        cur.execute('DELETE FROM ТестКейси WHERE student_id=?',
(student_id,))
        cur.execute('DELETE FROM Баги WHERE student_id=?',
(student_id,))
        cur.execute('DELETE FROM Студенти WHERE id=?',
(student_id,))

    return redirect(url_for('admin_dashboard'))

# === РЕДАГУВАННЯ ВИКЛАДАЧА ===
@app.route('/admin/edit_teacher/<int:teacher_id>',
methods=['GET', 'POST'])
def edit_teacher(teacher_id):
    if session.get('role') != 'admin':
        return redirect(url_for('login'))

    with sqlite3.connect('database.db') as con:
        cur = con.cursor()
        if request.method == 'POST':
            name = request.form['name']
            login = request.form['login']
            password = request.form['password']
            cur.execute('UPDATE Викладачі SET name=?, login=?,
password=? WHERE id=?',
                        (name, login, password, teacher_id))
            return redirect(url_for('admin_dashboard'))
        else:
            cur.execute('SELECT * FROM Викладачі WHERE id=?',
(teacher_id,))
            teacher = cur.fetchone()
            return render_template('edit_teacher.html',
teacher=teacher)

# === ВИДАЛЕННЯ ВИКЛАДАЧА ===
@app.route('/admin/delete_teacher/<int:teacher_id>',
methods=['POST'])
def admin_delete_teacher(teacher_id):
```

```
    if session.get('role') != 'admin':
        return redirect(url_for('login'))

    with sqlite3.connect('database.db') as con:
        cur = con.cursor()

        # Отримуємо всі testcases вчителя
        cur.execute('SELECT id FROM ТестКейси WHERE
teacher_id=?', (teacher_id,))
        testcase_ids = [row[0] for row in cur.fetchall()]

        for tc_id in testcase_ids:
            cur.execute('DELETE FROM КрокиТестКейсу WHERE
testcase_id=?', (tc_id,))

        cur.execute('DELETE FROM ТестКейси WHERE teacher_id=?',
(teacher_id,))
        cur.execute('DELETE FROM Баги WHERE teacher_id=?',
(teacher_id,))
        cur.execute('DELETE FROM Завдання WHERE teacher_id=?',
(teacher_id,))
        cur.execute('DELETE FROM Викладачі WHERE id=?',
(teacher_id,))

    return redirect(url_for('admin_dashboard'))

# === ДОДАВАННЯ ВИКЛАДАЧА ===
@app.route('/admin/add_teacher', methods=['GET', 'POST'])
def add_teacher():
    if session.get('role') != 'admin':
        return redirect(url_for('login'))
    if request.method == 'POST':
        name = request.form['name']
        login = request.form['login']
        password = request.form['password']
        with sqlite3.connect('database.db') as con:
            cur = con.cursor()
            cur.execute('INSERT INTO Викладачі (name, login,
password) VALUES (?, ?, ?)', (name, login, password))
        return redirect(url_for('admin_dashboard'))
    return render_template('add_teacher.html')

# === ФУНКЦІЇ ВИКЛАДАЧА ===
@app.route('/teacher', methods=['GET', 'POST'])
def teacher_dashboard():
    if session.get('role') != 'teacher':
        return redirect(url_for('login'))
    user_id = session['user_id']

    # Обробка фільтрів (для багів і тест-кейсів)
    filter_task_id = request.args.get('filter_task_id',
type=int)
```

```
    filter_student_id = request.args.get('filter_student_id',
type=int)
    filter_status = request.args.get('filter_status')  # <- но-
вий фільтр

    with sqlite3.connect('database.db') as con:
        cur = con.cursor()

        # Завдання викладача
        cur.execute('SELECT * FROM Завдання WHERE teacher_id=?',
(user_id,))
        tasks = cur.fetchall()

        # Студенти (усі)
        cur.execute('SELECT * FROM Студенти')
        students = cur.fetchall()

        # Фільтрація багів
        query_bugs = 'SELECT * FROM Баги WHERE teacher_id=?'
        params_bugs = [user_id]
        if filter_task_id:
            query_bugs += ' AND task_id=?'
            params_bugs.append(filter_task_id)
        if filter_student_id:
            query_bugs += ' AND student_id=?'
            params_bugs.append(filter_student_id)
        if filter_status:  # <- додано
            query_bugs += ' AND status=?'
            params_bugs.append(filter_status)

        cur.execute(query_bugs, params_bugs)
        bugs = cur.fetchall()

