ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Веб-додаток для обліку студентів навчальних закладів»

на здобуття освітнього ступеня бакалавра
зі спеціальності 126, інформаційні системи та технології
(код, найменування спеціальності)
освітньо-професійної програми інформаційні системи та технології

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

_____ Захар ГОЛУБ
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач вищої освіти гр.ІСД-43
Захар ГОЛУБ
Ім'я, ПРІЗВИЩЕ
Керівник: д.т.н, професор, Катерина НЕСТЕРЕНКО
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Рецензент: _____
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інформаційних систем та технологій

Ступінь вищої освіти бакалавр

Спеціальність інформаційні системи та технології

Освітньо-професійна програма інформаційні системи та технології

ЗАТВЕРДЖУЮ

Завідувач кафедрою Інф. систем та технологій

Каміла СТОРЧАК .

«____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Голуб Захар Олександрович

(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи:" Веб-додаток для обліку студентів навчальних закладів"

керівник кваліфікаційної роботи Катерина НЕСТЕРЕНКО, доктор технічних наук, професор

(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» березня 2025 р. №56

2. Строк подання кваліфікаційної роботи «30» травня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: Науково технічна література з теми бакалаврської роботи, Аналіз існуючих систем для обліку студентів

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

1. Аналіз предметної області
2. Огляд та порівняння технологій розробки
3. Розробка додатку

5. Перелік ілюстративного матеріалу: *презентація*

6. Дата видачі завдання «24» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вступ	14.03.25 - 26.03.25	
2	Розділ 1. Аналіз предметної області	27.03.25 - 29.03.25	
3	Аналіз доступних рішень у сфері обліку студентів	30.03.25 - 09.04.25	
4	Розділ 2. Огляд та порівняння технологій розробки	10.04.25 - 21.04.25	
5	Розробка архітектури та програмного забезпечення веб-додатку.	22.04.25 - 25.04.25	
6	Розділ 3. Розробка додатку	26.04.25 - 29.04.25	
7	Висновки	30.04.25 – 02.05.25	
8	Оформлення дипломної бакалаврської роботи (чистовий варіант)	03.05.25	

Здобувач вищої освіти _____ Захар ГОЛУБ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Катерина НЕСТЕРЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 85 стор., 23 рис., 1 табл., 21 джерела.

Мета роботи – створення веб-додатку для обліку студентів навчальних закладів, що забезпечує централізоване зберігання, обробку та ефективне управління академічними даними.

Об'єкт дослідження – процес управління академічними даними студентів у закладах вищої освіти.

Предмет дослідження – методи, інструменти та сучасні інформаційні технології для створення веб-додатків, орієнтованих на облік, моніторинг та аналіз студентських даних.

Методи дослідження – аналіз існуючих систем обліку студентів, порівняння технологій веб-розробки, проєктування архітектури додатку, реалізація програмного забезпечення за допомогою Flask, SQLite, Bootstrap, тестування працездатності та інтерфейсу системи. У результаті:

– проаналізовано наявні рішення та їх функціональні можливості;

– обґрунтовано вибір стеку технологій;

– спроєктовано базу даних і архітектуру додатку;

– розроблено інтерфейси для різних ролей користувачів (студент, викладач, адміністратор);

– реалізовано функції додавання, редагування, пошуку та перегляду даних студентів і груп;

– забезпечено захист паролів і розмежування прав доступу;

– протестовано працездатність системи в різних режимах.

Короткий зміст роботи:

У роботі проведено аналіз предметної області управління студентськими даними, досліджено сучасні інформаційні системи та виявлено їх недоліки. Показано доцільність створення власного веб-додатку з урахуванням потреб

українських закладів освіти. Виокремлено ефективний стек технологій для реалізації системи. Розроблено прототип додатку, впроваджено функції взаємодії з користувачами відповідно до їхніх ролей. Обґрунтовано безпечність та гнучкість системи, а також окреслено перспективи її подальшого розвитку.

КЛЮЧОВІ СЛОВА: СТУДЕНТ, ВЕБ-ДОДАТОК, FLASK, БАЗА ДАНИХ, ОБЛІК, ГРУПА, АДМІНІСТРАТОР, ІНТЕРФЕЙС, ІНФОРМАЦІЙНА СИСТЕМА, ОСВІТА

.

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

ПОДАННЯ
ГОЛОВІ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ
ЩОДО ЗАХИСТУ КВАЛІФІКАЦІЙНОЇ РОБОТИ
на здобуття освітнього ступеня бакалавра

Направляється здобувач Голуб З.О. до захисту кваліфікаційної роботи
(прізвище та ініціали)
за спеціальністю 126, інформаційні системи та технології
(код, найменування спеціальності)
освітньо-професійної програми інформаційні системи та технології
(назва)
на тему: «Веб-додаток для обліку студентів навчальних закладів».

Кваліфікаційна робота і рецензія додаються.

Директор ННІ _____ Катерина НЕСТЕРЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Висновок керівника кваліфікаційної роботи

Здобувач Голуб З.О успішно виконав поставлені перед ним завдання, продемонстрував, належний рівень теоритичної підготовки та практичних навичок у процесі створення веб-додатк Робота є актуальною, завершеною та може бути рекомендованою до захисту

Все це дозволяє оцінити виконану кваліфікаційну роботу здобувача Голуба З.О наоцінку «відмінно» та присвоїти йому кваліфікацію бакалавр з інформаційних систем та технологій.

Керівник кваліфікаційної роботи _____ Катерина НЕСТЕРЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

«____»_____20___року

Висновок кафедри про кваліфікаційну роботу

Кваліфікаційна робота розглянута. Здобувач *Голуб З.О.* допускається до захистуданої роботи в Екзаменаційній комісії.

Завідувач кафедрою _____ _____ _____
(назва) *(підпис)* *(Ім'я, ПРІЗВИЩЕ)*

ВІДГУК РЕЦЕНЗЕНТА

на кваліфікаційну роботу

на здобуття освітнього ступеня бакалавра

Здобувача вищої освіти Голуба Захара Олександровича

(прізвище, ім'я, по батькові)

на тему «"Веб-додаток для обліку студентів навчальних закладів»

Актуальність.

У ВНЗ існує потреба в оптимізації процесів обліку студентів її вирішення дозволить зменшити витрати часу та ресурсів на адміністративну роботу. Веб-додаток для обліку студентів спроможний значно полегшити роботу адміністрації та викладачів студентами, надаючи комфортний та швидкий доступ до інформації. Автоматизація різних робочих процесів є необхідним кроком у час стрімкого розвитку інформаційних систем у всіх сферах.

Позитивні сторони.

1. Зміст бакалаврської кваліфікаційної роботи повністю відповідає завданню, а поставлені задачі виконані у повному обсязі
2. Розроблений веб-додаток є актуальним, практично орієнтованим, має зручний інтерфейс і демонструє ефективну реалізацію поставлених функцій.

Недоліки.

1. Обмежений функціонал, який може бути розширений у майбутніх версіях
2. Потребує подальшого тестування на масштабованість та навантаження

Відзначені зауваження не впливають на загальну позитивну оцінку кваліфікаційної роботи бакалаврської.

Висновок: *кваліфікаційна робота на здобуття ступеня бакалавра заслуговує оцінку "відмінно", а здобувач* Голуб Захар Олександрович *заслуговує присвоєння кваліфікації:* бакалавр з інформаційних систем та технологій

Рецензент:

науковий ступінь, вчене звання _____ *підпис* _____ *Ім'я, ПРІЗВИЩЕ*

ЗМІСТ

ВСТУП

У сучасних умовах цифрової трансформації освіти покращення процесів управління академічною інформацією студентів є одним із ключових пріоритетів для вищих навчальних закладів. Зростаюча кількість студентів, розширення освітніх програм і підвищення вимог до якості навчання створюють необхідність у впровадженні ефективних IT-рішень для автоматизації обліку та аналізу академічної діяльності. Класичні підходи до роботи з даними, засновані на паперовій документації або використанні застарілих програмних засобів, таких як Excel чи Access, уже не відповідають сучасним стандартам. Вони характеризуються низькою ефективністю, обмеженою масштабованістю, складністю в адмініструванні та підвищеним ризиком помилок через ручне введення й обробку інформації.

Відсутність централізованої системи управління даними створює значні організаційні труднощі. Інформація розпорошується між викладачами, кафедрами чи адміністративними підрозділами, що ускладнює її оновлення, аналіз і обмін. Такий підхід знижує продуктивність навчального процесу, ускладнює планування, моніторинг успішності студентів і координацію між підрозділами. Саме тому виникає потреба у створенні веб-додатку, який би забезпечив єдиний інформаційний простір для роботи з академічними даними, спростив доступ до інформації та підвищив ефективність управління.

Мета роботи. Основною метою є створення веб-додатку для управління академічними даними студентів, який забезпечить централізоване зберігання інформації, швидкий доступ до неї, ефективну обробку та можливість керування різними аспектами навчального процесу. Система має бути зручною для щоденного використання, легко адаптуватися до змін у структурі навчального закладу та підтримувати розширення функціона

Завдання дослідження:

Провести аналіз існуючих систем управління студентськими даними, оцінити їхні переваги, недоліки та можливості застосування в умовах українських ВНЗ. Обґрунтувати вибір технологій для реалізації веб-додатку, зокрема фреймворку Flask як серверної основи та SQLite як простої у використанні СУБД. Реалізувати базові та розширені функціональні можливості системи, включаючи додавання, редагування, видалення, пошук студентів, керування групами, ролями користувачів тощо. Провести тестування працездатності веб-додатку, перевірити коректність роботи кожного з модулів та оцінити зручність інтерфейсу для користувачів різних ролей.

Об'єкт дослідження: процес управління академічними даними студентів у закладах вищої освіти, включаючи збирання, зберігання, обробку, аналіз і відображення даних для різних категорій користувачів.

Предмет дослідження: методи, інструменти та сучасні інформаційні технології, що застосовуються для створення веб-додатків, орієнтованих на облік, моніторинг та аналіз студентських даних.

Наукова новизна. Основна новизна дослідження полягає в інтеграції сучасних технологій веб-розробки (Flask, SQLAlchemy, SQLite, Bootstrap) у сферу академічного обліку студентів. Запропонований підхід дозволяє автоматизувати рутинні процеси та адаптувати систему до потреб конкретного навчального закладу. На відміну від існуючих рішень, система вирізняється гнучкістю, простотою впровадження, масштабованістю та орієнтацією на користувацький досвід. Особливу увагу приділено безпеці зберігання да
авторизації користувачів і розмежуванню доступу відповідно до їхніх ро

Практичне значення роботи полягає у створенні веб-додатку, який може бути впроваджений у практику навчальних закладів. Його використання зменшує навантаження на адміністративний персонал, підвищує ефективність роботи з даними та покращує організацію навчального процесу. Система має потенціал до вдосконалення та масштабування, що робить її перспективною

для використання не лише у ВНЗ, а й у професійно-технічних закладах чи освітніх центрах.

З огляду на глобальні тенденції цифровізації освіти, розробка подібних систем набуває міждисциплінарного значення. Вона поєднує знання з програмування, проєктування баз даних, UX-дизайну, інформаційної безпеки та освітнього менеджменту. Реалізація такого веб-додатку має не лише прикладне значення, а й демонструє цінність інтеграції інформаційних технологій у традиційні освітні процеси, сприяючи стратегії сталого розвитку освіти.

Актуальність локального контексту. Українські вищі навчальні заклади часто стикаються з обмеженими ресурсами для впровадження складних IT-систем. Розроблений веб-додаток враховує ці обмеження, пропонуючи легке у розгортанні рішення, яке не потребує значних фінансових вкладень чи складної інфраструктури. Використання SQLite як бази даних і Flask як серверного фреймворку дозволяє швидко розгорнути систему навіть на локальних серверах, що робить її доступною для невеликих закладів.

Освітній потенціал системи. Окрім прямого практичного застосування, веб-додаток може слугувати навчальним інструментом для студентів IT-спеціальностей. Він демонструє реальний приклад використання сучасних технологій веб-розробки, включаючи роботу з базами даних, створення безпечних систем авторизації та розробку зручних інтерфейсів. Це сприяє підвищенню якості підготовки фахівців, які зможуть створювати аналогічні рішення для інших сфер.

Соціальний вплив. Впровадження централізованої системи управління даними сприяє підвищенню прозорості освітнього процесу. Студенти отримують швидкий доступ до власних академічних даних, викладачі — до інформації про групи, а адміністрація — до аналітичних звітів. Це створює більш відкрите та ефективне середовище, що відповідає сучасним принципам інклюзивної освіти.

Перспективи розвитку. У майбутньому систему можна розширити шляхом інтеграції з іншими платформами, наприклад, системами дистанційного навчання чи електронними журналами. Додавання функцій аналізу даних, таких як прогнозування успішності чи автоматичне формування звітів, може ще більше підвищити її цінність. Крім того, перехід на хмарні технології дозволить масштабувати додаток для використання в мережах навчальних закладів або на національному рівні.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальний огляд існуючих рішень

У цьому розділі буде розглянуто деякі з уже існуючих варіантів програмного забезпечення, призначеного для обліку студентів, які були знайдені під час аналізу наявних інформаційних ресурсів та огляду ринку цифрових рішень у сфері освіти. Сьогодні існує велика кількість різноманітних додатків і платформ, що виконують завдання, пов’язані з веденням академічної документації, управлінням групами студентів, аналізом успішності, формуванням звітності тощо. Проте, попри чисельність подібних продуктів, більшість із них мають лише базовий набір функцій, орієнтований на стандартні потреби. Вони не враховують специфіку певних навчальних закладів, факультетів або освітніх програм, і, як наслідок, не завжди здатні повною мірою задовольнити індивідуальні вимоги користувачів.

Значна частина наявних рішень фокусується на автоматизації обліку студентів загального рівня: реєстрація, призначення до груп, перегляд оцінок, формування академічних звітів. Проте сучасний ВНЗ — це складна динамічна система, що постійно змінюється. Деякі навчальні програми потребують врахування додаткових характеристик студента, таких як участь у наукових гуртках, волонтерській діяльності, рейтинги для стипендій, активність у проектах, зворотний зв’язок з викладачами або навіть зовнішніми партнерами (наприклад, роботодавцями). Багато типових програм не враховують ці вимоги, а їх гнучкість щодо адаптації під індивідуальні потреби користувача часто є обмеженою.

Тому, вивчивши існуючі варіанти програмного забезпечення, я поставив перед собою завдання не просто зробити копію вже наявного рішення, а створити веб-додаток, який буде пристосований до конкретних вимог навчального закладу, простий у використанні, легко масштабований, захищений і водночас інтуїтивно зрозумілий для користувачів різних категорій. Перед створенням власної системи було важливо чітко визначити,

які типи програм бувають, які переваги та недоліки кожен із них має, і в яких випадках вони доцільні для використання.

Типи програм для обліку студентів

1. Десктопний додаток

Десктопні програми зазвичай встановлюються безпосередньо на комп’ютер користувача та можуть працювати без постійного підключення до інтернету. Це дозволяє забезпечити автономність роботи, що особливо актуально в умовах обмеженого доступу до мережі.Також завдяки використанню ресурсів персонального комп’ютера, десктопні додатки здатні обробляти великі обсяги даних та виконувати складні обчислення.Часто такі програми мають ширший функціонал порівняно з мобільними додатками або веб-рішеннями, оскільки не обмежені вимогами до швидкодії та споживання енергії. Разом із тим, десктопні додатки вимагають окремого встановлення на кожному пристрої, складніші в оновленні та розгортанні, а також менш зручні для командної роботи або доступу з різних місць.

2. Мобільний додаток

Мобільні додатки призначені для використання на смартфонах або планшетах, що робить їх незамінними в умовах мобільності користувачів. Вони дозволяють оперативно переглядати необхідну інформацію в будь-який момент часу, незалежно від місцезнаходження.Такі додатки інтегруються з функціоналом пристрою — камерою, мікрофоном, геолокацією, біометричними сенсорами тощо. Це дає змогу розширити можливості взаємодії користувача з додатком, наприклад, завантаження фото документів або ідентифікація за відбитком пальця. Інтерфейс мобільних додатків розроблений із врахуванням сенсорного управління, обмеженого екранного простору та простоти навігації. Завдяки цьому вони зручні у щоденному користуванні, але рідко використовуються для введення або обробки великих масивів даних. Зазвичай мобільні додатки потребують розробки окремо для iOS та Android, що збільшує витрати на підтримку та розширення функціоналу.

3. Веб-додаток

Веб-додатки є найбільш універсальним рішенням, яке не потребує встановлення на пристрій. Вони працюють безпосередньо в браузері, що робить їх доступними з будь-якого комп’ютера, ноутбука або мобільного пристрою, незалежно від операційної системи. Користувачам достатньо мати доступ до інтернету та браузер, щоб отримати повний функціонал додатка. Це значно спрощує адміністрування, оскільки оновлення застосунку відбувається централізовано — всі зміни одразу доступні для всіх користувачів. Веб-рішення, як правило, мають адаптивний інтерфейс, що підлаштовується під розмір екрану пристрою. Завдяки цьому досягається універсальність та зручність користування незалежно від того, чи це ПК, планшет чи телефон. Додатковою перевагою є те, що основна логіка та обробка даних відбувається на сервері, тому клієнтські пристрої не перевантажуються. Це дозволяє підтримувати одночасну роботу багатьох користувачів, зберігаючи стабільність системи навіть при високому навантаженні. З огляду на вищезазначене, для реалізації цілей мого дипломного проєкту було обрано саме веб-додаток, як найбільш гнучкий, доступний, економічно доцільний і масштабований варіант. У наступних підрозділах буде детальніше розглянуто особливості реалізації функціоналу, архітектури та вибору технологій для його створення.

1.1.1 PowerSchool SIS

PowerSchool SIS — це одна з найпоширеніших і найпотужніших систем управління навчальним процесом у середній та вищій освіті. Вона пропонує комплексний набір інструментів, що дозволяють навчальним закладам ефективно організовувати адміністративну, академічну та комунікаційну діяльність. Основна функція системи — централізоване зберігання та обробка даних про студентів, викладачів, навчальні дисципліни, розклад, оцінювання та відвідуваність.

Система дозволяє адміністраторам створювати і редагувати інформацію про навчальні класи, призначати викладачів до предметів, налаштовувати періоди навчання, а також формувати розклад занять і розподіляти ресурси. Завдяки широкому набору параметрів, PowerSchool підтримує гнучку конфігурацію під потреби конкретного навчального закладу. Вбудовані модулі дають змогу вести електронні журнали, автоматично генерувати табелі успішності та здійснювати моніторинг академічної активності учнів або студентів.