        # Фільтрація тест-кейсів
        query_tc = 'SELECT * FROM ТестКейси WHERE teacher_id=?'
        params_tc = [user_id]
        if filter_task_id:
            query_tc += ' AND task_id=?'
            params_tc.append(filter_task_id)
        if filter_student_id:
            query_tc += ' AND student_id=?'
            params_tc.append(filter_student_id)
        cur.execute(query_tc, params_tc)
        testcases = cur.fetchall()

    return render_template('teacher_dashboard.html',
                           tasks=tasks,
                           students=students,
                           bugs=bugs,
                           testcases=testcases,
                           filter_task_id=filter_task_id,
                           filter_student_id=filter_student_id,
                           filter_status=filter_status)
```

```
# === ДОДАВАННЯ ЗАВДАННЯ ===
@app.route('/teacher/add_task', methods=['POST'])
def add_task():
    if session.get('role') != 'teacher':
        return redirect(url_for('login'))
    title = request.form.get('title')
    teacher_id = session['user_id']
    with sqlite3.connect('database.db') as con:
        cur = con.cursor()
        cur.execute('INSERT INTO Завдання (title, teacher_id)
VALUES (?, ?)', (title, teacher_id))
    return redirect(url_for('teacher_tasks'))

# === ВИДАЛЕННЯ ЗАВДАННЯ ===
@app.route('/teacher/delete_task/<int:task_id>', metho?-
 s=['POST'])
def delete_task(task_id):
    if session.get('role') != 'teacher':
        return redirect(url_for('login'))
    with sqlite3.connect('database.db') as con:
        cur = con.cursor()
        # Видалити тест кейси і баги, пов'язані із завданням
        cur.execute('DELETE FROM ТестКейси WHERE task_id=?',
(task_id,))
        cur.execute('DELETE FROM Баги WHERE task_id=?',
(task_id,))
        # Видалити завдання
        cur.execute('DELETE FROM Завдання WHERE id=?',
(task_id,))
    return redirect(url_for('teacher_tasks'))

# === ЗАВАНТАЖЕННЯ СТУДЕНТА ===
@app.route('/teacher/upload_students', methods=['GET', 'POST'])
def upload_students():
    if session.get('role') != 'teacher':
        return redirect(url_for('login'))

    if request.method == 'POST':
        file = request.files.get('file')
        if file and allowed_file(file.filename):
            filename = secure_filename(file.filename)
            filepath = os.path.join(app.config['UPLOAD_FOLDER'],
filename)
            file.save(filepath)

            try:
                df = pd.read_excel(filepath)

                required_columns = {'name', 'login', 'password',
'group_name'}
                if not
required_columns.issubset(set(df.columns)):
```

```
                    flash("❌ Excel-файл повинен містити стовп-
ці: name, login, password, group_name", "danger")
                    return redirect(request.url)

                with sqlite3.connect('database.db') as con:
                    cur = con.cursor()
                    for _, row in df.iterrows():
                        try:
                            cur.execute('''
                                INSERT INTO Студенти (name,
login, password, group_name)
                                VALUES (?, ?, ?, ?)
                            ''', (row['name'], row['login'],
row['password'], row['group_name']))
                        except sqlite3.IntegrityError:
                            flash(f"⚠ Студент з логіном
'{row['login']}' вже існує і був пропущений.", "warning")

                    con.commit()
                flash("✅ Студенти успішно додані!", "success")
                return redirect(url_for('teacher_students'))

            except Exception as e:
                flash(f"❌ Помилка обробки файлу: {e}",
"danger")
        else:
            flash("❌ Неправильний формат файлу. Завантажуйте
лише .xlsx", "danger")

    # GET запит — показати студентів
    with sqlite3.connect('database.db') as con:
        cur = con.cursor()
        cur.execute('SELECT * FROM Студенти')
        students = cur.fetchall()

    return render_template('teacher_students.html',
students=students)

# === ДОДАВАННЯ СТУДЕНТА ===
@app.route('/teacher/add_student', methods=['GET', 'POST'])
def add_student():
    if session.get('role') != 'teacher':
        return redirect(url_for('login'))
    name = request.form.get('name')
    login = request.form.get('login')
    password = request.form.get('password')
    group_name = request.form.get('group')
    with sqlite3.connect('database.db') as con:
        cur = con.cursor()
        cur.execute('INSERT INTO Студенти (name, login,
password, group_name) VALUES (?, ?, ?, ?)', (name, login,
password, group_name))
```

```
    return redirect(url_for('teacher_students'))

# === ВИДАЛЕННЯ СТУДЕНТА ===
@app.route('/teacher/delete_student/<int:student_id>',
methods=['POST'])
def delete_student(student_id):
    if session.get('role') != 'teacher':
        return redirect(url_for('login'))
    with sqlite3.connect('database.db') as con:
        cur = con.cursor()
        # Видалити баги і тест кейси студента
        cur.execute('DELETE FROM ТестКейси WHERE student_id=?',
(student_id,))
        cur.execute('DELETE FROM Баги WHERE student_id=?',
(student_id,))
        # Видалити студента
        cur.execute('DELETE FROM Студенти WHERE id=?',
(student_id,))
    return redirect(url_for('teacher_students'))

# === ДОДАВАННЯ КОМЕНТАРЯ БАГУ ===
@app.route('/teacher/bug/<int:bug_id>/add_comment',
methods=['POST'])
def add_bug_comment(bug_id):
    if session.get('role') != 'teacher':
        return redirect(url_for('login'))
    comment = request.form.get('comment')
    with sqlite3.connect('database.db') as con:
        cur = con.cursor()
        cur.execute('UPDATE Баги SET notes = ? WHERE id=?',
(comment, bug_id))
    return redirect(url_for('teacher_bugs'))

# === ДОДАВАННЯ ОЦІНКИ БАГУ ===
@app.route('/teacher/bug/<int:bug_id>/add_grade',
methods=['POST'])
def add_bug_grade(bug_id):
    if session.get('role') != 'teacher':
        return redirect(url_for('login'))
    grade = request.form.get('grade')
    with sqlite3.connect('database.db') as con:
        cur = con.cursor()
        cur.execute('UPDATE Баги SET grade = ? WHERE id=?',
(grade, bug_id))
    return redirect(url_for('teacher_bugs'))

# === ДОДАВАННЯ КОМЕНТАРЯ ТЕСТ КЕЙСУ ===
@app.route('/teacher/testcase/<int:testcase_id>/add_comment',
methods=['POST'])
def add_testcase_comment(testcase_id):
    if session.get('role') != 'teacher':
        return redirect(url_for('login'))
```

```
    comment = request.form.get('comment')
    with sqlite3.connect('database.db') as con:
        cur = con.cursor()
        cur.execute('UPDATE ТестКейси SET notes = ? WHERE id=?',
(comment, testcase_id))
        con.commit()
    return redirect(url_for('teacher_testcase_detail',
testcase_id=testcase_id))

# === ДОДАВАННЯ ОЦІНКИ ТЕСТ КЕЙСУ ===
@app.route('/teacher/testcase/<int:testcase_id>/add_grade',
methods=['POST'])
def add_testcase_grade(testcase_id):
    if session.get('role') != 'teacher':
        return redirect(url_for('login'))
    grade = request.form.get('grade')
    with sqlite3.connect('database.db') as con:
        cur = con.cursor()
        cur.execute('UPDATE ТестКейси SET grade = ? WHERE id=?',
(grade, testcase_id))
    return redirect(url_for('teacher_testcases'))

# === РОБОТА З ЗАВДАННЯМИ ===
@app.route('/teacher/tasks')
def teacher_tasks():
    if session.get('role') != 'teacher':
        return redirect(url_for('login'))
    user_id = session['user_id']
    with sqlite3.connect('database.db') as con:
        cur = con.cursor()
        cur.execute('SELECT * FROM Завдання WHERE teacher_id=?',
(user_id,))
        tasks = cur.fetchall()
    return render_template('teacher_tasks.html', tasks=tasks)

# === РОБОТА ЗІ СТУДЕНТАМИ ===
@app.route('/teacher/students', methods=['GET', 'POST'])
def teacher_students():
    if session.get('role') != 'teacher':
        return redirect(url_for('login'))

    if request.method == 'POST':
        if 'file' in request.files:
            file = request.files.get('file')
            if file and allowed_file(file.filename):
                filename = secure_filename(file.filename)
                filepath =
os.path.join(app.config['UPLOAD_FOLDER'], filename)
                file.save(filepath)

                try:
                    df = pd.read_excel(filepath)
```

```
                    flash(f"ℹ️ Зчитано {len(df)} рядків з
Excel", "info")

                    required = {'name', 'login', 'password',
'group_name'}
                    if not required.issubset(df.columns):
                        flash("❌ Excel має містити колонки
name, login, password, group_name", "danger")
                    else:
                        inserted = 0
                        skipped = 0
                        with sqlite3.connect('database.db') as
con:
                            cur = con.cursor()
                            for _, row in df.iterrows():
                                name = str(row['name']).strip()
                                login =
str(row['login']).strip()
                                password =
str(row['password']).strip()
                                group =
str(row['group_name']).strip()

                                if not all([name, login,
password, group]):
                                    skipped += 1
                                    continue