Окремий функціональний модуль розроблений спеціально для студентів, які можуть, авторизувавшись у системі, переглядати власні оцінки, домашні завдання, коментарі від викладачів, а також розклад занять. Батьки учнів (у разі шкільного використання) також можуть мати обмежений доступ до перегляду успішності, що забезпечує прозорість та підвищує рівень залученості до освітнього процесу.

Інтерфейс програми має сучасний багатофункціональний дизайн із численними розділами, вкладками та фільтрами. Інформація в ньому структурована таким чином, щоб кожен тип користувача мав доступ лише до тих функцій, які йому дозволено використовувати. Це покращує безпеку, мінімізує ризик помилок та підвищує загальну зручність користування.

Крім того, PowerSchool SIS підтримує інтеграцію з іншими освітніми платформами та сервісами, такими як Google Classroom, Microsoft Teams та сторонні системи аналітики. Серед додаткових можливостей можна відзначити імпорт та експорт даних у різних форматах, підтримку API для розробників, ведення статистики та генерацію звітів за різними критеріями.

Попри всі переваги, система орієнтована переважно на великі школи та коледжі в США і може виявитися надто складною для невеликих навчальних закладів. Її повноцінне використання потребує певного рівня технічної підготовки персоналу, а також відповідних ресурсів для підтримки і обслуговування. (Рис. 1.1).

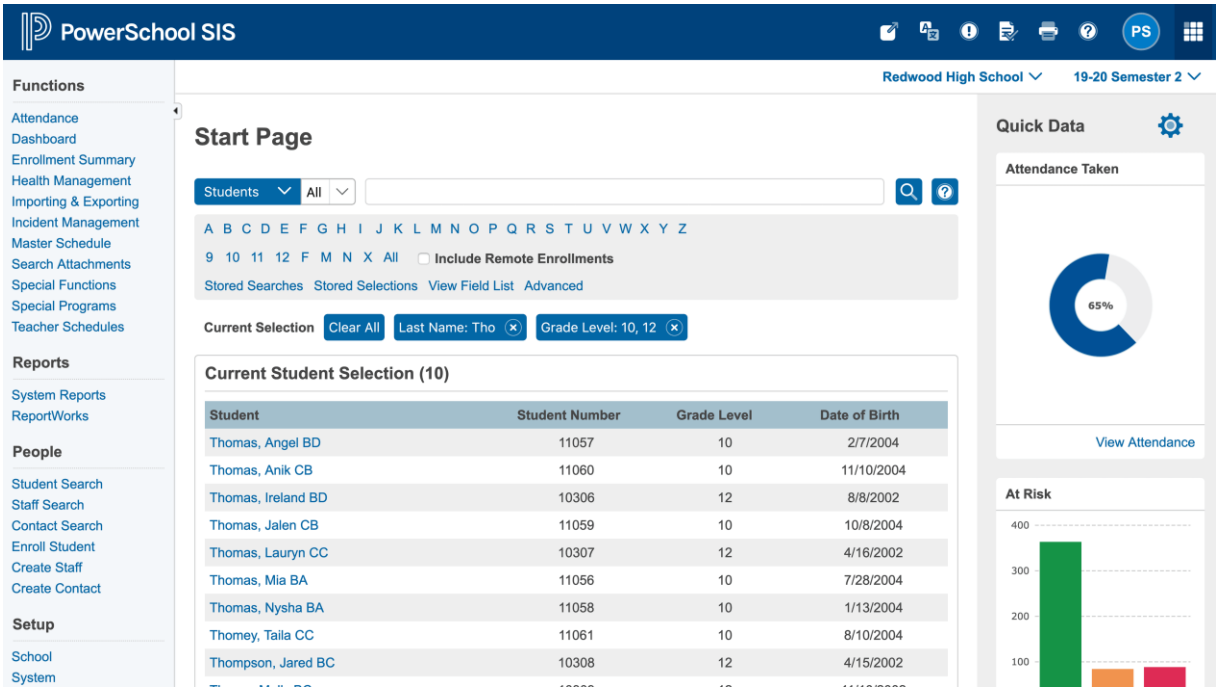


Рис. 1.1 Інтерфейс PowerSchool SIS

1.1.2 Educationalist School App

Даний мобільний додаток дає можливість вчителям керувати своїм класом, для учнів можна створити свій кабінет для полегшення процесу навчання. Вигляд програми відображено на Рис. 1.2.

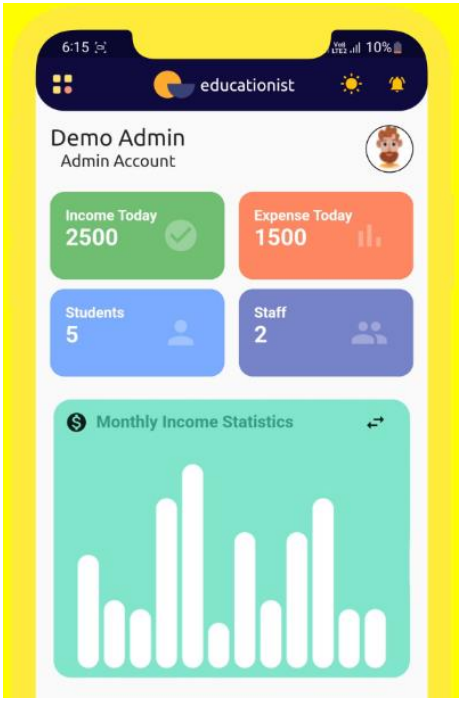


Рис1.2 Інтерфейс Educationalist School App

1.1.3 eSckooly

У цій програмі для керування навчальним процесом існує декілька відмінностей від вище наведених. Ця програма була створена також і для ВНЗ, має десктопну версію під різні операційні системи і власний додаток. В додатку є різні унікальні розділи які можна плбачити на (Рис. 1.3).

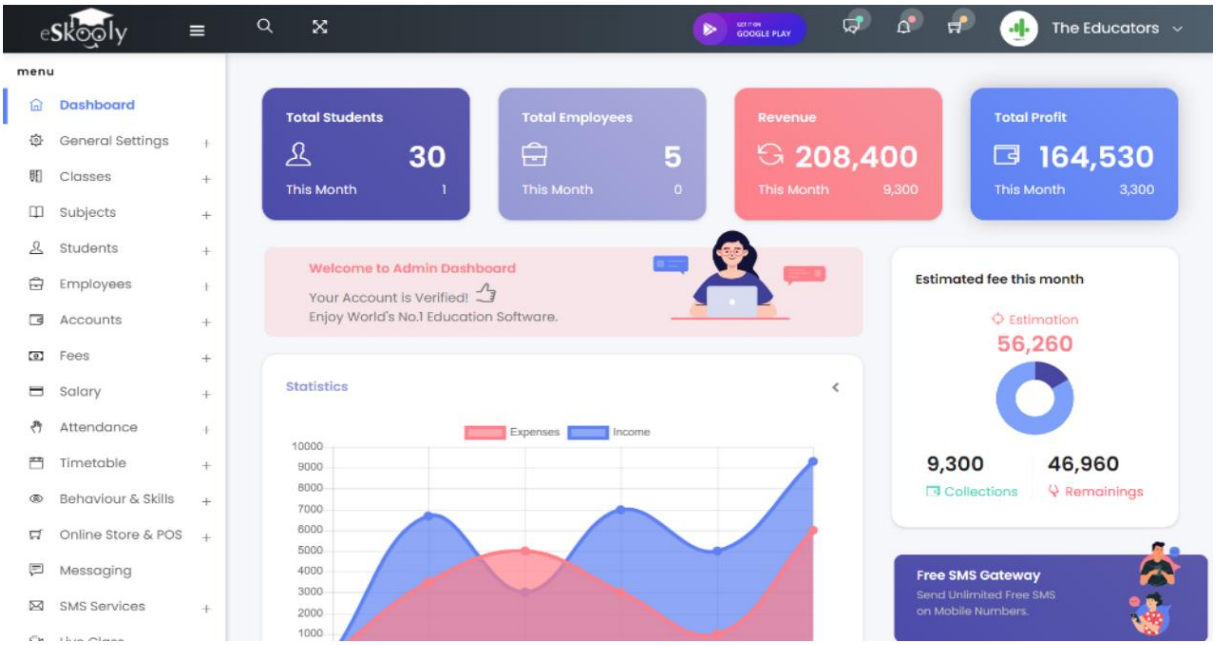


Рис1.3 Інтерфейс eSckooly

1.2 Порівняння існуючих рішень

Кожна з програм може бути порівняна з іншими. Для цієї цілі була введена таблиця для порівняння (табл. 1.1). Наявні функції будуть оцінені у вигляді "+" (функціонал відсутній) та "-" (функціонал наявний).

Таблиця 1.1

Назва	Підключення до бази даних	Наявність мультіплатформе ності	Можливість залучення роботодавців	Взаємодія зі студентами/у чнями	Реалізація пошуку студентів

PowerSchool SIS	+	+	-	+	+
eSckooly	+	-	+	+	+
Educationalist	+	-	-	+	+

Оцінюючи дану таблицю порівняння, можна зробити висновок, що кожне з представлених програмних рішень має свої характерні переваги, але водночас — і низку суттєвих недоліків. Жодна з проаналізованих систем не є універсальною або повністю адаптованою до специфіки вищої освіти, особливо в українських реаліях. Зокрема, дві з трьох програм були розроблені переважно для шкільного середовища, що значно обмежує можливості інтеграції з ринком праці, відсутні механізми професійної орієнтації, системи комунікації з роботодавцями, підтримка реального моніторингу стажування або автоматичного формування звітів про успішність студентів у зовнішніх освітніх або практичних середовищах.

Незважаючи на загальну схожість функцій — таких як підключення до бази даних, реалізація пошуку студентів, доступ до базової інформації — більшість із цих систем не дозволяють налаштувати розширене управління освітнім процесом, інтегруватися з внутрішніми службами ВНЗ або розширити функціонал без суттєвих технічних втручань. Наприклад, відсутність можливості для додавання індивідуальних характеристик студентів, фіксації динаміки їх успішності або гнучкої зміни ролей користувачів у системі може бути критичною для повсякденного функціонування реального навчального закладу.

Позитивним фактором є використання баз даних усіма трьома системами, що свідчить про загальну тенденцію до автоматизації збереження та обробки інформації. Це дозволяє зменшити навантаження на працівників навчального закладу, підвищити точність обліку та оперативність внесення змін. Крім того, реалізований пошук і часткова взаємодія зі студентами в кожному з рішень робить ці продукти функціональними з точки зору базових

освітніх задач. Проте ці функції часто є занадто загальними, не враховують контекстні потреби (наприклад, підтримку міжфакультетської міграції студентів, зберігання історії змін даних, прив’язку до навчальних планів, модульних систем оцінювання тощо).

1.3 Висновок до розділу

Дослідження та порівняльний аналіз доступних програмних продуктів для обліку студентів показали, що незважаючи на різноманітність рішень, більшість із них мають вузьку функціональну спрямованість і орієнтовані переважно на шкільну аудиторію або стандартизовані навчальні процеси. Основний акцент робиться на базових операціях — збереження облікових даних студентів, оцінок, розкладу — без урахування розширених можливостей для глибшого аналізу, міжрівневої інтеграції та цифрової трансформації управління освітою.

Більшість рішень мають обмежену взаємодію з зовнішніми середовищами — роботодавцями, онлайн-платформами, державними освітніми реєстрами. Це суттєво знижує потенціал програм для комплексного супроводу освітньо-професійної траєкторії студента, починаючи від вступу до ВНЗ і завершуючи працевлаштуванням. Відсутність механізмів аналітики, автоматизованих рекомендацій, історичної звітності, зворотного зв’язку зі стейкхолдерами (викладачами, батьками, керівництвом, роботодавцями) зменшує ефективність і функціональність таких продуктів у довгостроковій перспективі.

Розроблена в межах цієї роботи система принципово відрізняється підходом до побудови інформаційного середовища. Вона не лише забезпечує базовий облік студентів, а й орієнтована на розширення освітньої екосистеми ВНЗ. У додатку реалізовано механізми для створення, зберігання та аналізу характеристик студентів, формування єдиної бази як для діючих студентів, так і для випускників, що відкриває нові перспективи в аспекті побудови бази

даних контактів для кар’єрного розвитку, профорієнтації, залучення до спільних проєктів із бізнесом та органами місцевого самоврядування.

Система може використовуватись не лише як внутрішній інструмент управління, а й як основа для зовнішньої комунікації з партнерами — базами практик, HR-відділами компаній, які можуть отримати доступ до відкритої частини інформації про студентів. Крім того, наявність можливості подальшого масштабування, підтримка сучасних протоколів безпеки, гнучке налаштування ролей і прав доступу роблять додаток перспективним рішенням для широкого впровадження.

Таким чином, проведене дослідження обґрунтовує актуальність створення власного веб-додатку, що відповідає вимогам часу, потребам навчальних закладів і має достатній потенціал для адаптації під різні моделі організації освітнього процесу. Цей проєкт демонструє, що навіть в умовах обмежених ресурсів можна створити ефективний, простий і водночас функціонально насичений інструмент для цифрового управління академічною інформацією.

2. ОГЛЯД ТА ПОРІВНЯННЯ ТЕХНОЛОГІЙ РОЗРОБКИ

2.1 Вибір технологій для розробки системи керування студентами

Створення веб-додатку для керування студентами — це не лише про написання коду, а й про стратегічний вибір інструментів, які забезпечують ефективність, зручність розробки та подальшу масштабованість. У цьому розділі детально розглядаються технології, обрані для реалізації проєкту, їх переваги та альтернативи.

2.1.1 Серверна частина: Flask як оптимальний вибір

Для реалізації серверної частини було вирішено скористатися Flask — мікрофреймворком на мові Python. Його основною перевагою є легкість у використанні та відсутність зайвого «багажу». Flask не нав’язує жорстку структуру проєкту, як це роблять більш важкі фреймворки. Це дозволяє розробнику самостійно визначати, як буде виглядати архітектура системи, поступово додаючи лише ті функції, які дійсно необхідні.

Особливо цінною є гнучкість Flask. Наприклад, можна без зусиль підключити модуль аутентифікації користувачів за допомогою розширення Flask-Login, а для роботи з базою даних — SQLAlchemy. Якщо в майбутньому виникне потреба додати функціонал на основі машинного навчання або аналітики даних, інтеграція буде доволі простою завдяки широкій підтримці Python-бібліотек.

Важливо зазначити, що серед альтернатив розглядалися також Django та FastAPI. Django, хоч і дуже потужний, має багато вбудованих компонентів, які у межах цього проєкту були б зайвими. Наприклад, автоматична адміністративна панель, система роботи з формами, складна структура додатків — усе це потребує часу на освоєння і не завжди виправдане для невеликої студентської системи. FastAPI, своєю чергою, більше орієнтований на розробку REST API і чудово підходить для застосунків, де важлива висока

продуктивність і обробка JSON-запитів. Однак у нашому випадку перевагу отримав класичний підхід із серверним рендерингом сторінок, і Flask якраз ідеально вписується в таку концепцію.

2.1.2 Вибір бази даних

Одним з важливих рішень стало визначення, яку систему управління базами даних використовувати. Для дипломного проєкту найбільш логічним вибором виявилася база даних SQLite. Її головна перевага — простота. Щоб почати з нею працювати, не потрібно налаштовувати окремий сервер, створювати користувачів чи налаштовувати права доступу. Достатньо вказати шлях до файлу — і база готова до роботи.

Для проєкту, який охоплює декілька сотень записів (наприклад, дані про студентів, викладачів, оцінки тощо), продуктивності SQLite більш ніж достатньо. До того ж вона дуже зручна у розробці: будь-яку таблицю чи стовпець можна швидко змінити, не ризикуючи порушити загальну логіку системи. Це дозволяло експериментувати з моделями даних без страху, що зміни призведуть до складних наслідків.

Також важливо розуміти, що вибір на користь SQLite не означає відмову від масштабування. У разі необхідності база може бути замінена на PostgreSQL чи MySQL — це не потребуватиме кардинальної перебудови застосунку, адже використовувалася універсальна ORM-бібліотека SQLAlchemy, яка підтримує різні типи СУБД.

2.1.3 Користувацький інтерфейс: Bootstrap 5 і Jinja2

Зовнішній вигляд веб-додатку не менш важливий, ніж його функціональна частина. Навіть найпотужніша система повинна бути зручною для користувача, особливо якщо мова йде про щоденне використання у навчальному закладі. Саме тому для створення інтерфейсу було використано Bootstrap 5 — сучасний CSS-фреймворк, який дозволяє швидко реалізовувати адаптивні сторінки без глибокого занурення в стилі.

За допомогою Bootstrap вдалося створити інтерфейс, що однаково добре відображається як на комп’ютерах, так і на мобільних пристроях. Крім того, фреймворк надає набір готових компонентів — кнопок, форм, меню — що значно скоротило час розробки.

Що стосується шаблонізації, то тут використовується Jinja2 — потужний механізм, вбудований у Flask. Він дозволяє динамічно підставляти дані з Python у HTML-шаблони, що робить сторінки «живими». Наприклад, список студентів, оцінки, дані про групи — усе це виводиться автоматично, без ручного дублювання коду. Це суттєво спрощує підтримку проєкту: достатньо змінити шаблон один раз — і зміни будуть застосовані на всіх сторінках.

На етапі вибору технологій для фронтенду також розглядалися сучасні JavaScript-фреймворки, такі як React або Vue. Проте їх застосування було б недоцільним у межах цього проєкту. Такі фреймворки більше підходять для складних інтерактивних інтерфейсів або SPA (single-page applications), де велика частина логіки переноситься на клієнт. У нашому ж випадку достатньо було класичного підходу з серверною генерацією HTML. Це не тільки спростило реалізацію, а й зробило код більш зрозумілим і прозорим.

2.1.4 Інші важливі технології

Окрім основних компонентів, у проєкті були використані також кілька додаткових бібліотек, які значно покращили зручність і безпеку:

Flask-Login — забезпечив реалізацію авторизації користувачів, дозволив розділити ролі (наприклад, студент, викладач, адміністратор) і керувати сесіями без зайвих зусиль.

Werkzeug — надійний інструмент для безпечного зберігання паролів. Завдяки йому усі паролі перед збереженням у базі даних були хешовані.

Flask-Migrate — полегшив оновлення структури бази даних. Це особливо зручно на етапі активної розробки, коли зміни у моделях трапляються доволі часто.

2.1.5 Підсумки

Обраний стек технологій виявився вдалим рішенням для створення навчального веб-додатку. Flask забезпечив гнучкість і простоту, SQLite — швидкий старт без складної конфігурації, а Bootstrap з Jinja2 — зручний інтерфейс, придатний для щоденного використання.

Ці інструменти дозволили не лише реалізувати усі функції, заплановані в межах дипломної роботи, а й залишили простір для розвитку. У майбутньому проєкт можна безболісно масштабувати, розширити функціональність, інтегрувати аналітичні модулі або перейти на продуктивніші бази даних. І все це — без потреби в повному переписуванні коду. Саме тому вибір цих технологій можна вважати виваженим та ефективним.

2.2 Порівняння технологій, альтернативи та обґрунтування вибору

Проєктування веб-застосунку, навіть якщо йдеться про освітню систему невеликого масштабу, потребує зваженого, стратегічного підходу до вибору інструментів розробки. Сучасна екосистема веб-технологій є надзвичайно різноманітною та динамічною. Щороку з’являються нові фреймворки, бібліотеки, сервіси та платформи, які пропонують розробникам широкий вибір інструментів для створення функціональних, безпечних та масштабованих рішень.

При цьому кожна з доступних технологій має свої сильні сторони, специфіку використання та обмеження. Наприклад, одні з них орієнтовані на швидку розробку MVP (minimum viable product), інші — на побудову корпоративних систем з високою продуктивністю та складною бізнес-логікою. Універсальних рішень не існує: вибір стеку технологій має відповідати цільовому призначенню проєкту, ресурсним обмеженням, досвіду команди розробки та очікуванням кінцевих користувачів.

Найбільш поширеною помилкою серед початківців є обрання «наймоднішої» або «найвідомішої» технології без глибокого аналізу відповідності її потребам проєкту. У результаті — надлишкова складність,

непотрібні залежності, труднощі з налаштуванням, проблеми із сумісністю та неефективне використання часу. Навпаки, обдуманий вибір дозволяє не лише спростити розробку, а й закласти основу для подальшої підтримки, масштабування та впровадження в реальні умови експлуатації.

Особливістю освітніх застосунків є багаторольовість (студент, викладач, адміністратор), потреба у простому інтерфейсі, захисті персональних даних, а також можливість адаптації під змінні освітні процеси. Враховуючи це, розробник має не просто «підібрати фреймворк», а сформувати збалансований набір технологій — стек, який буде відповідати як функціональним, так і нефункціональним вимогам системи. До таких вимог можна віднести швидкість розробки, легкість розгортання, підтримку безпеки, документованість, активність спільноти, сумісність між компонентами та можливість подальшого розширення.

У цьому розділі буде проведено глибокий аналіз технологій, які були розглянуті на етапі проєктування. Для кожного компонента системи (бекенд, база даних, інтерфейс, архітектура рендерингу, безпека) описано як основний вибір, так і альтернативні варіанти, які були відкинуті з обґрунтуванням причин. Такий підхід дозволяє не лише продемонструвати розуміння сучасних інструментів веб-розробки, а й показати вміння мислити критично, порівнювати та приймати технічно обґрунтовані рішення.

Таким чином, описаний нижче стек (Flask, SQLite, Bootstrap, Jinja2) не був обраний навмання. Кожна з цих технологій була обрана після зіставлення з альтернативами, з урахуванням масштабу проєкту, навчального контексту, обмежень середовища та перспектив подальшого використання. Як наслідок, проєкт отримав просту, але водночас гнучку, розширювану й безпечну архітектуру, яка цілком підходить для реального впровадження у навчальні заклади.

2.2.1. Альтернатива Flask: Django, FastAPI, Express.js

Flask було обрано як серверну платформу для розробки веб-додатка. Незважаючи на те, що Flask іноді позиціонують як "мікрофреймворк", його можливостей більш ніж достатньо для створення повноцінного, масштабованого веб-застосунку. Його мінімалістичний підхід дозволяє гнучко організовувати архітектуру, підключаючи лише необхідні компоненти. Однак перед тим, як зупинитися на Flask, були розглянуті інші фреймворки.

Django — повноцінний Python-фреймворк, який надає безліч функцій із коробки: ORM, панель адміністратора, автентифікацію, роботу з формами, шаблонізатор, маршрутизацію тощо. Його використання виправдане для великих і корпоративних проєктів, де потрібно обробляти великий обсяг даних, або де важлива сувора структура. Проте у випадку дипломного проєкту надлишкова складність Django могла стати перешкодою. Високий поріг входу, жорстка структура директорій, складність налаштування — усе це збільшує час реалізації.

FastAPI — відносно новий, асинхронний фреймворк для побудови REST API на Python. Його основною перевагою є продуктивність та автоматична генерація документації (OpenAPI, Swagger UI). Але FastAPI розрахований переважно на back-end системи без шаблонізації HTML. У нашому випадку акцент був зроблений саме на рендерінг сторінок на сервері, тому FastAPI виявився не надто придатним.

Express.js — серверний фреймворк для JavaScript/Node.js. Має високу продуктивність та просту структуру, широко використовується у проєктах із клієнтською логікою (React, Vue, Angular). Проте його застосування у дипломному проєкті вимагало б зміни мови програмування на JavaScript, що ускладнило б реалізацію через необхідність працювати з різними мовами на бекенді та фронтенді.

У висновку Flask був обраний через свою гнучкість, легкість, наявність розвиненої екосистеми розширень (Flask-Login, Flask-Migrate, Flask-WTF), зручний шаблонізатор Jinja2 та активну спільноту.

2.2.2. Бази даних: PostgreSQL, MySQL, SQLite

У проєкті використовується SQLite — легка файлова база даних, яка працює без сервера. Альтернативи включали такі поширені системи:

PostgreSQL — потужна об’єктно-реляційна СУБД з підтримкою складних запитів, транзакцій, тригерів, індексів тощо. Це чудовий варіант для великих систем з великою кількістю користувачів і високими вимогами до цілісності даних. Проте її розгортання потребує окремого серверного середовища та певного досвіду адміністрування.

MySQL/MariaDB — ще одна популярна серверна СУБД, яка широко використовується у веб-проєктах. Вона швидка, надійна і добре підтримується хостингами. Але, як і у випадку з PostgreSQL, робота з нею потребує додаткової конфігурації та підтримки.

SQLite, порівняно з ними, має суттєві переваги саме для невеликого проєкту а саме: швидке розгортання без встановлення додаткового ПЗ, простота резервного копіювання (достатньо скопіювати один файл), достатня продуктивність для навчального навантаження (до кількох сотень записів).

Єдине, що варто враховувати — обмеження на паралельні запити і відсутність підтримки складної транзакційної логіки. Але у рамках цієї системи таких потреб не виникає, тому SQLite — логічний вибір для MVP та демонстраційної версії системи.

2.2.3. Альтернатива Bootstrap: Tailwind, Bulma, Materialize

Для реалізації інтерфейсу було обрано Bootstrap 5, який є одним із найпопулярніших фреймворків для розробки адаптивного інтерфейсу. Він має

розгалужену документацію, безліч компонентів, а головне — дозволяє швидко зробити базовий дизайн без поглиблених знань CSS.

Водночас існують альтернативи:

Tailwind CSS — сучасний утилітарний CSS-фреймворк, що надає набір класів для точного керування стилями. Він дозволяє створювати оригінальні дизайни без використання готових компонентів. Однак Tailwind вимагає більше часу на розробку, адже розробник має самостійно продумувати структуру й зовнішній вигляд усіх елементів.

Bulma — легкий CSS-фреймворк, схожий на Bootstrap, але з більш простою філософією. Містить зручну систему сіток і базових компонентів, але менш розповсюджений.

Materialize — фреймворк, створений на основі концепції Material Design від Google. Він надає стильні компоненти, але менш гнучкий, ніж Bootstrap, і обмежений у налаштуваннях.

Враховуючи простоту, популярність та швидкість розробки, Bootstrap 5 виявився найбільш збалансованим вибором для проєкту, де інтерфейс має бути функціональним і зрозумілим, але не перевантаженим візуальними ефектами.

2.2.4. Серверний рендеринг чи SPA?

Окремо варто згадати дилему між серверним рендерингом (SSR) і односторінковими додатками (SPA). У сучасній веб-розробці популярності набули SPA-підходи з використанням React, Angular чи Vue. Такі застосунки забезпечують високу інтерактивність, швидке оновлення даних без перезавантаження сторінок та можливість реалізації складної логіки на клієнті.

Однак реалізація SPA потребує не лише знання JavaScript, а й побудови повноцінного API на бекенді, що суттєво ускладнює розробку. У випадку навчального проєкту, де основна логіка пов’язана з управлінням користувачами, групами та оцінками, серверного рендерингу цілком

достатньо. Він дозволяє спростити архітектуру, уникнути дублювання логіки на фронтенді та бекенді, а також забезпечити кращу індексацію сторінок.

Саме тому було вирішено використовувати Jinja2 як серверний шаблонізатор у поєднанні з Flask, що дозволило генерувати HTML-сторінки безпосередньо на сервері.

2.2.5. Безпека, автентифікація, розширюваність

Ще одним критично важливим аспектом є безпека користувачів і збереження персональних даних. Для цього було використано:

Flask-Login — модуль для автентифікації користувачів, що дозволяє зручно реалізувати систему входу та розмежування прав доступу;

Werkzeug.security — механізм безпечного хешування паролів, який запобігає їх компрометації навіть у разі витоку бази даних.

Архітектура додатку побудована таким чином, щоб у майбутньому було легко додати нові ролі , реалізувати облік відвідувань, оцінок або автоматичний аналіз успішності.

2.2.6 Висновок

Ретельний аналіз доступних технологій дозволив сформувати оптимальний стек для реалізації навчальної інформаційної системи. Вибір Flask, SQLite, Bootstrap і Jinja2 не був випадковим — кожен компонент пройшов через етап порівняння з альтернативами, і в кожному випадку було враховано доцільність у контексті дипломного проєкту.

Ці технології забезпечили просту архітектуру, легкість у розгортанні, достатню функціональність і, що найважливіше, можливість подальшого розвитку проєкту без кардинальних змін. Такий підхід дозволяє поєднати навчальні цілі з реальним досвідом веб-розробки, що є важливим аспектом підготовки сучасного фахівця в галузі інформаційних технологій.

2.3 Функціональні можливості додатка

Розроблений веб-додаток для обліку студентів навчальних закладів пропонує комплексний набір функціональних можливостей, які забезпечують ефективне управління академічними даними та взаємодію між адміністрацією, викладачами та студентами. Система створена з урахуванням потреб користувачів різних ролей, що дозволяє централізовано керувати інформацією, забезпечити безпеку даних і зручний інтерфейс.

1. Автентифікація та розмежування ролей

Система підтримує механізм реєстрації та входу користувачів із використанням захищеного зберігання паролів (хешування). Всі користувачі при вході отримують відповідні права доступу, залежно від призначеної ролі: адміністратор, викладач, або студент. Для кожної ролі визначено окремий інтерфейс, що мінімізує ризик випадкового доступу до несанкціонованої інформації.

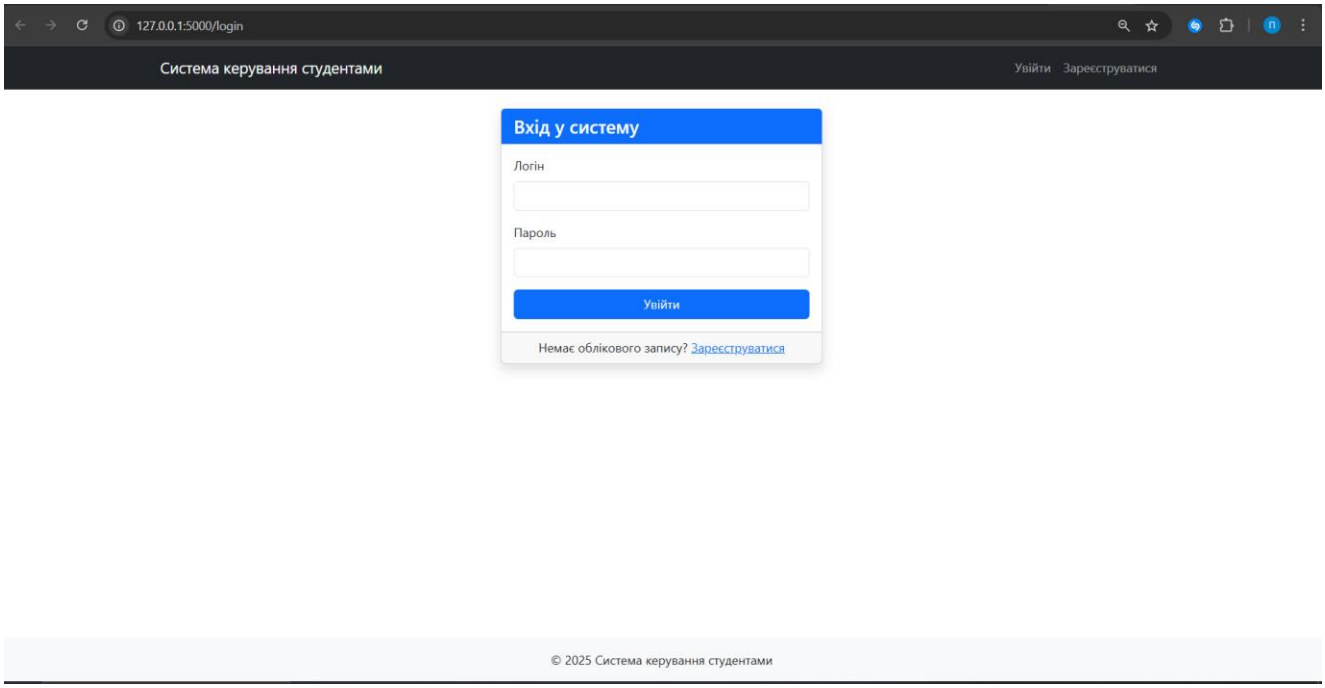


Рис. 2.1. Інтерфейс автентифікації.

2. Панель адміністратора

Адміністратор має повний контроль над даними в системі. Основні можливості:

1. Керування користувачами: створення нових облікових записів, редагування даних, зміна ролей, видалення.
2. Керування студентами: перегляд, редагування, видалення записів, контроль за навчальними групами.
3. Керування групами: створення нових груп із перевіркою унікальності назв, автоматичне переведення назв у верхній регістр.
4. Моніторинг активності користувачів: адміністратор бачить список усіх зареєстрованих користувачів та може змінювати їхній статус або роль.

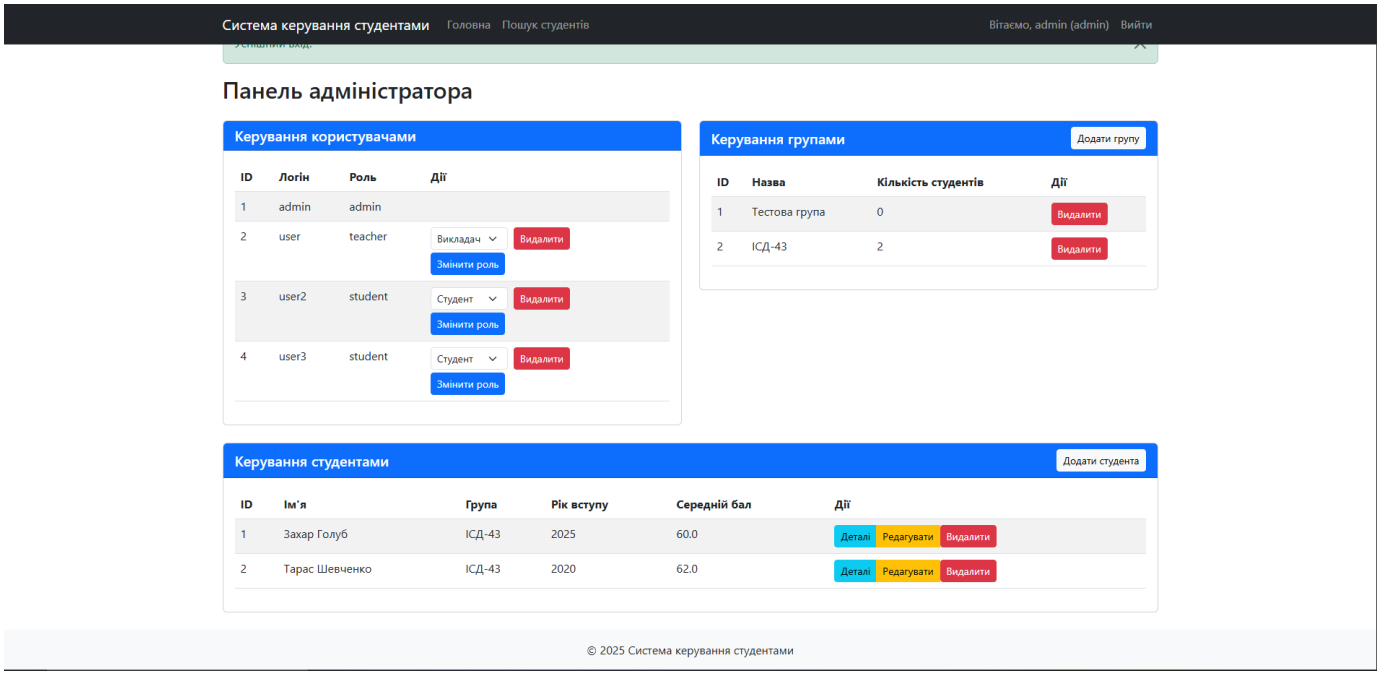


Рис. 2.2 Інтерфейс для ролі адміністратор

3. Панель викладача

Викладач має доступ до груп, які закріплені за ним, і функціонал для взаємодії зі студентами:

1. Перегляд списку студентів своєї групи.
2. Редагування характеристик студентів (наприклад, опис академічних досягнень чи поведінки).
3. Внесення або оновлення середнього балу.
4. Пошук студентів за ПІБ.