                                try:
                                    cur.execute(
                                        'INSERT INTO Студенти
(name, login, password, group_name) VALUES (?, ?, ?, ?)',
                                        (name, login, password,
group)
                                    )
                                    inserted += 1
                                except sqlite3.IntegrityError:
                                    skipped += 1
                            con.commit()
                        flash(f"✅ Додано {inserted} студентів,
пропущено {skipped} (логіни дублювались або були порожні)",
"success")
                except Exception as e:
                    flash(f"❌ Помилка при читанні Excel: {e}",
"danger")
            else:
                flash("❌ Будь ласка, завантажте файл формату
.xlsx", "warning")
            return redirect(url_for('teacher_students'))

        # Обробка форми ручного додавання
        name = request.form.get('name')
```

```
        login = request.form.get('login')
        password = request.form.get('password')
        group = request.form.get('group')

        if name and login and password and group:
            try:
                with sqlite3.connect('database.db') as con:
                    cur = con.cursor()
                    cur.execute('INSERT INTO Студенти (name,
login, password, group_name) VALUES (?, ?, ?, ?)',
                                (name.strip(), login.strip(),
password.strip(), group.strip()))
                    con.commit()
                flash("✅ Студента додано", "success")
            except sqlite3.IntegrityError:
                flash("❌ Логін вже існує", "danger")
        else:
            flash("❌ Всі поля мають бути заповнені", "danger")

        return redirect(url_for('teacher_students'))

    # GET: показати список студентів
    with sqlite3.connect('database.db') as con:
        cur = con.cursor()
        cur.execute('SELECT * FROM Студенти')
        students = cur.fetchall()

    return render_template('teacher_students.html',
students=students)

# === РОБОТА З БАГАМИ ===
@app.route('/teacher/bugs')
def teacher_bugs():
    if session.get('role') != 'teacher':
        return redirect(url_for('login'))
    user_id = session['user_id']

    filter_task_id = request.args.get('filter_task_id',
type=int)
    filter_student_id = request.args.get('filter_student_id',
type=int)
    filter_status = request.args.get('filter_status')

    with sqlite3.connect('database.db') as con:
        cur = con.cursor()

        # Завдання
        cur.execute('SELECT * FROM Завдання WHERE teacher_id=?',
(user_id,))
        tasks = cur.fetchall()

        # Студенти
```

```
        cur.execute('SELECT * FROM Студенти')
        students = cur.fetchall()

        # Баги
        query_bugs = '''SELECT Баги.*, Студенти.name AS
student_name, Завдання.title AS task_title
                        FROM Баги
                        JOIN Студенти ON Баги.student_id = Сту-
денти.id
                        JOIN Завдання ON Баги.task_id = Завдан-
ня.id
                        WHERE Баги.teacher_id=?'''
        params = [user_id]

        if filter_task_id:
            query_bugs += ' AND Баги.task_id=?'
            params.append(filter_task_id)
        if filter_student_id:
            query_bugs += ' AND Баги.student_id=?'
            params.append(filter_student_id)
        if filter_status:
            query_bugs += ' AND Баги.status=?'
            params.append(filter_status)

        cur.execute(query_bugs, params)
        bugs = [dict(zip([column[0] for column in
cur.description], row)) for row in cur.fetchall()]

    return render_template('teacher_bugs.html',
                           bugs=bugs,
                           tasks=tasks,
                           students=students,
                           filter_task_id=filter_task_id,
                           filter_student_id=filter_student_id,
                           filter_status=filter_status)

# === РОБОТА З ТЕСТ КЕЙСАМИ ===
@app.route('/teacher/testcases', methods=['GET'])
def teacher_testcases():
    if session.get('role') != 'teacher':
        return redirect(url_for('login'))
    user_id = session['user_id']

    # Отримання параметрів фільтрації
    filter_task_id = request.args.get('filter_task_id',
type=int)
    filter_student_id = request.args.get('filter_student_id',
type=int)
    filter_status = request.args.get('filter_status')

    with sqlite3.connect('database.db') as con:
        con.row_factory = sqlite3.Row
        cur = con.cursor()
```

```
        # Завдання викладача
        cur.execute('SELECT * FROM Завдання WHERE teacher_id=?',
(user_id,))
        tasks = cur.fetchall()

        # Студенти
        cur.execute('SELECT * FROM Студенти')
        students = cur.fetchall()

        # Базовий запит
        query_tc = '''
            SELECT ТестКейси.*, Студенти.name AS student_name,
Завдання.title AS task_title
            FROM ТестКейси
            JOIN Студенти ON ТестКейси.student_id = Студенти.id
            JOIN Завдання ON ТестКейси.task_id = Завдання.id
            WHERE ТестКейси.teacher_id=?
        '''
        params = [user_id]

        # Додавання фільтрів
        if filter_task_id:
            query_tc += ' AND ТестКейси.task_id=?'
            params.append(filter_task_id)
        if filter_student_id:
            query_tc += ' AND ТестКейси.student_id=?'
            params.append(filter_student_id)
        if filter_status:
            query_tc += ' AND ТестКейси.status=?'
            params.append(filter_status)

        query_tc += ' ORDER BY ТестКейси.id DESC'
        cur.execute(query_tc, params)
        testcases = cur.fetchall()

    return render_template('teacher_testcases.html',
                           tasks=tasks,
                           students=students,
                           testcases=testcases,
                           filter_task_id=filter_task_id,
                           filter_student_id=filter_student_id,
                           filter_status=filter_status)

# === ПЕРЕГЛЯД БАГУ(ВИКЛАДАЧ) ===
@app.route('/teacher/view_bug/<int:bug_id>', methods=['GET',
'POST'])
def view_bug_teacher(bug_id):
    if session.get('role') != 'teacher':
        return redirect(url_for('login'))

    with sqlite3.connect('database.db') as con:
        con.row_factory = sqlite3.Row
```

```
        cur = con.cursor()

        # Отримати баг
        cur.execute('''
            SELECT Баги.*, Студенти.name AS student_name, За-
вдання.title AS task_title, Викладачі.name AS teacher_name
            FROM Баги
            JOIN Студенти ON Баги.student_id = Студенти.id
            JOIN Завдання ON Баги.task_id = Завдання.id
            JOIN Викладачі ON Баги.teacher_id = Викладачі.id
            WHERE Баги.id=?
        ''', (bug_id,))
        bug = cur.fetchone()

        if not bug:
            return 'Баг не знайдено', 404

        # Отримати список коментарів
        cur.execute('SELECT * FROM КоментаріБаги')
        comment_options = cur.fetchall()

        # Обробка форми
        if request.method == 'POST':
            new_status = request.form.get('status')
            new_grade = request.form.get('grade')
            new_notes = request.form.get('notes')
            new_comment = request.form.get('comment_teacher')

            cur.execute('''
                UPDATE Баги
                SET status=?, grade=?, notes=?,
comment_teacher=?, updated_at=datetime('now')
                WHERE id=?
            ''', (new_status, new_grade, new_notes, new_comment,
bug_id))
            con.commit()
            return redirect(url_for('view_bug_teacher',
bug_id=bug_id))

    return render_template('view_bug_teacher.html',
                           bug=bug,
                           comment_options=comment_options)

# === ПЕРЕГЛЯД ТЕСТ КЕЙСУ (ВИКЛАДАЧ) ===
@app.route('/teacher/view_testcase/<int:testcase_id>',
methods=['GET', 'POST'])
def view_testcase_teacher(testcase_id):
    if session.get('role') != 'teacher':
        return redirect(url_for('login'))

    with sqlite3.connect('database.db') as con:
        con.row_factory = sqlite3.Row
        cur = con.cursor()
```

```
        # Отримати тест-кейс
        cur.execute('''
            SELECT ТестКейси.*, Студенти.name AS student_name,
Завдання.title AS task_title, Викладачі.name AS teacher_name
            FROM ТестКейси
            JOIN Студенти ON ТестКейси.student_id = Студенти.id
            JOIN Завдання ON ТестКейси.task_id = Завдання.id
            JOIN Викладачі ON ТестКейси.teacher_id = Виклада-
чі.id
            WHERE ТестКейси.id=?
        ''', (testcase_id,))
        testcase = cur.fetchone()

        if not testcase:
            return 'Тест-кейс не знайдено', 404

        # Отримати кроки
        cur.execute('SELECT * FROM КрокиТестКейсу WHERE
testcase_id=? ORDER BY step_number', (testcase_id,))
        steps = cur.fetchall()

        # Отримати список доступних коментарів
        cur.execute('SELECT * FROM КоментаріТестКейси')
        comment_options = cur.fetchall()

        # Обробка оновлення
        if request.method == 'POST':
            new_status = request.form.get('status')
            new_grade = request.form.get('grade')
            new_notes = request.form.get('notes')
            new_comment = request.form.get('comment_teacher')

            cur.execute('''
                UPDATE ТестКейси
                SET status=?, grade=?, notes=?,
comment_teacher=?, updated_at=datetime('now')
                WHERE id=?
            ''', (new_status, new_grade, new_notes, new_comment,
testcase_id))
            con.commit()
            return redirect(url_for('view_testcase_teacher',
testcase_id=testcase_id))

    return render_template('view_testcase_teacher.html',
                           testcase=testcase,
                           steps=steps,
                           comment_options=comment_options)

# === ФУНКЦІЇ СТУДЕНТА ===
@app.route('/student/dashboard')
def student_dashboard():
    if session.get('role') != 'student':
```