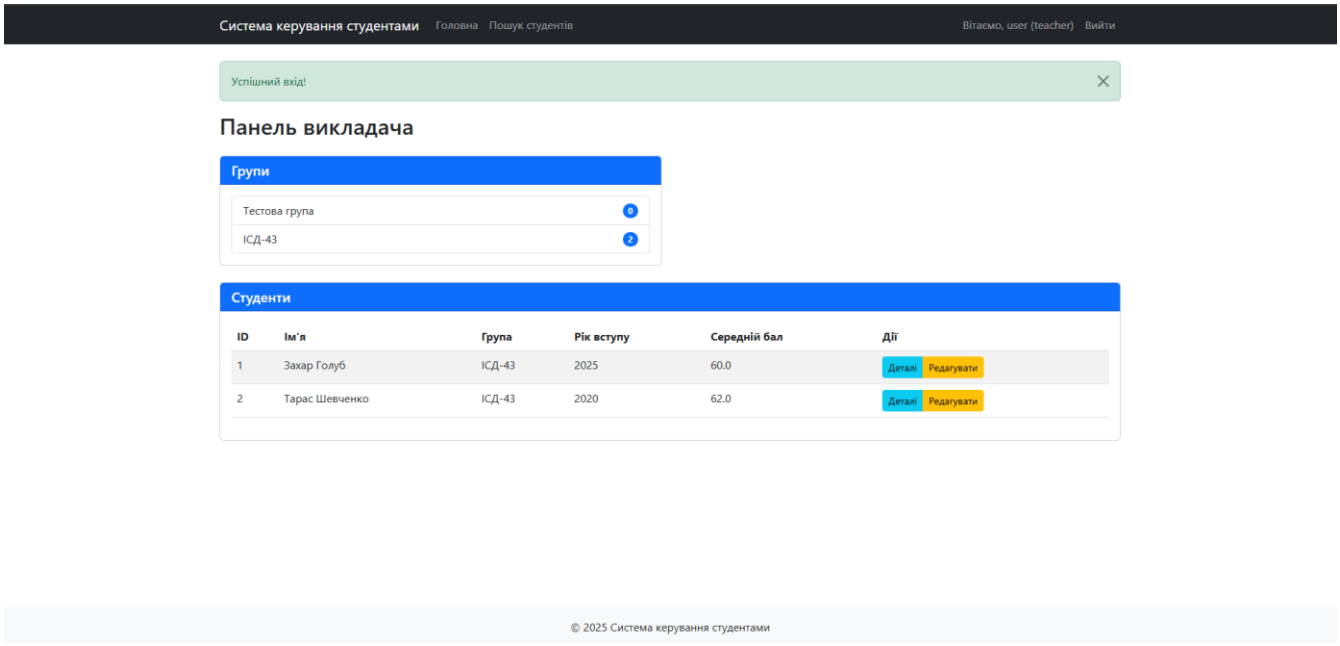


Рис. 2.3.Інтерфейс для користувача ролі викладач

4. Особистий кабінет студента

Кожен студент після входу бачить власні дані в особистому кабінеті:

1. Ім’я, прізвище, дата народження, група.
2. Середній бал, характеристика, які можуть оновлюватися викладачем.

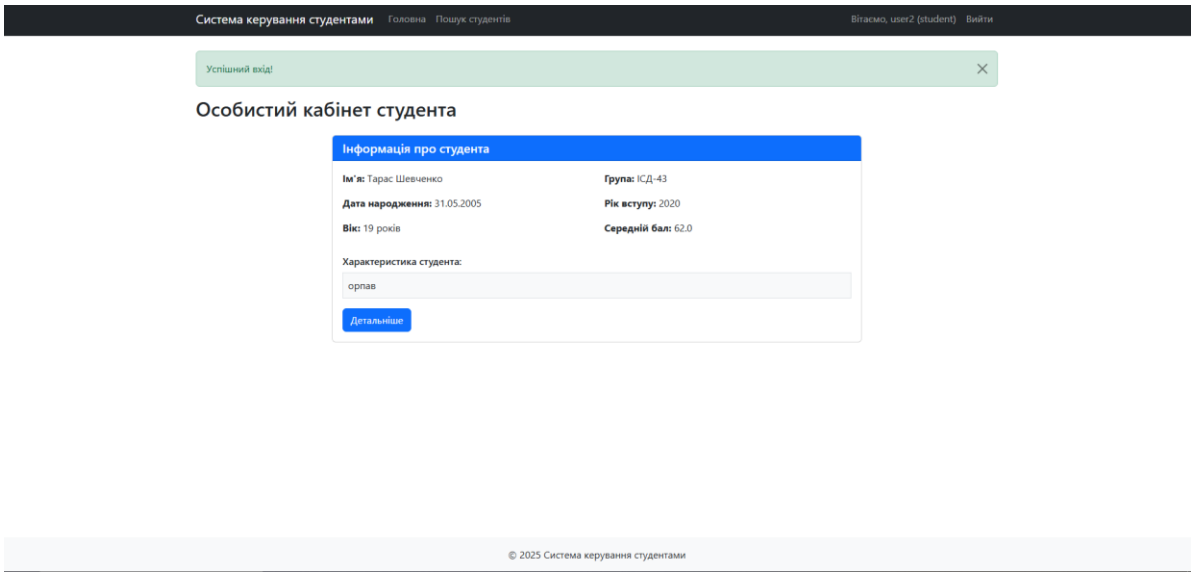


Рис. 2.4. Інтерфейс для ролі студент

5. Додавання та редагування студентів

Форма додавання студентів включає перевірку на обов’язкові поля, автоматичну нормалізацію введених даних (наприклад, формат ПІБ), а також сповіщення про успішне додавання через flash-повідомлення.

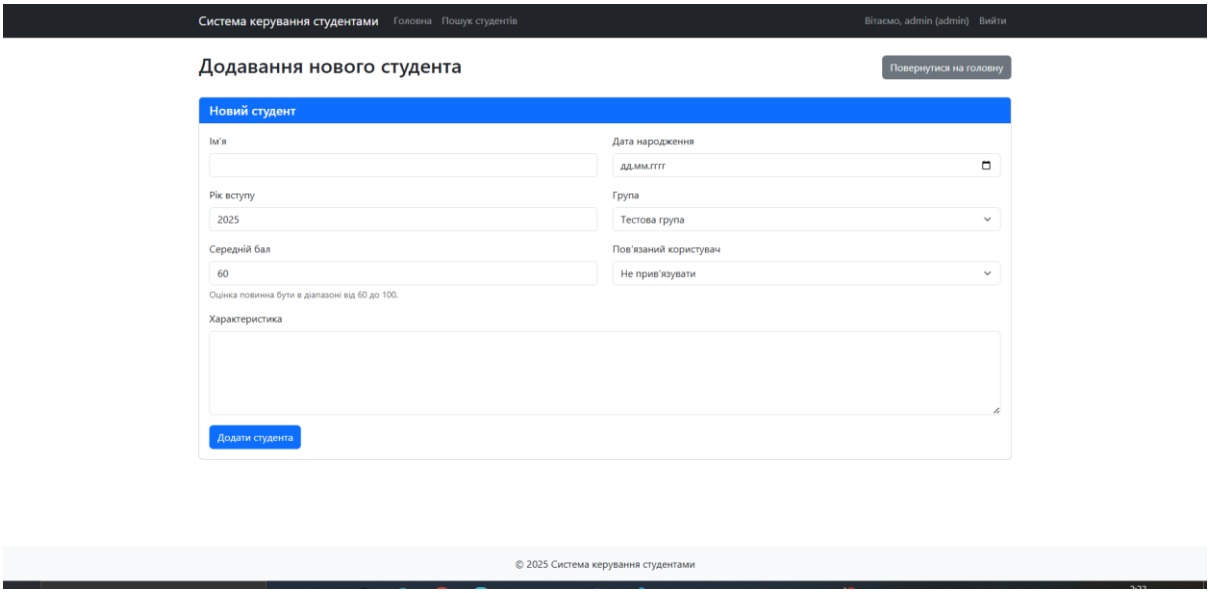


Рис. 2.5. Форма додавання студента

6. Додавання груп

Під час створення нової групи система перевіряє її унікальність. Якщо така група вже існує, з’являється відповідне повідомлення. Усі назви автоматично приводяться до верхнього регістру.

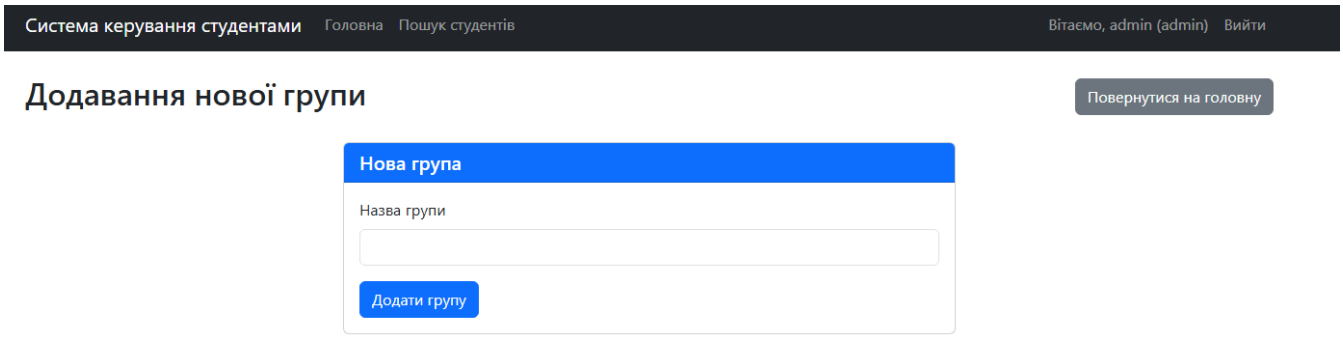


Рис. 2.6. Форма додавання груп

7. Пошук студентів

Реалізовано функціонал пошуку студентів за частковим або повним збігом імені чи прізвища. Це значно пришвидшує навігацію, особливо у великих базах даних.

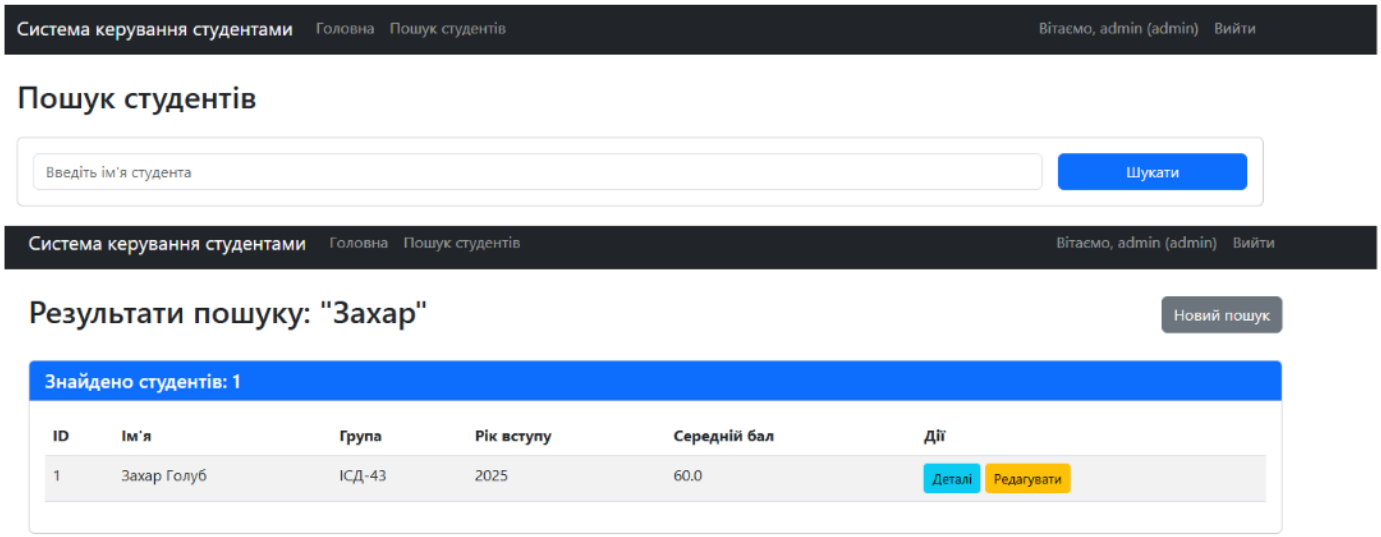


Рис. 2.7. Пошук студентів та результат пошуку

8. Детальний перегляд інформації про студента

При перегляді конкретного студента система виводить:

1. Особисті дані;
2. Академічні показники;
3. Характеристику;
4. Належність до групи;
5. Можливість редагування (залежно від ролі).

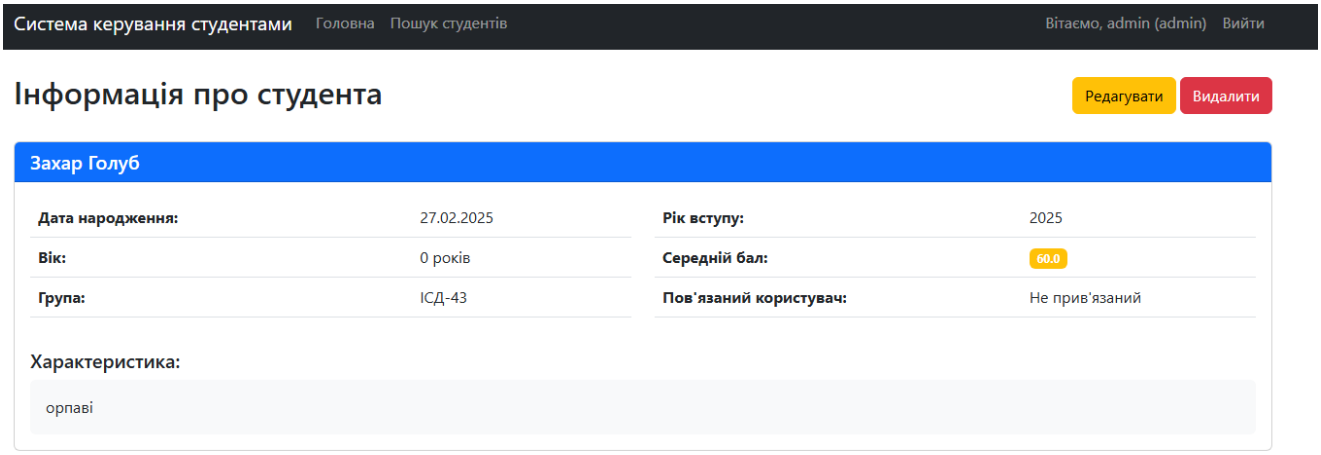


Рис. 2.8. Деталі студента

9. Адаптивність інтерфейсу

Завдяки використанню Bootstrap, веб-додаток повністю адаптивний. Інтерфейс коректно відображається на будь-яких пристроях — від комп’ютерів до смартфонів. Це дозволяє користувачам взаємодіяти з системою у будь-якому зручному для них середовищі.

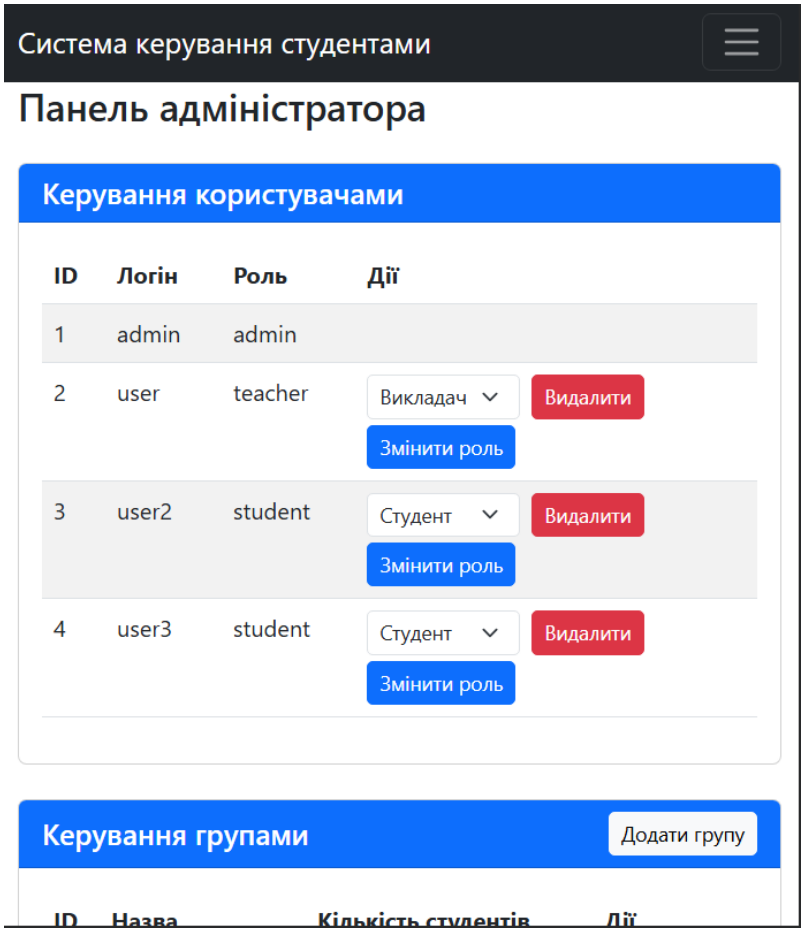


Рис. 2.9. Панель адміністратора з мобільного пристрою

10. Сповіщення та валідація

В усіх формах реалізовано валідацію введених даних, як на фронтенді (HTML5, Bootstrap alerts), так і на бекенді (Flask). Flash-повідомлення інформують користувачів про успішні або помилкові дії: додавання записів, видалення, зміни, неправильний логін, порожні поля тощо.