```
        return redirect(url_for('login'))

    student_id = session['user_id']
    with sqlite3.connect('database.db') as con:
        con.row_factory = sqlite3.Row
        cur = con.cursor()

        cur.execute('''
            SELECT ТестКейси.*, Завдання.title AS task_title
            FROM ТестКейси
            JOIN Завдання ON ТестКейси.task_id = Завдання.id
            WHERE ТестКейси.student_id = ?
        ''', (student_id,))
        testcases = cur.fetchall()

        cur.execute('''
            SELECT Баги.*, Завдання.title AS task_title
            FROM Баги
            JOIN Завдання ON Баги.task_id = Завдання.id
            WHERE Баги.student_id = ?
        ''', (student_id,))
        bugreports = cur.fetchall()

    return render_template('student_dashboard.html',
                           testcases=testcases,
                           bugreports=bugreports)

# === ДОДАВАННЯ ТЕСТ КЕЙСУ ===
@app.route('/add_testcase', methods=['GET', 'POST'])
def add_testcase():
    if 'user_id' not in session or session['role'] != 'student':
        return redirect(url_for('login'))

    conn = sqlite3.connect('database.db')
    cur = conn.cursor()

    cur.execute('SELECT id, name FROM Викладачі')
    teachers = cur.fetchall()
    cur.execute('SELECT id, title FROM Завдання')
    tasks = cur.fetchall()

    if request.method == 'POST':
        student_id = session['user_id']
        teacher_id = request.form['teacher_id']
        task_id = request.form['task_id']
        topic = request.form['topic']
        preconditions = request.form['preconditions']
        importance = request.form['importance']
        status = 'на перевірку'
        notes = ''
        grade = ''
        created_at = updated_at = datetime.now().strftime('%Y-
%m-%d %H:%M:%S')
```

```
        cur.execute('''
            INSERT INTO ТестКейси (
                student_id, teacher_id, task_id, status, topic,
preconditions,
                importance, created_at, updated_at, notes, grade
            ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
        ''', (
            student_id, teacher_id, task_id, status, topic,
preconditions,
            importance, created_at, updated_at, notes, grade
        ))

        testcase_id = cur.lastrowid

        # Кроки тест-кейсу
        steps = request.form.getlist('steps[]')
        expected_results =
request.form.getlist('expected_results[]')
        for i in range(len(steps)):
            if steps[i].strip():
                cur.execute('''
                    INSERT INTO КрокиТестКейсу (testcase_id,
step_number, action, expected_result)
                    VALUES (?, ?, ?, ?)
                ''', (testcase_id, i + 1, steps[i],
expected_results[i]))

        conn.commit()
        conn.close()
        flash("Тест-кейс додано!")
        return redirect(url_for('student_dashboard'))

    return render_template('add_testcase.html',
teachers=teachers, tasks=tasks)

# === РЕДАГУВАННЯ ТЕСТ КЕЙСУ ===
@app.route('/student/edit_testcase/<int:testcase_id>',
methods=['GET', 'POST'])
def edit_testcase(testcase_id):
    if session.get('role') != 'student':
        return redirect(url_for('login'))

    with sqlite3.connect('database.db') as con:
        cur = con.cursor()
        cur.execute('SELECT * FROM ТестКейси WHERE id=?',
(testcase_id,))
        testcase = cur.fetchone()

        cur.execute('SELECT ID, title FROM Завдання')
        tasks = cur.fetchall()

        cur.execute('SELECT ID, name FROM Викладачі')
```

```
        teachers = cur.fetchall()

        # Витягнути кроки тест-кейсу (якщо вони зберігаються
окремо)
        cur.execute('SELECT * FROM КрокиТестКейсу WHERE
testcase_id=? ORDER BY step_number', (testcase_id,))
        steps = cur.fetchall()

    if request.method == 'POST':
        teacher_id = request.form['teacher_id']
        task_id = request.form['task_id']
        topic = request.form['topic']
        preconditions = request.form['preconditions']
        importance = request.form['importance']

        # Зібрати кроки тест-кейсу з форми (очікуємо що кроки
мають назви step_action_1, step_expected_1 і т.д.)
        steps_data = []
        i = 1
        while True:
            action_key = f'step_action_{i}'
            expected_key = f'step_expected_{i}'
            if action_key in request.form and expected_key in
request.form:
                action = request.form[action_key].strip()
                expected = request.form[expected_key].strip()
                if action and expected:
                    steps_data.append((testcase_id, i, action,
expected))
                    i += 1
                else:
                    break
            else:
                break

        with sqlite3.connect('database.db') as con:
            cur = con.cursor()
            # Оновлюємо основний тест-кейс
            cur.execute('''
                UPDATE ТестКейси
                SET teacher_id=?, task_id=?, topic=?, preco-
?ditions=?, importance=?, status='на перевірку',
updated_at=datetime('now')
                WHERE id=?
            ''', (teacher_id, task_id, topic, preconditions,
importance, testcase_id))