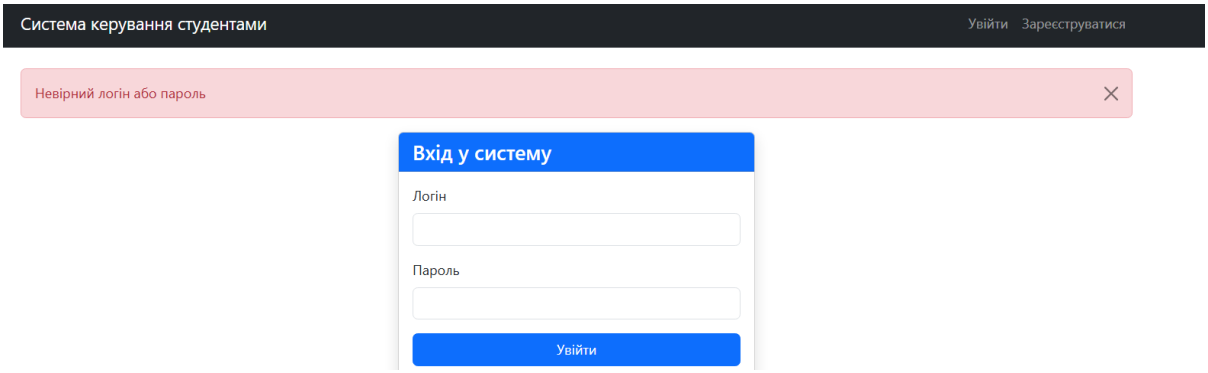


Рис. 2.10. Сповіщення при помилці входу

Можна констатувати що функціональні можливості веб-додатка, реалізованого в межах цього проєкту, охоплюють широкий спектр задач, пов’язаних з управлінням академічною інформацією студентів. Враховуючи вимоги сучасних навчальних закладів до цифровізації процесів, система надає всі базові інструменти, необхідні для ефективного ведення обліку студентів, супроводу їхньої навчальної діяльності, а також забезпечення прозорості та контрольованості даних. Завдяки підтримці кількох типів ролей користувачів — адміністратора, викладача та студента — створено багаторівневу систему доступу до функціоналу, що дозволяє зручно розподіляти повноваження та обов’язки між усіма учасниками освітнього процесу.

Користувачі мають змогу виконувати ключові операції — створення, перегляд, редагування, видалення та пошук студентів — у зручному форматі. Реалізовано також керування академічними групами, профілювання студентів із характеристиками, середніми балами, персональними даними. Це сприяє глибшому аналітичному підходу до управління академічною інформацією, адже всі дії систематизовані та доступні через логічно структуровані модулі.

Особливу увагу було приділено дизайну користувацького інтерфейсу, зокрема його адаптивності до мобільних пристроїв та зручності для

недосвідчених користувачів. Завдяки використанню сучасного фреймворку Bootstrap інтерфейс є легким, швидким у завантаженні, гнучким до розмірів екрана і не перевантажений зайвими елементами. Простота навігації дозволяє здійснювати найпоширеніші дії — пошук, додавання, редагування — за лічені секунди, не потребуючи глибоких технічних знань. Це особливо важливо у навчальному середовищі, де користувачами можуть бути не лише IT-фахівці, а й адміністративний персонал або викладачі з обмеженим досвідом користування інформаційними системами.

Крім реалізованого функціоналу, архітектура системи є відкритою до подальшого розвитку. У перспективі вона може бути доповнена модулями для створення електронних журналів, складання розкладу занять, фіксації відвідуваності, інтеграції з електронним документообігом, модульної системи оцінювання, автоматичного формування звітів успішності та інших корисних інструментів. Така гнучкість дозволяє навчальному закладу поступово масштабувати функціональність додатку відповідно до зростаючих потреб, не змінюючи при цьому базової структури або логіки системи.

Крім внутрішньої роботи, система потенційно може бути інтегрована із зовнішніми джерелами: державними освітніми реєстрами, платформами дистанційного навчання (Google Classroom, Moodle), хмарними сховищами для академічних матеріалів, а також системами комунікації зі студентами (наприклад, через Telegram-бот або електронну пошту). Це дозволить створити повноцінне цифрове середовище взаємодії, у якому усі учасники навчального процесу зможуть отримувати актуальну інформацію, залишати коментарі, отримувати сповіщення про зміни в системі — наприклад, про оновлені оцінки, зміни у розкладі чи призначення нового куратора групи.

Надійність системи забезпечується використанням сучасних засобів безпеки, таких як хешування паролів, захист від несанкціонованого доступу, обмеження прав користувача згідно з його роллю. Це дозволяє бути впевненими, що персональні дані студентів, а також службова інформація навчального закладу не опиняться у відкритому доступі чи не будуть змінені

сторонніми особами. Підтримка автентифікації користувачів, ведення сесій, а також можливість розширення авторизаційних механізмів (наприклад, через двофакторну ідентифікацію) роблять систему готовою до впровадження у реальному навчальному середовищі.

Важливою перевагою розробленого додатку є також його соціальна та економічна доцільність. Впровадження власного внутрішнього програмного рішення дозволяє навчальному закладу уникнути витрат на дорогі комерційні продукти, ліцензування та обслуговування. Окрім того, розробка подібного додатку дає змогу долучати студентів до практичного програмування в межах навчального процесу — у вигляді лабораторних, курсових або дипломних проєктів — що додатково стимулює розвиток IT-компетентностей серед здобувачів освіти.

Таким чином, реалізований веб-додаток є не лише демонстраційним прикладом роботи зі студентськими даними, а й повноцінною функціональною платформою, яка може бути впроваджена в освітнє середовище будь-якого навчального закладу. Його використання дозволяє значно зменшити навантаження на адміністративний персонал, оптимізувати обробку та аналіз даних, покращити якість освітнього процесу, а також створити основу для подальшої цифрової трансформації освітнього середовища з урахуванням викликів XXI століття.

3 РОЗРОБКА ДОДАТКУ

3.1 Загальна архітектура та функціональна структура веб-додатку

Для створення веб-додатку, призначеного для обліку студентів у закладах освіти, було обрано технологічний стек, що включає мову програмування Python, веб-фреймворк Flask, технології для фронтенду HTML і CSS, а також систему управління базами даних SQLite. Кожен компонент цього стеку був ретельно підібраний з урахуванням їхніх переваг, які забезпечують ефективну розробку, зручність використання та можливість адаптації до потреб проєкту. Нижче наведено детальне пояснення вибору кожного інструменту, розширене для збільшення обсягу тексту, без використання каскадної структури.

Мова програмування Python була обрана як основа для розробки завдяки своїм численним характеристикам, які роблять її оптимальною для такого роду проєктів. Python відомий своєю простотою у вивченні та високою читабельністю коду, що дозволяє розробникам швидко освоювати мову та створювати зрозумілі програми. Це знижує ймовірність помилок, що особливо важливо для освітніх проєктів, де можуть бути задіяні розробники-початківці. Python підтримує широкий набір бібліотек, таких як Flask для веб-розробки, SQLAlchemy для взаємодії з базами даних і Jinja2 для шаблонізації, що значно прискорює створення функціональних додатків. Завдяки своїй універсальності Python активно використовується в освіті, аналітиці даних і веб-розробці, що додає проєкту практичної цінності. Активна спільнота розробників і велика кількість доступних навчальних ресурсів, прикладів коду та документації дозволяють швидко вирішувати технічні питання, що виникають у процесі розробки. Ці особливості роблять Python ідеальним вибором для створення зрозумілого та надійного коду веб-додатку.

Для реалізації серверної частини було обрано веб-фреймворк Flask, який належить до категорії мікрофреймворків. Його ключова перевага полягає в

легкій і гнучкій структурі, що дозволяє створювати модульні додатки, які легко адаптувати до конкретних вимог проєкту. На відміну від більш складних фреймворків, таких як Django, Flask не накладає жорстких конвенцій чи надлишкових залежностей, що робить його ідеальним для проєктів із обмеженим обсягом, таких як розробки для освітніх цілей. Flask підтримує шаблонізацію через бібліотеку Jinja2, яка спрощує створення HTML-сторінок, дозволяючи вбудовувати Python-код для відображення динамічних даних. Інтеграція з ORM-системами, зокрема SQLAlchemy, забезпечує зручну роботу з базами даних, що є критично важливим для управління інформацією про студентів і групи. Завдяки мінімалістичному підходу Flask дозволяє розробникам зосередитися на реалізації основної логіки додатку, а не на налаштуванні складних компонентів, що підвищує ефективність і знижує поріг входження для нових учасників проєкту.

Для управління даними про студентів, групи, викладачів і ролі користувачів було обрано SQLite як основну систему управління базами даних. SQLite є легкою та зручною для локальної розробки, оскільки не потребує розгортання окремого серверного процесу. Усі дані зберігаються в одному файлі, що спрощує налаштування та тестування додатку. Це робить SQLite ідеальним вибором для студентських проєктів, де важлива простота використання та мінімальні вимоги до інфраструктури. Завдяки інтеграції з SQLAlchemy, яка забезпечує абстракцію для роботи з базою даних, розробники можуть легко створювати, оновлювати чи видаляти записи, використовуючи Python-об’єкти замість написання SQL-запитів. SQLite забезпечує достатню продуктивність для невеликих додатків, дозволяючи ефективно зберігати та обробляти дані про користувачів і їхні ролі.

Фронтенд-частина додатку базується на веб-технологіях HTML5 і CSS3. HTML5 використовується для створення структури сторінок, забезпечуючи семантичну розмітку, що полегшує підтримку коду та покращує доступність інтерфейсу для користувачів. CSS3 відповідає за стилізацію, дозволяючи створювати привабливий і сучасний дизайн, який адаптується до різних

розмірів екранів. Для прискорення розробки інтерфейсу застосовується фреймворк Bootstrap, який пропонує готові стилі та компоненти, такі як кнопки, форми чи таблиці, а також забезпечує адаптивну верстку. Bootstrap дозволяє створювати зручний і стандартизований інтерфейс із мінімальними зусиллями, що є важливим для проєктів із обмеженим часом на розробку. Поєднання HTML5, CSS3 і Bootstrap забезпечує створення функціонального та естетичного інтерфейсу, який відповідає сучасним стандартам веб-розробки.

Для організації процесу розробки було обрано Visual Studio Code як основне середовище розробки. Цей редактор коду є універсальним завдяки підтримці плагінів, автодоповненню коду та інтеграції з системами контролю версій, такими як Git. Visual Studio Code дозволяє налаштувати робоче середовище під потреби розробки, включаючи підтримку Python, HTML і CSS, що робить його зручним для всіх учасників проєкту. Для управління кодом використовується Git у поєднанні з платформою GitHub. Git забезпечує контроль версій, дозволяючи відстежувати зміни, повертатися до попередніх версій коду та координувати роботу кількох розробників. GitHub слугує централізованим сховищем, спрощуючи командну розробку, рецензування коду та вирішення конфліктів.

Архітектура додатку побудована на модульному принципі, що сприяє зрозумілості та легкості підтримки коду. Код організовано в окремі файли та директорії: маршрути (routes) відповідають за обробку HTTP-запитів, моделі (models) описують структуру бази даних, форми (forms) обробляють введення даних, шаблони (templates) містять HTML-файли, а конфігурації (config) визначають налаштування додатку. Така структура дозволяє швидко знаходити потрібні компоненти та вносити зміни. Права доступу розділені за ролями (адміністратор, викладач, студент), що забезпечує безпеку та персоналізацію даних. Для створення єдиного стилю інтерфейсу використовується базовий шаблон base.html, який містить спільні елементи,

такі як навігаційна панель чи футер, що зменшує дублювання коду та полегшує оновлення дизайну.

Обраний стек технологій забезпечує баланс між простотою розробки, функціональністю та можливістю адаптації до вимог проєкту. Python і Flask дозволяють швидко створювати серверну логіку, SQLite забезпечує зручне управління даними, а HTML, CSS і Bootstrap формують сучасний і зручний інтерфейс. Інструменти, такі як Visual Studio Code і Git, сприяють ефективній розробці, а модульна архітектура гарантує легкість підтримки та розширення. Ці інструменти є стандартними в веб-розробці, що підвищує цінність рішення та його відповідність потребам освітніх закладів.

3.2 Основний файл додатку app.py

Файл app.py є головним модулем веб-додатку, де описується конфігурація, маршрути, ініціалізація бази даних і реалізація основного функціоналу. У цьому підрозділі подано структуру коду та функціональні блоки з поясненнями.

3.2.1 Імпортування бібліотек та ініціалізація додатку

Цей блок коду включає набір імпортів бібліотек і модулів, які є основою для створення функціонального веб-додатку. Ці компоненти забезпечують широкий спектр можливостей, необхідних для реалізації всіх ключових аспектів роботи програми, починаючи від обробки запитів і закінчуючи взаємодією з базою даних. Розглянемо детальніше, які саме інструменти залучені та як вони сприяють розробці веб-додатку.

Перш за все, до складу імпортів входить бібліотека Flask, яка є центральним елементом додатку. Вона забезпечує базову структуру для створення веб-програми, дозволяючи обробляти запити, маршрутизувати їх і повертати відповіді. Зокрема, функція render_template використовується для відображення HTML-шаблонів, що дозволяє створювати динамічні веб-сторінки, які адаптуються до даних, отриманих від користувача чи бази даних.

Функція request відповідає за обробку HTTP-запитів, таких як GET або POST, що дає змогу отримувати дані з форм або параметрів URL. Для перенаправлення користувачів на інші сторінки застосовуються redirect і url_for, які спрощують навігацію в додатку, генеруючи правильні URL-адреси. Крім того, flash забезпечує систему повідомлень, що дозволяє інформувати користувачів про успішні дії чи помилки, а session допомагає зберігати тимчасові дані користувача між запитами, що є важливим для підтримки стану сесії.

Для реалізації системи авторизації в додатку використовується бібліотека Flask-Login, яка значно спрощує управління користувацькими сесіями. Зокрема, LoginManager відповідає за налаштування та керування процесом авторизації, забезпечуючи безпечний доступ до захищених сторінок. Функції login_user і logout_user дозволяють відповідно входити в систему або виходити з неї, а декоратор login_required захищає певні маршрути, дозволяючи доступ лише авторизованим користувачам. Об’єкт current_user забезпечує зручний доступ до даних поточного користувача, що полегшує персоналізацію вмісту чи перевірку прав доступу.

Безпека є ще одним важливим аспектом веб-додатку, і для цього в коді передбачено інструменти для роботи з паролями. Функція generate_password_hash дозволяє створювати захищені хеші паролів, що забезпечує їх безпечне зберігання в базі даних. У свою чергу, check_password_hash використовується для перевірки введених користувачем паролів, порівнюючи їх із збереженим хешем, що гарантує надійність процесу аутентифікації.

Окрім основних бібліотек, до коду включено додаткові утиліти, які розширюють функціональність додатку. Наприклад, модуль datetime дозволяє працювати з датами та часом, що може бути корисним для створення тимчасових міток, фільтрації даних чи відображення актуальної інформації. Модуль os забезпечує взаємодію з операційною системою, що може

знадобитися для роботи з файлами, шляхами чи іншими системними ресурсами.

Важливою частиною додатку є робота з базою даних, для чого використовуються ORM-моделі. Об’єкт db представляє підключення до бази даних, а моделі User, Student і Group відповідають за взаємодію з відповідними таблицями. Ці моделі дозволяють зручно створювати, оновлювати, видаляти чи отримувати дані, використовуючи об’єктно-орієнтований підхід, що значно спрощує роботу з базою даних.

Таким чином, набір імпортованих бібліотек і модулів формує міцну основу для створення повноцінного веб-додатку. Вони забезпечують усі необхідні інструменти для реалізації авторизації, обробки запитів, роботи з шаблонами, управління базою даних і забезпечення безпеки. Завдяки цьому розробники можуть зосередитися на створенні функціоналу, не витрачаючи зайвих зусиль на налаштування базових компонентів. Цей комплексний підхід робить код універсальним і придатним для реалізації різноманітних веб-проектів, від простих особистих сайтів до складних систем із великою кількістю користувачів і даних.

```
1  from flask import Flask, render_template, request, redirect, url_for, flash, session
2  from flask_login import LoginManager, login_user, logout_user, login_required, current_user
3  from werkzeug.security import generate_password_hash, check_password_hash
4  from datetime import datetime
5  from models import db, User, Student, Group
6  import os
```

Рис.3.1 Блок коду де імпортуються необхідні бібліотеки

3.2.2 Конфігурація Flask-додатку та ініціалізація компонентів

У цьому блоці коду виконується початкове налаштування веб-додатку на основі фреймворку Flask, що є критично важливим етапом для забезпечення його коректної роботи. Ці налаштування створюють фундамент, на якому будується вся подальша функціональність програми, включаючи обробку запитів, авторизацію користувачів і взаємодію з базою даних. Давайте

розберемо кожен компонент цього процесу детальніше, щоб зрозуміти, як саме він сприяє створенню надійного та ефективного веб-додатку.

На першому етапі створюється екземпляр Flask-додатку за допомогою конструкції Flask(name). Цей об’єкт є серцем програми, оскільки він відповідає за обробку всіх вхідних запитів, маршрутизацію та повернення відповідей. Використання name дозволяє Flask автоматично визначити розташування файлів шаблонів і статичних ресурсів відносно основного модуля проєкту, що робить структуру додатку більш організованою та зручною для розробки.

Далі відбувається конфігурація основних параметрів додатку, які визначають його поведінку та безпеку. Одним із ключових параметрів є SECRET_KEY, який використовується для захисту сесій користувачів і підпису токенів. Цей ключ має бути унікальним, складним і надійно захищеним, особливо в реальних проєктах, що розгортаються в продакшені. Неправильне або передбачуване значення цього ключа може призвести до вразливостей, таких як підробка сесій, тому в продакшені рекомендується генерувати його випадковим чином і зберігати в безпечному місці, наприклад, у змінних середовища.

Ще одним важливим аспектом конфігурації є налаштування підключення до бази даних за допомогою параметра SQLALCHEMY_DATABASE_URI. У цьому випадку використовується база даних SQLite, а шлях до неї вказує на файл students.db, розташований у теці проєкту. SQLite є легкою та зручною базою даних для невеликих проєктів або розробки, оскільки вона не потребує окремого серверного процесу й зберігає всі дані в одному файлі. Цей параметр забезпечує зв’язок між додатком і базою даних, дозволяючи виконувати операції з даними, такі як створення, оновлення чи видалення записів.

Для оптимізації роботи з базою даних також налаштовується параметр SQLALCHEMY_TRACK_MODIFICATIONS, який у цьому випадку вимикається. Ця опція відповідає за відстеження змін в об’єктах бази даних,

але її активація може створювати зайві сповіщення та споживати додаткові ресурси. Вимкнення цього параметра підвищує продуктивність додатку, що особливо важливо для проєктів із великою кількістю операцій із базою даних.

Наступним кроком є ініціалізація LoginManager, компонента бібліотеки Flask-Login, який відповідає за керування процесом авторизації користувачів. Цей об’єкт зв’язується з додатком за допомогою методу init_app(app), що дозволяє інтегрувати механізми авторизації в структуру Flask. Крім того, налаштовується параметр login_view = 'login', який указує, що сторінка входу в систему доступна за маршрутом, пов’язаним із функцією або роутом із назвою login. Це забезпечує автоматичне перенаправлення неавторизованих користувачів на сторінку логіну, якщо вони намагаються отримати доступ до захищених ресурсів.

Нарешті, активується підключення до бази даних через виклик db.init_app(app). Цей метод ініціалізує об’єкт SQLAlchemy, який забезпечує зручну роботу з базою даних за допомогою ORM (об’єктно-реляційного відображення). Завдяки цьому розробники можуть взаємодіяти з таблицями бази даних як із Python-об’єктами, що значно спрощує операції з даними та робить код більш читабельним і підтримувальним.

Ці базові налаштування є стандартними для більшості проєктів, побудованих на Flask, особливо якщо вони включають авторизацію користувачів і роботу з базою даних. Вони створюють міцну основу, яка дозволяє розробникам зосередитися на реалізації бізнес-логіки, не витрачаючи часу на повторне налаштування базових компонентів. Завдяки продуманій структурі та гнучкості Flask, ці параметри можна легко адаптувати до потреб конкретного проєкту, будь то невеликий прототип чи масштабна веб-система з великою кількістю користувачів.

```
app = Flask(__name__)
app.config['SECRET_KEY'] = 'your-secret-key'
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///students.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

login_manager = LoginManager()
login_manager.init_app(app)
login_manager.login_view = 'login'

db.init_app(app)
```

Рис. 3.2. Блок коду конфігурації Flask-додатку

3.2.3 Завантаження користувача та ініціалізація бази даних

У цьому блоці коду описано дві ключові функції — load_user та init_database, які відіграють важливу роль у забезпеченні коректної роботи Flask-додатку з авторизацією та базою даних. Ці функції взаємодіють із системою аутентифікації, моделями даних і базою даних, створюючи основу для функціонування веб-програми. Нижче подано розширений опис цих функцій, їхньої ролі та взаємодії, з додаванням деталей для збільшення обсягу тексту, але без використання каскадної структури.

Функція load_user є невід'ємною частиною системи авторизації, що реалізується за допомогою бібліотеки Flask-Login. Її основне завдання полягає в тому, щоб забезпечити можливість завантаження даних користувача за його унікальним ідентифікатором (ID) під час кожного HTTP-запиту. Це необхідно для того, щоб Flask-Login міг підтримувати стан авторизації користувача протягом усього сеансу роботи з додатком. Наприклад, коли користувач переходить між сторінками, система аутентифікації використовує цю функцію, щоб перевірити, чи є він авторизованим, і отримати доступ до його даних, таких як ім'я, роль або інші атрибути. Для пошуку користувача в базі даних функція використовує ORM-можливості бібліотеки SQLAlchemy, яка дозволяє виконувати запити до таблиці користувачів зручно та ефективно. Важливим аспектом є те, що ID користувача, який передається до функції у

вигляді рядка, конвертується в ціле число перед виконанням запиту. Це необхідно, оскільки в базі даних ідентифікатори зазвичай зберігаються у числовому форматі, і така конвертація забезпечує коректну взаємодію з моделлю даних. Якщо користувача з відповідним ID не знайдено, функція повертає None, що сигналізує системі про відсутність активного користувача. Цей механізм робить load_user критично важливою для підтримки безперервної авторизації та персоналізації вмісту в додатку.

Функція init_database, у свою чергу, відповідає за підготовку бази даних до роботи додатку. Вона виконує кілька ключових операцій, які забезпечують наявність необхідних таблиць і початкових даних для тестування чи демонстрації. На першому етапі функція викликає метод db.create_all(), який автоматично створює всі таблиці, визначені в ORM-моделях, таких як User і Group. Цей метод спрощує процес ініціалізації бази даних, оскільки розробнику не потрібно вручну писати SQL-запити для створення таблиць — SQLAlchemy робить це автоматично на основі структури моделей. Після створення таблиць функція додає обліковий запис адміністратора з логіном admin і паролем admin123, що є зручним для початкового тестування додатку. Для забезпечення безпеки пароль не зберігається в явному вигляді — він хешується за допомогою функції з бібліотеки Werkzeug, яка генерує захищений хеш, придатний для зберігання в базі даних. Це дозволяє захистити облікові дані адміністратора від несанкціонованого доступу, навіть якщо база даних буде скомпрометована. Крім того, функція створює тестову групу, яка може використовуватися для демонстрації роботи додатку, наприклад, для відображення списків студентів чи інших сутностей. Щоб уникнути дублювання даних, функція містить перевірки, які переконуються, що адміністратор або тестова група створюються лише за їхньої відсутності в базі даних. Такий підхід підвищує надійність і запобігає помилкам під час повторного виконання ініціалізації.

Обидві функції тісно пов’язані з моделями User і Group, які є основою для роботи з даними в базі. Функція load_user використовує модель User для

пошуку користувача за ID, тоді як init_database створює екземпляри моделей User і Group для наповнення бази даних. Завдяки SQLAlchemy ці операції виконуються на високому рівні абстракції, що дозволяє розробникам працювати з даними як із Python-об’єктами, а не з сирими SQL-запитами. Наприклад, створення нового користувача чи групи виглядає як ініціалізація об’єкта з подальшим збереженням його в базі даних через методи SQLAlchemy, такі як db.session.add() і db.session.commit(). Ця абстракція значно спрощує розробку та підвищує читабельність коду.

Взаємодія між load_user та init_database проявляється в їхній спільній меті — забезпечити безперебійну роботу системи авторизації та бази даних. Функція init_database створює початкове середовище, наповнюючи базу даних необхідними таблицями та даними, такими як обліковий запис адміністратора. У свою чергу, load_user використовує ці дані, щоб підтримувати авторизацію користувачів, дозволяючи системі Flask-Login ідентифікувати їх під час роботи з додатком. Разом ці функції формують цілісну систему, яка забезпечує як підготовку даних, так і їх використання для реалізації функціоналу авторизації.

Таким чином, функції load_user і init_database є фундаментальними компонентами Flask-додатку, які забезпечують його готовність до роботи та підтримку ключових можливостей, таких як авторизація та управління даними. Їхня інтеграція з SQLAlchemy та Flask-Login дозволяє створювати надійні та зручні веб-програми, які легко масштабуються та адаптуються до потреб проєкту. Завдяки продуманій структурі та чіткому розподілу обов’язків ці функції роблять розробку більш ефективною, дозволяючи зосередитися на створенні бізнес-логіки, а не на низькорівневих операціях із базою даних чи авторизацією.

```python
@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))

def create_tables():
    db.create_all()

    admin = User.query.filter_by(username='admin').first()
    if not admin:
        admin = User(
            username='admin',
            password_hash=generate_password_hash('admin123'),
            role='admin'
        )
        db.session.add(admin)

        test_group = Group.query.filter_by(name='Тестова група').first()
        if not test_group:
            test_group = Group(name='Тестова група')
            db.session.add(test_group)

        db.session.commit()
```

Рис. 3.3. Блок коду завантаження користувача та ініціалізація бази даних

3.2.4 Маршрути для автентифікації

Описаний блок коду реалізує основну логіку системи авторизації та навігації у Flask-додатку, включаючи перенаправлення, обробку входу, реєстрацію, вихід із системи та заходи безпеки. Ці компоненти забезпечують зручну взаємодію користувача з додатком, одночасно гарантуючи захист даних і стабільність роботи. Нижче подано розширений опис цієї системи з детальним поясненням кожного аспекту, без використання каскадної структури, із додаванням деталей для збільшення обсягу тексту.

Система перенаправлень у додатку побудована таким чином, щоб забезпечити інтуїтивно зрозумілу навігацію залежно від стану авторизації користувача. Головна сторінка, доступна за маршрутом /, виконує роль початкової точки взаємодії. Її логіка автоматично визначає, чи авторизований користувач. Якщо користувач уже увійшов у систему, він перенаправляється на сторінку dashboard, яка зазвичай містить персоналізований контент,

наприклад, список даних, доступних лише авторизованим користувачам. Якщо ж користувач не авторизований, його перенаправляють на сторінку входу (/login). Такий підхід спрощує взаємодію з додатком, оскільки користувач одразу потрапляє до релевантного розділу, без необхідності вручну обирати потрібну сторінку. Використання перенаправлень через функцію redirect і генерацію URL за допомогою url_for забезпечує гнучкість і захист від помилок у маршрутизації, оскільки URL створюються автоматично на основі назв функцій обробки маршрутів.

Логіка входу в систему, реалізована за маршрутом /login, є центральним елементом системи авторизації. Ця сторінка обробляє як GET-запити (для відображення форми входу), так і POST-запити (для обробки введених даних). Спочатку перевіряється, чи є користувач уже авторизованим. Якщо так, він перенаправляється на dashboard, щоб уникнути повторного входу. У разі POST-запиту додаток отримує дані з форми, зокрема логін і пароль, і виконує їх перевірку. Логін порівнюється з записами в базі даних за допомогою моделі User і ORM-можливостей SQLAlchemy. Якщо користувач із таким логіном існує, його пароль перевіряється за допомогою функції check_password_hash із бібліотеки Werkzeug. Ця функція порівнює введений пароль із захешованим паролем, збереженим у базі даних, що гарантує безпечну перевірку без розкриття оригінального пароля. У разі успішної автентифікації викликається функція login_user із Flask-Login, яка встановлює сесію користувача, після чого він перенаправляється на dashboard. Якщо ж логін або пароль некоректні, користувач отримує повідомлення про помилку через систему flash, яка відображає текстове сповіщення на сторінці. Такий механізм забезпечує зручність і прозорість взаємодії, інформуючи користувача про результат його дій.

Система реєстрації, доступна за маршрутом /register, призначена для створення нових облікових записів. Щоб уникнути дублювання функціоналу, доступ до цієї сторінки блокується для вже авторизованих користувачів — вони перенаправляються на dashboard. Форма реєстрації запитує логін, пароль

і підтвердження пароля. Перед створенням нового користувача додаток перевіряє унікальність логіна, виконуючи запит до бази даних через SQLAlchemy. Якщо логін уже зайнятий, користувач отримує flash-повідомлення з проханням обрати інший. Далі порівнюються введений пароль і його підтвердження, щоб переконатися, що користувач не припустився помилки під час введення. У разі успішної валідації пароль хешується за допомогою generate_password_hash і зберігається в базі даних разом із іншими даними користувача. За замовчуванням новому користувачу присвоюється роль student, що дозволяє легко розрізняти звичайних користувачів і адміністраторів у системі. Після успішної реєстрації користувач перенаправляється на сторінку входу, а flash-повідомлення інформує його про необхідність авторизації. Якщо валідація не пройшла (наприклад, паролі не збігаються), користувач отримує відповідне повідомлення про помилку. Цей підхід забезпечує надійність і зручність процесу реєстрації, мінімізуючи ризик створення некоректних облікових записів.

Вихід із системи, реалізований за маршрутом /logout, є простим, але захищеним процесом. Цей маршрут доступний лише авторизованим користувачам, що гарантується декоратором @login_required із Flask-Login. При зверненні до маршруту викликається функція logout_user, яка завершує сесію користувача, видаляючи його дані з контексту авторизації. Після цього користувач перенаправляється на сторінку входу, що дозволяє йому повторно авторизуватися або залишити додаток. Використання декоратора забезпечує захист від несанкціонованого доступу до цього маршруту, що є важливим для безпеки.

Безпека є одним із пріоритетів системи. Усі паролі перед збереженням у базі даних хешуються за допомогою generate_password_hash, що унеможливлює їх компрометацію навіть у разі витоку даних. Система flash-повідомлень використовується для інформування користувачів про успішні дії (наприклад, реєстрацію чи вхід) або помилки (наприклад, некоректний пароль чи зайнятий логін). Перевірка унікальності логінів запобігає створенню

дублікатів, що могло б призвести до плутанини чи конфліктів у системі. Обов'язкова перевірка підтвердження пароля під час реєстрації знижує ймовірність помилок, пов'язаних із неправильним введенням даних. Нарешті, використання декоратора @login_required для захисту маршрутів, таких як /dashboard або /logout, гарантує, що лише авторизовані користувачі можуть отримати доступ до чутливих функцій додатку.

Таким чином, описана система забезпечує повноцінну авторизацію, зручну навігацію та високий рівень безпеки. Логіка перенаправлень, входу, реєстрації та виходу з системи побудована так, щоб мінімізувати зусилля користувача, одночасно захищаючи його дані та додаток від потенційних загроз. Використання сучасних інструментів, таких як Flask-Login, SQLAlchemy і Werkzeug, дозволяє створювати надійні та ефективні веб-додатки, які легко масштабуються та адаптуються до потреб проєкту. Завдяки продуманій структурі та чіткому розподілу функціональності ця система є міцною основою для подальшої розробки складних веб-програм.

```
43  @app.route('/')
44  def index():
45      if current_user.is_authenticated:
46          return redirect(url_for('dashboard'))
47      return redirect(url_for('login'))
48
49  @app.route('/login', methods=['GET', 'POST'])
50  def login():
51      if current_user.is_authenticated:
52          return redirect(url_for('dashboard'))
53
54      if request.method == 'POST':
55          username = request.form['username']
56          password = request.form['password']
57
58          user = User.query.filter_by(username=username).first()
59
60          if user and check_password_hash(user.password_hash, password):
61              login_user(user)
62              flash('Успішний вхід!', 'success')
63              return redirect(url_for('dashboard'))
64          else:
65              flash('Невірний логін або пароль', 'danger')
66
67      return render_template('login.html')
68
```

Рис. 3.4. Маршрути аутентифікації у Flask-додатку

3.2.5 Маршрут головної панелі (dashboard)

Описаний блок коду реалізує систему захисту маршрутів, розподілу функціоналу за ролями користувачів, взаємодію з базою даних і особливості реалізації в рамках Flask-додатку. Ця система забезпечує чітке розділення прав доступу, ефективну роботу з даними та гнучкість для подальшого розвитку. Нижче подано розширений опис кожного аспекту з детальним поясненням, без використання каскадної структури, із додаванням деталей для збільшення обсягу тексту.

Система захисту маршрутів побудована на основі декоратора @login_required, який є частиною бібліотеки Flask-Login. Цей декоратор відіграє ключову роль у забезпеченні безпеки, дозволяючи доступ до певних сторінок або функцій лише авторизованим користувачам. Коли користувач намагається звернутися до захищеного маршруту, декоратор перевіряє, чи є активна сесія, тобто чи авторизований користувач. Якщо сесії немає, користувач автоматично перенаправляється на сторінку входу, визначену в налаштуваннях LoginManager (наприклад, маршрут /login). Крім того, для подальшої деталізації доступу використовується перевірка ролі поточного користувача через об’єкт current_user.role, доступний завдяки Flask-Login. Цей атрибут містить інформацію про роль користувача (наприклад, "адміністратор", "викладач" або "студент"), що дозволяє додатку динамічно визначати, який функціонал і які дані показувати. Такий підхід забезпечує не лише захист від неавторизованого доступу, а й точне налаштування прав залежно від типу користувача, що є важливим для створення безпечних і зручних веб-додатків.

Ролі користувачів у системі чітко визначені та мають різний рівень доступу до даних і функціоналу. Кожна роль відповідає певному типу користувача й асоціюється з унікальним шаблоном для відображення інтерфейсу. Адміністратор має найвищий рівень доступу: він може переглядати повний список усіх користувачів, студентів і груп, отримуючи доступ до всіх даних, що зберігаються в базі даних. Для цього

використовується шаблон admin_dashboard.html, який відображає всю доступну інформацію в структурованому вигляді, наприклад, у вигляді таблиць чи списків. Викладач має обмеженіший доступ порівняно з адміністратором: він може бачити списки студентів і груп, але не має доступу до повного списку користувачів. Його інтерфейс реалізовано через шаблон teacher_dashboard.html, який адаптовано для відображення даних, релевантних для викладацької діяльності, таких як списки студентів у певних групах. Студент, у свою чергу, має найбільш обмежений доступ: він може переглядати лише власні дані, які фільтруються за його унікальним ідентифікатором (user_id). Для цього застосовується шаблон student_dashboard.html, який відображає персоніфіковану інформацію, наприклад, оцінки, розклад або дані про групу студента. Такий розподіл ролей забезпечує чітке розділення обов'язків і даних, що підвищує безпеку та зручність використання додатку.

Взаємодія з базою даних здійснюється за допомогою ORM-механізмів бібліотеки SQLAlchemy, що дозволяє ефективно й зручно працювати з даними. Для отримання всіх записів із таблиць використовуються запити User.query.all(), Student.query.all() і Group.query.all(), які повертають списки всіх користувачів, студентів і груп відповідно. Ці методи є простими та універсальними, що робить їх ідеальними для використання в адміністративному інтерфейсі, де потрібен повний огляд даних. Для більш точкового пошуку, наприклад, щоб отримати дані конкретного студента, застосовується метод Student.query.filter_by(), який дозволяє фільтрувати записи за певними критеріями, такими як user_id. Це особливо корисно для студентів, яким потрібно відображати лише їхні персональні дані. Використання ORM спрощує написання запитів, оскільки розробнику не потрібно працювати з сирими SQL-командами — замість цього він оперує Python-об'єктами, що підвищує читабельність і підтримуваність коду. Крім того, SQLAlchemy оптимізує запити, зменшуючи навантаження на базу даних і забезпечуючи швидку обробку даних навіть у разі великих обсягів інформації.

Особливості реалізації системи підкреслюють її продуманість і готовність до подальшого розвитку. Чітке розділення прав доступу за ролями дозволяє легко контролювати, які дані та функції доступні кожному типу користувача, що є важливим для безпеки та зручності. Наприклад, студент не може випадково чи навмисно отримати доступ до даних іншого користувача, а викладач не має можливості змінювати налаштування, доступні лише адміністратору. Запити до бази даних оптимізовані завдяки використанню методів SQLAlchemy, які автоматично адаптуються до структури таблиць і мінімізують кількість звернень до бази. Використання різних шаблонів для кожної ролі (admin_dashboard.html, teacher_dashboard.html, student_dashboard.html) забезпечує гнучкість у дизайні інтерфейсу: кожен шаблон може бути налаштований відповідно до потреб конкретного типу користувача, що покращує користувацький досвід. Крім того, система спроектована з урахуванням можливості розширення. Наприклад, додавання нової ролі чи функціоналу (такого як редагування даних або створення звітів) може бути реалізовано шляхом додавання нових умов у логіку перевірки ролей і створення відповідних шаблонів, без необхідності змінювати базову архітектуру.