            # Видаляємо старі кроки
            cur.execute('DELETE FROM КрокиТестКейсу WHERE
testcase_id=?', (testcase_id,))
            # Вставляємо оновлені кроки
            cur.executemany('''
```

```
                INSERT INTO КрокиТестКейсу(testcase_id,
step_number, action, expected_result)
                VALUES (?, ?, ?, ?)
            ''', steps_data)

        return redirect(url_for('student_dashboard'))

    return render_template('edit_testcase.html',
testcase=testcase, tasks=tasks, teachers=teachers, steps=steps)

# === ВИДАЛЕННЯ ТЕСТ КЕЙСУ ===
@app.route('/student/delete_testcase/<int:testcase_id>',
methods=['POST'])
def delete_testcase(testcase_id):
    if session.get('role') != 'student':
        return redirect(url_for('login'))

    with sqlite3.connect('database.db') as con:
        cur = con.cursor()

        # Видалити кроки тест-кейсу
        cur.execute('DELETE FROM КрокиТестКейсу WHERE
testcase_id=?', (testcase_id,))

        # Видалити сам тест-кейс
        cur.execute('DELETE FROM ТестКейси WHERE id=?',
(testcase_id,))

    return redirect(url_for('student_dashboard'))

# === ДОДАВАННЯ БАГУ ===
@app.route('/add_bug', methods=['GET', 'POST'])
def add_bug():
    if 'user_id' not in session or session['role'] != 'student':
        return redirect(url_for('login'))

    conn = sqlite3.connect('database.db')
    cur = conn.cursor()

    # Дані для селекторів
    cur.execute('SELECT id, name FROM Викладачі')
    teachers = cur.fetchall()
    cur.execute('SELECT id, title FROM Завдання')
    tasks = cur.fetchall()

    if request.method == 'POST':
        student_id = session['user_id']
        teacher_id = request.form['teacher_id']
        task_id = request.form['task_id']
        severity = request.form['severity']
        priority = request.form['priority']
        short_desc = request.form['short_desc']
        description = request.form['description']
```

```
        steps = request.form['steps']
        expected_res = request.form['expected_res']
        actual_res = request.form['actual_res']
        operating_system = request.form.get('os', '')
        browser = request.form.get('browser', '')
        browser_version = request.form.get('browser_version',
'')
        device = request.form.get('device', '')
        additional_info = request.form.get('additional_info',
'')
        notes = ''
        grade = ''
        comment_teacher = ''
        status = 'на перевірку'
        created_at = updated_at = datetime.now().strftime('%Y-
%m-%d %H:%M:%S')

        # Завантаження файла
        attachment = ''
        file = request.files.get('attachment')
        if file and file.filename:
            filename = secure_filename(file.filename)
            upload_path = os.path.join('static/uploads',
filename)
            file.save(upload_path)
            attachment = filename

        # Додавання в БД з новими полями
        cur.execute('''
            INSERT INTO Баги (
                student_id, teacher_id, task_id, status,
severity, priority,
                short_desc, description, steps, expected_res,
actual_res,
                attachment, operating_system, browser,
browser_version, device,
                additional_info, created_at, updated_at, notes,
grade, comment_teacher
            ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
?, ?, ?, ?, ?, ?, ?, ?)
        ''', (
            student_id, teacher_id, task_id, status, severity,
priority,
            short_desc, description, steps, expected_res,
actual_res,
            attachment, operating_system, browser,
browser_version, device,
            additional_info, created_at, updated_at, notes,
grade, comment_teacher
        ))

        conn.commit()
        conn.close()
```

```
            flash("✅ Баг-репорт успішно додано!")
            return redirect(url_for('student_dashboard'))

    conn.close()
    return render_template('add_bug.html', teachers=teachers,
tasks=tasks)

# === РЕДАГУВАННЯ БАГУ ===
@app.route('/student/edit_bug/<int:bug_id>', methods=['GET',
'POST'])
def edit_bug(bug_id):
    if session.get('role') != 'student':
        return redirect(url_for('login'))

    with sqlite3.connect('database.db') as con:
        cur = con.cursor()
        cur.execute('SELECT * FROM Баги WHERE id=?', (bug_id,))
        bug = cur.fetchone()

        cur.execute('SELECT ID, title FROM Завдання')
        tasks = cur.fetchall()

        cur.execute('SELECT ID, name FROM Викладачі')
        teachers = cur.fetchall()

    if request.method == 'POST':
        teacher_id = request.form['teacher_id']
        task_id = request.form['task_id']
        severity = request.form['severity']
        priority = request.form['priority']
        short_desc = request.form['short_desc']
        description = request.form['description']
        steps = request.form['steps']
        expected_res = request.form['expected_res']
        actual_res = request.form['actual_res']
        osys = request.form['os']
        browser = request.form['browser']
        browser_version = request.form['browser_version']
        device = request.form['device']

        file = request.files.get('attachment')
        filename = bug[9]  # поточне ім'я файлу
        if file and file.filename != '':
            filename = secure_filename(file.filename)
            filepath = os.path.join('static/uploads', filename)
            file.save(filepath)

        with sqlite3.connect('database.db') as con:
            cur = con.cursor()
            cur.execute('''
                UPDATE Баги
                SET teacher_id=?, task_id=?, severity=?,
priority=?, short_desc=?, description=?, steps=?,
```

```
                    expected_res=?, actual_res=?, attachment=?,
operating_system=?, browser=?,
                    browser_version=?, device=?, status='на пе-
ревірку', updated_at=datetime('now')
                WHERE id=?
            ''', (teacher_id, task_id, severity, priority,
short_desc, description, steps,
                  expected_res, actual_res, filename, osys,
browser,
                  browser_version, device, bug_id))

        return redirect(url_for('student_dashboard'))

    return render_template('edit_bug.html', bug=bug,
tasks=tasks, teachers=teachers)

# === ВИДАЛЕННЯ БАГУ ===
@app.route('/student/delete_bug/<int:bug_id>', methods=['POST'])
def delete_bug(bug_id):
    if session.get('role') != 'student':
        return redirect(url_for('login'))

    student_id = session.get('user_id')  # або інше поле, яке
зберігає ID студента

    with sqlite3.connect('database.db') as con:
        cur = con.cursor()

        # Перевірка приналежності багу студенту
        cur.execute('SELECT student_id FROM Баги WHERE id=?',
(bug_id,))
        row = cur.fetchone()
        if not row or row[0] != student_id:
            return redirect(url_for('student_dashboard'))  # до-
ступ заборонено

        # Видалення багу
        cur.execute('DELETE FROM Баги WHERE id=?', (bug_id,))

    return redirect(url_for('student_dashboard'))

# === ПЕРЕГЛЯД ТЕСТ КЕЙСУ ===
@app.route('/view_testcase/<int:testcase_id>')
def view_testcase(testcase_id):
    if 'user_id' not in session or session['role'] != 'student':
        return redirect(url_for('login'))

    conn = sqlite3.connect('database.db')
    conn.row_factory = sqlite3.Row
    cur = conn.cursor()

    # Основна інформація про тест-кейс + приєднані імена
    cur.execute('''
```

```
        SELECT tc.*,
               s.name AS student_name,
               t.name AS teacher_name,
               task.title AS task_title
        FROM ТестКейси tc
        LEFT JOIN Студенти s ON tc.student_id = s.id
        LEFT JOIN Викладачі t ON tc.teacher_id = t.id
        LEFT JOIN Завдання task ON tc.task_id = task.id
        WHERE tc.id = ? AND tc.student_id = ?
    ''', (testcase_id, session['user_id']))
    testcase = cur.fetchone()

    if not testcase:
        flash('Тест-кейс не знайдено')
        return redirect(url_for('student_dashboard'))

    # Отримання кроків
    cur.execute('''
        SELECT * FROM КрокиТестКейсу
        WHERE testcase_id = ?
        ORDER BY step_number
    ''', (testcase_id,))
    steps = cur.fetchall()

    conn.close()

    return render_template('view_testcase.html',
testcase=testcase, steps=steps)

# === ПЕРЕГЛЯД БАГУ ===
@app.route('/student/view_bug/<int:bug_id>')
def view_bug(bug_id):
    if 'user_id' not in session or session['role'] != 'student':
        return redirect(url_for('login'))

    with sqlite3.connect('database.db') as con:
        con.row_factory = sqlite3.Row  # Дозволяє звертатися до
полів за назвами
        cur = con.cursor()
        cur.execute('''
            SELECT b.*,
                   s.name AS student_name,
                   t.name AS teacher_name,
                   task.title AS task_title
            FROM Баги b
            LEFT JOIN Студенти s ON b.student_id = s.id
            LEFT JOIN Викладачі t ON b.teacher_id = t.id
            LEFT JOIN Завдання task ON b.task_id = task.id
            WHERE b.id = ? AND b.student_id = ?
        ''', (bug_id, session['user_id']))
        bug = cur.fetchone()

    if not bug:
```

```
            flash('Баг-репорт не знайдено')
            return redirect(url_for('student_dashboard'))

    return render_template('view_bug.html', bug=bug)

# === ВИХІД ===
@app.route('/logout')
def logout():
    session.clear()
    return redirect(url_for('login'))

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