Таким чином, описана система захисту маршрутів, розподілу ролей і роботи з базою даних створює міцну основу для функціонального та безпечного Flask-додатку. Вона поєднує чітке розділення прав доступу, ефективну взаємодію з базою даних і гнучкість для подальшого розвитку, що робить її придатною як для невеликих проєктів, так і для складних веб-систем. Використання сучасних інструментів, таких як Flask-Login і SQLAlchemy, разом із продуманою структурою забезпечує надійність, безпеку та зручність у розробці й експлуатації додатку.

```python
# Головна панель
@app.route('/dashboard')
@login_required
def dashboard():
    if current_user.role == 'admin':
        users = User.query.all()
        students = Student.query.all()
        groups = Group.query.all()
        return render_template('admin_dashboard.html', users=users, students=students, groups=groups)
    elif current_user.role == 'teacher':
        students = Student.query.all()
        groups = Group.query.all()
        return render_template('teacher_dashboard.html', students=students, groups=groups)
    else:  # student
        student = Student.query.filter_by(user_id=current_user.id).first()
        return render_template('student_dashboard.html', student=student)

# Деталі студента
```

Рис. 3.5. Код головної панелі

3.2.6 Маршрути для роботи зі студентами

Маршрут перегляду деталей студента /student-details/<int:student_id>

Даний маршрут призначений для відображення повної інформації про конкретного студента системи. Він використовує динамічний URL, де student_id служить унікальним ідентифікатором запису в базі даних. Доступ до цього маршруту обмежений авторизованими користувачами завдяки застосуванню декоратора @login_required, що гарантує захист даних від несанкціонованого доступу.

При обробці запиту система спочатку виконує пошук студента в базі даних за вказаним ідентифікатором. Якщо запис не знайдено, автоматично генерується відповідь з кодом помилки 404, що інформує про відсутність такого студента. Для користувачів з роллю "студент" додатково виконується перевірка відповідності між ідентифікатором поточного користувача та ідентифікатором, пов'язаним з запитом студента. Це запобігає перегляду чужих персональних даних - при спробі такого доступу система перенаправляє користувача на головну панель з відповідним попередженням. Адміністратори та викладачі отримують повний доступ до перегляду інформації про будь-якого студента без обмежень. Для візуалізації даних використовується

спеціалізований шаблон student_details.html, який структуровано відображає всі доступні поля запису.

Маршрут пошуку студентів /search-students

Цей маршрут реалізує функціональність пошуку студентів за їх іменами з використанням текстового запиту. Він підтримує обидва основні типи HTTP-запитів: GET для відображення форми пошуку та POST для обробки введених даних і виведення результатів. Форма пошуку, яка відображається при GET-запиті, надає зручний інтерфейс для введення критеріїв пошуку. Після відправки форми (POST-запит) система виконує аналіз введеного тексту (search_term) та ініціює запит до бази даних з використанням умови contains, що відповідає операції LIKE в SQL. Це дозволяє знаходити студентів, в іменах яких міститься вказаний підрядок, незалежно від його позиції в рядку.

Результати пошуку систематизуються у вигляді списку та передаються до шаблону search_results.html для візуалізації. У разі відсутності збігів повертається порожній список, про що інформується користувач. Для відображення самої форми пошуку використовується окремий шаблон search_students.html. Важливо відзначити, що доступ до функції пошуку, як і до перегляду деталей, мають лише авторизовані користувачі, що забезпечує відповідний рівень конфіденційності даних.

Технічні аспекти реалізації

У основі функціоналу пошуку лежить ORM SQLAlchemy, який забезпечує безпечну та ефективну взаємодію з базою даних. Використання методу contains дозволяє генерувати оптимальні SQL-запити з умовою LIKE, що забезпечує гнучкість пошуку. Система обробки помилок інтегрована з механізмом flash-повідомлень Flask, що забезпечує зручне інформування користувача про будь-які проблеми або особливості виконання запиту. Технічна реалізація також передбачає обробку крайніх випадків, таких як відсутність результатів пошуку або спроба некоректного доступу, з відповідними реакціями системи.

```python
# Деталі студента
@app.route('/student-details/<int:student_id>')
@login_required
def student_details(student_id):
    student = Student.query.get_or_404(student_id)

    # Перевірка прав доступу
    if current_user.role == 'student' and student.user_id != current_user.id:
        flash('У вас немає прав доступу до цієї сторінки', 'danger')
        return redirect(url_for('dashboard'))

    return render_template('student_details.html', student=student)

# Пошук студентів
@app.route('/search-students', methods=['GET', 'POST'])
@login_required
def search_students():
    if request.method == 'POST':
        search_term = request.form['search_term']
        students = Student.query.filter(Student.name.contains(search_term)).all()
        return render_template('search_results.html', students=students, search_term=search_term)
    return render_template('search_students.html')

# Функціонал для вчителя
```

Рис. 3.6. Код маршрутів student-details і search-students

3.2.7 Функціонал керування студентами для адміністраторів та викладачів

Даний розділ описує функціонал системи для роботи з даними студентів, доступний викладачам та адміністраторам. Основний акцент зроблено на розмежуванні прав доступу між різними ролями користувачів.

Для адміністраторів реалізовано повний спектр можливостей з управління студентськими даними. Це включає створення нових записів про студентів, повне редагування всіх наявних полів (особиста інформація, навчальні дані, успішність), а також видалення записів з бази даних. Особливістю є можливість прив'язки студентських записів до акаунтів користувачів системи.

Викладачі мають обмежений доступ до функціоналу. Вони можуть переглядати дані студентів та редагувати лише окремі поля, пов'язані з навчальним процесом - середній бал та характеристику студента. Це дозволяє

викладачам вести академічні записи та оцінювати успішність, але обмежує доступ до особистої інформації.

Система пошуку студентів доступна обом ролям та реалізована через фільтрацію за іменем з частковим збігом. Результати пошуку відображаються у зручному табличному вигляді з можливістю подальшого переходу до повного профілю студента.

Технічна реалізація включає сувору валідацію вхідних даних, особливо для числових полів (наприклад, середнього балу). Для забезпечення цілісності даних реалізовано перевірки на унікальність зв'язків між сутностями. У разі помилок система надає зрозумілі повідомлення про причини відмови у виконанні операції.

Важливою особливістю є розмежування прав на рівні маршрутів - кожен запит містить перевірку ролі користувача перед виконанням операції. Це запобігає несанкціонованому доступу та змінам даних. Для зручності роботи всі форми редагування містять підказки та приклади коректногозаповнення полів.

```python
@app.route('/edit-student/<int:student_id>', methods=['GET', 'POST'])
@login_required
def edit_student(student_id):
    if current_user.role not in ['admin', 'teacher']:
        flash('У вас немає прав доступу до цієї сторінки', 'danger')
        return redirect(url_for('dashboard'))

    student = Student.query.get_or_404(student_id)

    if request.method == 'POST':
        if current_user.role == 'admin':
            # Адміністратор може редагувати всі дані
            student.name = request.form['name']
            student.birth_date = datetime.strptime(request.form['birth_date'], '%Y-%m-%d')
            student.enrollment_year = int(request.form['enrollment_year'])
            student.group_id = int(request.form['group_id'])

        # І вчитель, і адмін можуть редагувати оцінку та характеристику
        try:
            average_grade = float(request.form['average_grade'])
            if 60 <= average_grade <= 100:
                student.average_grade = average_grade
            else:
                flash('Оцінка повинна бути в діапазоні від 60 до 100', 'danger')
                return redirect(url_for('edit_student', student_id=student_id))
        except ValueError:
            flash('Оцінка повинна бути числом', 'danger')
            return redirect(url_for('edit_student', student_id=student_id))

        student.characteristic = request.form['characteristic']

        db.session.commit()
        flash('Дані студента оновлено', 'success')
        return redirect(url_for('student_details', student_id=student_id))

    groups = Group.query.all()
    return render_template('edit_student.html', student=student, groups=groups)
```

Рис. 3.7. Код функціоналу для вчителя

3.2.8 Функціонал керування групами та користувачами

Описаний розділ представляє функціонал веб-додатку для управління групами та обліковими записами користувачів, доступний виключно адміністраторам системи. Цей набір можливостей дозволяє ефективно керувати навчальними групами та ролями користувачів, забезпечуючи безпеку, надійність і зручність роботи. Нижче подано розширений опис функціоналу, технічної реалізації та особливостей інтерфейсу, без використання каскадної структури, із додаванням деталей для збільшення обсягу тексту.

Система управління групами надає адміністраторам повний набір операцій CRUD, що охоплює створення, читання, оновлення та видалення навчальних груп. Адміністратор може створювати нові групи, вказуючи їхні унікальні назви, які є ключовим ідентифікатором у системі. Перед додаванням

нової групи до бази даних система автоматично перевіряє, чи не існує вже група з такою назвою, що запобігає появі дублікатів і забезпечує цілісність даних. Для редагування групи адміністратор може змінювати її назву чи інші атрибути, якщо це необхідно. Видалення груп можливе лише за умови, що до них не прив’язані студенти. Ця перевірка є важливим механізмом захисту, оскільки видалення групи зі студентами могло б призвести до втрати пов’язаних даних, таких як записи про студентів чи їхні оцінки. Такий підхід гарантує, що система залишається консистентною, а адміністратор отримує чітке повідомлення, якщо операцію видалення неможливо виконати через наявність залежностей.

Функціонал управління обліковими записами користувачів дозволяє адміністраторам виконувати низку важливих операцій. Однією з ключових можливостей є зміна ролей користувачів, що дає змогу призначати або змінювати ролі, такі як адміністратор, викладач чи студент. Це дозволяє гнучко налаштовувати права доступу, наприклад, підвищувати студента до викладача або призначати нового адміністратора. Крім того, адміністратор може повністю видаляти облікові записи, якщо вони більше не потрібні. При видаленні система автоматично синхронізує пов’язані дані, наприклад, видаляючи записи студента, асоційовані з обліковим записом, щоб уникнути залишкових або неконсистентних даних у базі. Цей процес забезпечує цілісність системи та запобігає накопиченню застарілої інформації.

Особлива увага в системі приділена захисту від конфліктних ситуацій, які можуть виникнути під час виконання адміністративних операцій. Наприклад, адміністратору заборонено змінювати власну роль, щоб уникнути випадкової втрати адміністративних прав, що могло б заблокувати доступ до ключових функцій. Аналогічно, система не дозволяє адміністратору видаляти власний обліковий запис, що є важливим заходом безпеки для запобігання саморуйнування. При спробі видалити групу, до якої прив’язані студенти, операція блокується, а адміністратор отримує повідомлення про необхідність спочатку перенести або видалити студентів. Перевірка унікальності назв груп

під час їх створення виключає можливість появи дублікатів, що могло б спричинити плутанину в системі. Ці захисні механізми роблять систему стійкою до помилок і підвищують її надійність.

Технічна реалізація функціоналу спроектована з урахуванням сучасних практик розробки. Для видалення даних використовується каскадне видалення, що забезпечує автоматичне очищення пов’язаних записів у базі даних, наприклад, при видаленні облікового запису студента разом із ним видаляються асоційовані записи. Усі операції, що впливають на базу даних, виконуються в рамках транзакцій, що гарантує атомарність змін: якщо якась частина операції не вдається, усі зміни відкочуються, щоб зберегти консистентність даних. Для відстеження дій адміністратора реалізовано деталізоване логування, яке фіксує всі ключові операції, такі як створення чи видалення груп і користувачів. Це дозволяє адміністраторам чи розробникам аналізувати історію змін у разі виникнення проблем. Валідація вхідних параметрів, таких як назви груп чи дані користувачів, виконується на сервері перед збереженням у базу даних, що запобігає введенню некоректних даних. Після кожної операції адміністратор отримує інформативне повідомлення через систему flash, яке пояснює, чи була операція успішною, чи виникла помилка, наприклад, через порушення унікальності назви групи.

Інтерфейс системи управління групами та користувачами розроблений з акцентом на зручність і інтуїтивність. Адміністратори взаємодіють із системою через веб-форми, які містять підказки та чіткі інструкції, що допомагають уникнути помилок під час введення даних. Наприклад, форма створення групи може включати поле для назви з приміткою про необхідність унікальності, а форма зміни ролі користувача — випадаючий список із доступними ролями. Для підвищення ефективності передбачено можливість масового оновлення даних, наприклад, зміни ролей для кількох користувачів одночасно або оновлення інформації про групу для кількох записів. Такий підхід економить час адміністратора, особливо в системах із великою кількістю користувачів чи груп. Інтерфейс побудований на основі шаблонів

Jinja2, що дозволяє легко адаптувати його вигляд і функціонал до потреб конкретного закладу освіти.

Загалом, система управління групами та обліковими записами забезпечує адміністраторам потужний і безпечний інструмент для виконання їхніх обов’язків. Повний цикл CRUD-операцій для груп, гнучке управління ролями користувачів і синхронізація даних створюють надійну основу для роботи з навчальними даними. Захисні механізми, такі як блокування самовидалення чи перевірка унікальності, мінімізують ризик помилок і конфліктів. Технічна реалізація з транзакціями, логуванням і валідацією даних відповідає стандартам сучасної веб-розробки, а інтуїтивний інтерфейс із формами та масовими операціями робить систему зручною для повсякденного використання. Завдяки модульній структурі та продуманому дизайну цей функціонал легко розширювати, наприклад, додаванням нових типів ролей чи додаткових перевірок, що робить його придатним для використання в різних освітніх контекстах.

```python
@app.route('/delete-student/<int:student_id>', methods=['POST'])
@login_required
def delete_student(student_id):
    if current_user.role != 'admin':
        flash('У вас немає прав доступу до цієї операції', 'danger')
        return redirect(url_for('dashboard'))

    student = Student.query.get_or_404(student_id)
    db.session.delete(student)
    db.session.commit()

    flash('Студент видалений', 'success')
    return redirect(url_for('dashboard'))

@app.route('/change-role/<int:user_id>', methods=['POST'])
@login_required
def change_role(user_id):
    if current_user.role != 'admin':
        flash('У вас немає прав доступу до цієї операції', 'danger')
        return redirect(url_for('dashboard'))

    if current_user.id == user_id:
        flash('Ви не можете змінити свою власну роль', 'danger')
        return redirect(url_for('dashboard'))

    user = User.query.get_or_404(user_id)
    new_role = request.form['role']

    if new_role not in ['admin', 'teacher', 'student']:
        flash('Недійсна роль', 'danger')
        return redirect(url_for('dashboard'))

    user.role = new_role
    db.session.commit()

    flash(f'Роль користувача змінена на {new_role}', 'success')
    return redirect(url_for('dashboard'))
```

Рис. 3.8. Код видалення студента та зміни ролі користувача

3.2.9 Ініціалізація та запуск веб-додатку

На заключному етапі розробки системи реалізовано комплексний механізм ініціалізації та запуску веб-додатку, який забезпечує стабільну роботу всіх компонентів системи. Цей процес включає низку важливих етапів, кожен з яких виконує визначені функції для коректного старту програми.

Першим критично важливим етапом є ініціалізація бази даних. Під час першого запуску системи автоматично виконується створення всіх необхідних таблиць у відповідності до визначених ORM-моделей. Це забезпечує

готовність бази даних до роботи без необхідності ручного створення структури. Одночасно відбувається наповнення системи початковими тестовими даними, серед яких обов'язково створюється стандартний обліковий запис адміністратора з заданими параметрами доступу. Додатково можуть додаватись тестові групи та інші сутності, необхідні для первинного тестування функціоналу.

Важливою складовою процесу ініціалізації є налаштування робочого середовища додатку. Тут виконується конфігурація всіх основних параметрів роботи системи, включаючи налаштування режиму відлагодження, який особливо важливий на етапі розробки. Встановлюються всі необхідні з'єднання з зовнішніми сервісами та ресурсами, ініціалізуються підключені модулі та бібліотеки. Особлива увага приділяється налаштуванню системи безпеки, включаючи ініціалізацію механізмів шифрування та автентифікації.

Фінальним етапом є безпосередній запуск веб-сервера, який забезпечує обробку вхідних запитів від користувачів. Стандартна реалізація передбачає використання вбудованого сервера Flask, який є достатнім для етапів розробки та тестування. Для промислового використання система може бути легко інтегрована з більш потужними production-серверами типу Gunicorn чи uWSGI. Під час запуску виконується комплексна перевірка готовності всіх компонентів системи до роботи, включаючи доступність бази даних, наявність необхідних ресурсів та коректність конфігураційних параметрів.

Технічна реалізація процесу ініціалізації включає низку важливих особливостей. Використання контексту додатку забезпечує правильний порядок ініціалізації залежних компонентів. Реалізовано розмежування робочих режимів (розробка/тестування/експлуатація), що дозволяє адаптувати поведінку системи до конкретного середовища виконання. Система автоматично перевіряє та створює необхідні каталоги для зберігання тимчасових файлів, завантажень та інших ресурсів.

Особливу увагу приділено обробці помилок ініціалізації. Система фіксує всі критичні події під час запуску в спеціальному лог-файлі, що значно

спрощує діагностику потенційних проблем. Реалізовано контроль версій бази даних, який дозволяє відстежувати зміни в структурі та автоматично застосовувати необхідні міграції. Виконується комплексна перевірка наявності всіх залежностей та їх відповідність необхідним версіям.

Запуск системи здійснюється через головний модуль, який інкапсулює всю логіку ініціалізації та забезпечує уніфікований інтерфейс для старту додатку. Це дозволяє легко адаптувати процес запуску під різні середовища виконання та варіанти розгортання. Для зручності розробників передбачено можливість додаткового налаштування параметрів запуску через конфігураційні файли або змінні середовища.

```
358  with app.app_context():
359      create_tables()
360
361  if __name__ == '__main__':
362      app.run(debug=True)
```

Рис. 3.9 Код запуску додатку

3.3 Проєктування бази даних

Проєктування бази даних є одним із фундаментальних етапів створення будь-якої інформаційної системи, особливо якщо мова йде про системи, які працюють з великими обсягами структурованої інформації та вимагають високої надійності, узгодженості даних і можливості масштабування. У межах реалізації веб-додатку для обліку студентів навчальних закладів база даних виступає центральним елементом архітектури, що об'єднує усі функціональні модулі системи — від реєстрації та аутентифікації користувачів до керування групами, характеристиками студентів і правами доступу.

Створення ефективної структури бази даних передбачає не лише логічне групування даних у таблиці, а й розуміння реальних бізнес-процесів, які ця база повинна підтримувати. У випадку з навчальним закладом такими процесами є: реєстрація нових студентів, закріплення їх за навчальними

групами, призначення викладачів, збереження академічної інформації, адміністрування доступу та контроль над правами користувачів.

Основні вимоги до бази даних:

Для досягнення вищезазначених цілей до бази даних було висунуто низку функціональних і нефункціональних вимог, зокрема:

Надійне зберігання особистої інформації студентів і викладачів, включаючи дані про народження, успішність, характеристики та належність до академічних груп;

Гарантована унікальність і цілісність даних — наприклад, унікальність логінів користувачів, коректність посилань між таблицями (через зовнішні ключі);

Підтримка ролей користувачів з можливістю розмежування доступу відповідно до їх повноважень;

Гнучкість та масштабованість структури, яка дозволить легко додати нові поля, сутності або функціональні зв’язки без руйнування поточної логіки;

Оптимізація для читання і запису, оскільки переважна більшість операцій у такій системі є транзакційними — реєстрація, зміна даних, пошук, перегляд;

Мінімізація дублювання інформації завдяки нормалізованій структурі;

Безпечність зберігання чутливих даних, зокрема паролів користувачів (з використанням хешування).

Ключові сутності бази даних

У розробленій моделі бази даних виділено кілька основних сутностей, які утворюють каркас всієї системи. Кожна з них відповідає певній реальній ролі або об’єкту у процесі функціонування закладу освіти.

1. User (Користувач)

Центральна таблиця, що зберігає облікові записи всіх користувачів — незалежно від того, чи є вони студентами, викладачами чи адміністраторами.

Основні поля:

id: первинний ключ (унікальний ідентифікатор);

username: логін користувача (унікальний);

password_hash: хешований пароль;

role: текстова роль (admin / teacher / student).

Ця таблиця лежить в основі системи авторизації та дозволяє керувати правами доступу, визначаючи, які дії доступні конкретному користувачу.

2. Student (Студент)

Містить детальну інформацію про студентів, у тому числі особисті й навчальні показники.

Основні поля:

id, name, surname, birth_date;

average_grade: середній бал студента;

description: характеристика (наприклад, «активний, бере участь у конференціях»);

group_id: зовнішній ключ, що вказує на приналежність до конкретної навчальної групи;

user_id: посилання на відповідного запис у таблиці користувачів (User).

3. Group (Група)

У таблиці groups зберігаються назви всіх академічних груп, до яких належать студенти.

Основні поля:

id: первинний ключ;

name: назва групи (унікальна, автоматично зберігається у верхньому регістрі для уніфікації).

4. Teacher (Викладач)

Описує викладачів, які мають змогу взаємодіяти з групами студентів.

Основні поля:

id, full_name;

user_id: зовнішній ключ до users.

5. GroupAssignment (Призначення груп викладачам)

Це службова таблиця, яка реалізує зв’язок багато-до-багатьох між викладачами та навчальними групами.

Основні поля:

id, teacher_id, group_id.

Ця таблиця дозволяє призначити одного викладача до кількох груп, або кількох викладачів до однієї групи — наприклад, у випадку командного викладання чи змінного навантаження.

Схема зв’язків між таблицями

Загальна логіка структури бази даних демонструє такі типи зв’язків:

Один-до-одного — між users і students, users і teachers (через user_id);

Один-до-багатьох — між groups і students;

Багато-до-багатьох — між teachers і groups, реалізований через проміжну таблицю group_assignments.

Ця структура повністю задовольняє вимоги системи до зберігання та обробки даних, дозволяє швидко здійснювати вибірки, фільтрування, агрегацію та статистичний аналіз.

Діаграма структури бази даних

Для наочності на рисунку 3.10 наведено ER-діаграму, що відображає логічну модель бази даних. На ній візуалізовано таблиці, ключові поля, типи зв’язків та зовнішні ключі, які поєднують таблиці між собою.

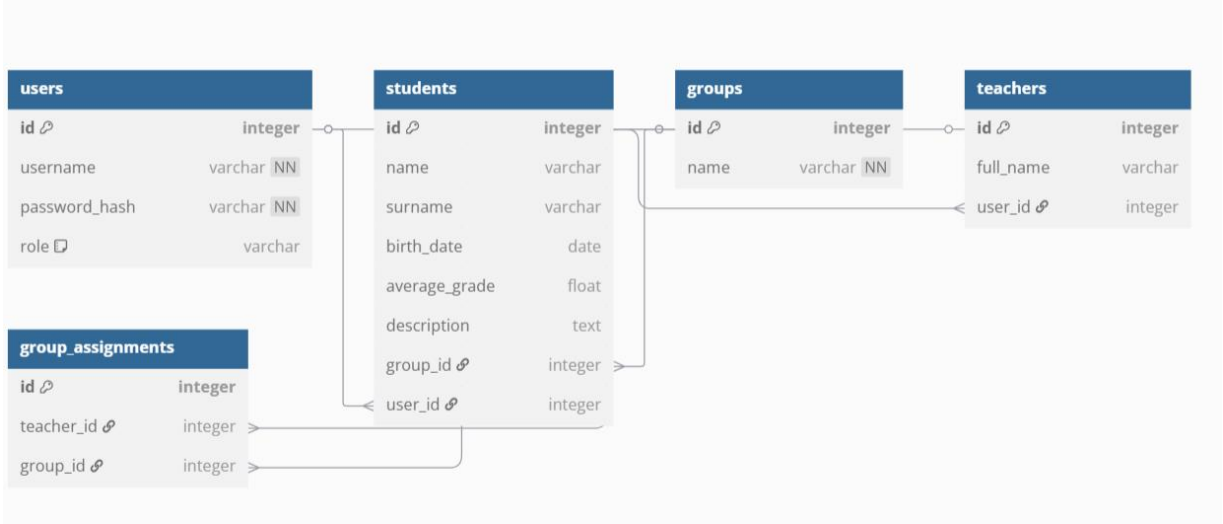


Рис. 3.10 – ER-діаграма структури бази даних веб-додатку

Забезпечення цілісності та захисту даних

Забезпечення надійності та цілісності бази даних — це ключовий компонент будь-якої системи, що працює з персональними або освітніми даними. У розробленій системі були реалізовані такі механізми:

Встановлення зовнішніх ключів між таблицями, що гарантує логічну узгодженість даних (наприклад, кожен студент має існуючу групу);

Обмеження NOT NULL для обов’язкових полів, що зменшує ризик появи "порожніх" записів;

Автоматичне перетворення назв груп у верхній регістр, що забезпечує консистентність найменувань;

Унікальність логінів користувачів і назв груп, що унеможливлює дублювання;

Хешування паролів за допомогою бібліотеки Werkzeug у Flask — це захищає від витоку паролів навіть при доступі до бази сторонніми особами;

Можливість масштабування — структура дозволяє у майбутньому додати, наприклад, таблиці відвідуваності, розкладу, оцінювання, журналів змін.

Висновок до підрозділу

Таким чином, база даних, реалізована в рамках даного дипломного проєкту, є повноцінним і гнучким інструментом для зберігання та обробки освітньої інформації. Вона відповідає як сучасним вимогам до безпеки та структурованості, так і практичним потребам навчального закладу — надаючи можливість централізованого управління всіма ключовими об’єктами: студентами, викладачами, групами, ролями. Застосування нормалізованої реляційної моделі, наявність чітких зовнішніх зв’язків і використання сучасних методів захисту даних дозволяє гарантувати не лише стабільну роботу системи сьогодні, а й її потенціал до подальшого розширення в майбутньому. Проєктована структура бази є масштабованою та легко адаптується до потреб як малих, так і великих освітніх закладів.

ВИСНОВКИ

Розробка веб-додатку для обліку студентів навчальних закладів, представлена в даній кваліфікаційній роботі, є відповіддю на сучасні виклики цифрової трансформації освіти. Проведений аналіз предметної області показав, що традиційні методи управління академічними даними, такі як паперова документація чи застарілі програмні засоби, не відповідають вимогам ефективності, масштабованості та безпеки. Існуючі системи, такі як PowerSchool SIS, eSckooly чи Educationalist, хоча й пропонують базовий функціонал, мають обмеження в адаптивності до потреб українських ВНЗ, зокрема через складність впровадження, орієнтацію на шкільну освіту чи відсутність гнучкості в налаштуванні.

Метою роботи було створення веб-додатку, який забезпечує централізоване зберігання, швидкий доступ і ефективну обробку академічних даних студентів. Для реалізації цього завдання було обрано оптимальний стек технологій: фреймворк Flask як серверна основа, SQLite як легка СУБД, SQLAlchemy для роботи з базою даних, Bootstrap для створення адаптивного інтерфейсу та Jinja2 для шаблонізації. Вибір цих інструментів обґрунтовано їхньою простотою, гнучкістю та відповідністю масштабам проєкту, що дозволило швидко розробити прототип із можливістю подальшого розширення.Проєктування бази даних стало ключовим етапом роботи. Розроблена структура включає п’ять основних таблиць (User, Student, Group, Teacher, GroupAssignment), які відображають основні сутності навчального процесу та підтримують зв’язки один-до-одного, один-до-багатьох і багато-до-багатьох. Використання зовнішніх ключів, обмежень унікальності та хешування паролів забезпечує цілісність і безпеку даних. ER-діаграма, представлена в роботі, наочно ілюструє логіку зв’язків, що полегшує розуміння структури для розробників і адміністраторів. Реалізований веб-додаток підтримує базові та розширені

функції, включаючи авторизацію користувачів, розмежування доступу за ролями (адміністратор, викладач, студент), пошук, додавання, редагування та видалення студентів і груп, а також управління призначеннями викладачів. Функціонал протестовано на предмет коректності роботи модулів, безпеки обробки даних і зручності інтерфейсу. Результати тестування підтвердили стабільність системи та її готовність до практичного використання.

Наукова новизна роботи полягає в інтеграції сучасних технологій веб-розробки для створення гнучкого рішення, адаптованого до потреб українських навчальних закладів. Практична цінність додатку полягає в його здатності автоматизувати рутинні процеси, зменшувати адміністративне навантаження та підвищувати прозорість освітнього процесу. Система є модульною та масштабованою, що дозволяє додавати нові функції, такі як облік відвідуваності чи інтеграція з платформами дистанційного навчання.

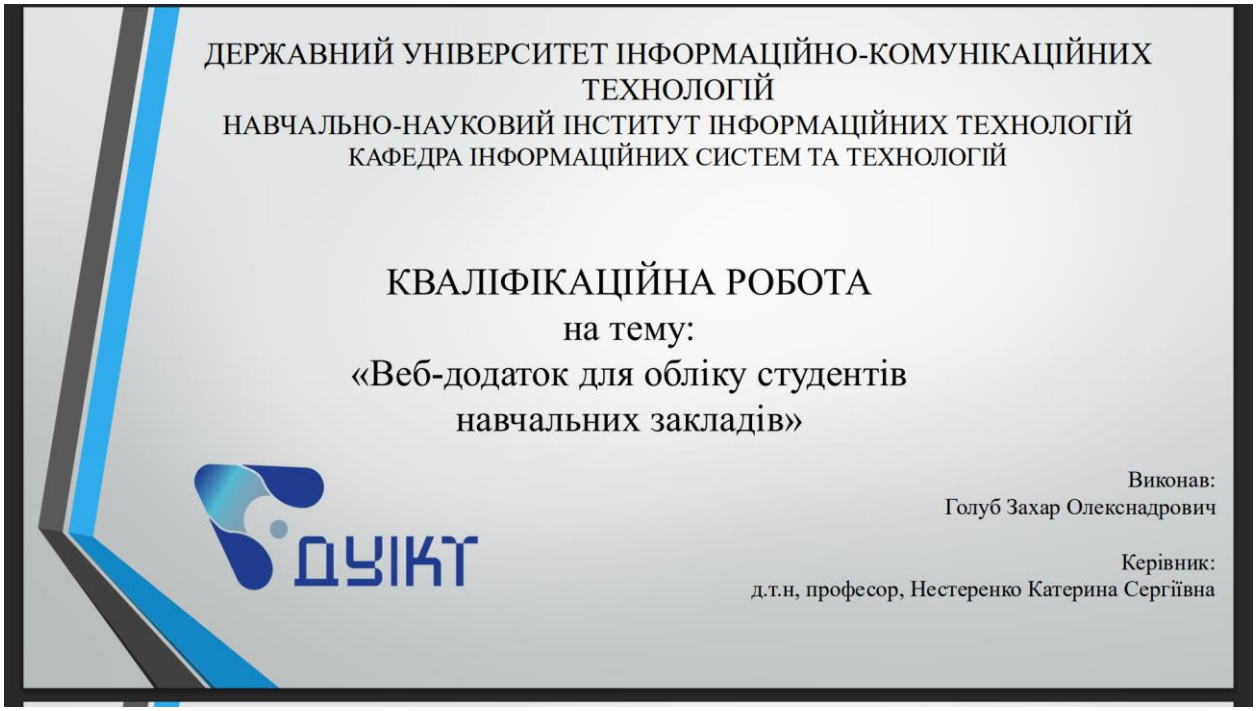
Перспективи розвитку додатку включають перехід на хмарні технології для підвищення доступності, додавання аналітичних модулів для прогнозування успішності студентів і автоматичного формування звітів, а також інтеграцію з державними освітніми реєстрами чи зовнішніми сервісами, такими як бази практик чи HR-платформи. Впровадження системи в реальних навчальних закладах сприятиме підвищенню ефективності управління академічними даними та формуванню основи для створення "розумного університету", що відповідає сучасним тенденціям цифровізації освіти.

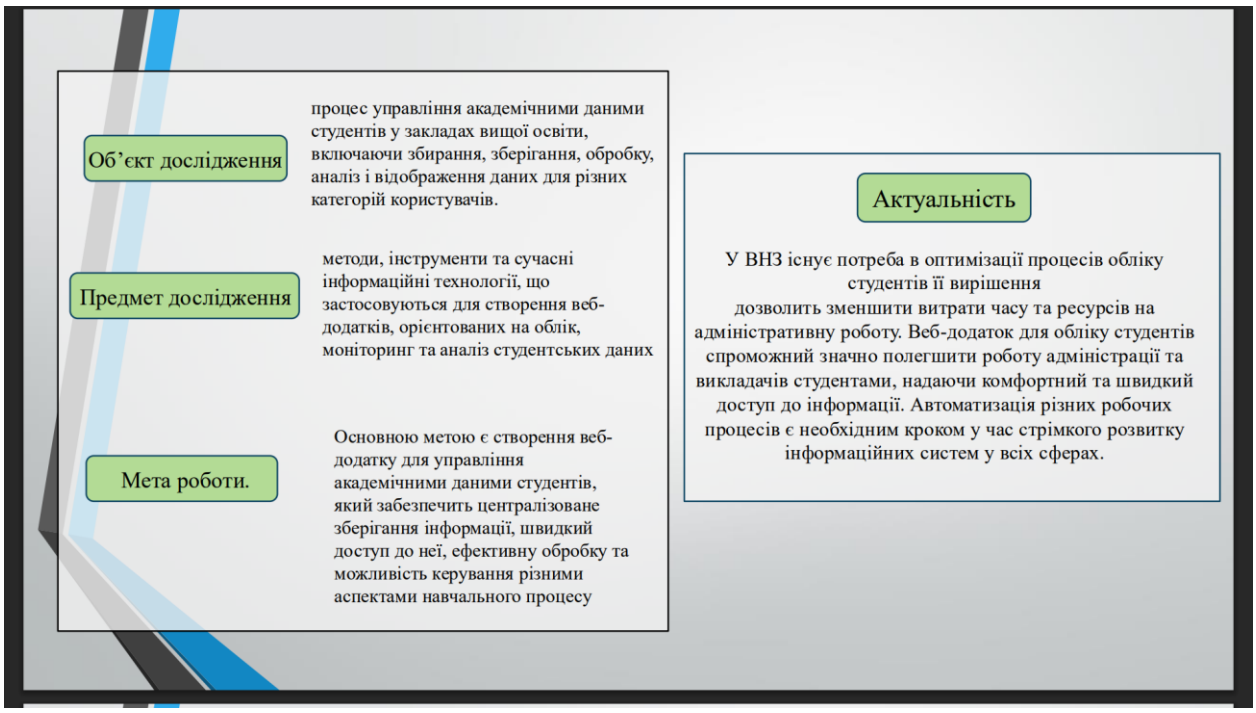
ПЕРЕЛІК ПОСИЛАНЬ

1. Гайна, Георгій Анатолійович, Основи проектування баз даних :Київ : Кондор, 2024.
2. Kameron Hussain, Frahaan Hussain, Mastering Bootstrap 5: From Basics to Expert Projects, Опубліковано незалежно 2023
3. Висоцька, Вікторія Анатоліївна, PYTHON : алгоритмізація та програмування Львів : Видавництво "Новий Світ-2000", 2023.
4. Jay A. Kreibich, Using SQLite: Small. Fast. Reliable. Choose Any Three, O'Reilly Media; 1st edition 2010
5. Берко А. Ю., Верес О. М., Пасічник В. В. Системи баз даних та знань. Книга 2. Системи управління базами даних та знань. Львів: Магнолія-2006, 2021. ISBN: 978-966-2025-56-9
6. Пасічник, Оксана Володимирівна, Веб-дизайн Львів : Видавництво "Магнолія 2006", 2021.
7. Цеслів О. В. Основи програмування та веб-дизайн: Навч. посіб. Київ, 2020. 149 с.
8. Підручник "Основи веб-розробки" Харківський національний університет радіоелектроніки, 2023.
9. Eric Matthes. Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to Programming 2nd Edition, No Starch Press; 2nd edition 2019
10. Flask Documentation. URL: https://flask.palletsprojects.com/
11. Python.org URL: https://docs.python.org/3/
12. SQLite Documentation URL: https://www.sqlite.org/docs.html
13. НАВЧАЛЬНИЙ ПОСІБНИК ПРОГРАМУВАННЯ МОВОЮ PYTHON (основи,інклюзивнийкурс).URL: https://duikt.edu.ua/uploads/l_2237_33503430.pdf
14. Проектування інформаційних систем. URL: https://duikt.edu.ua/uploads/l_144_42481385.pdf

15. Bootstrap 5 Documentation. URL: https://getbootstrap.com/
16. Bootstrap 5 Tutorial. URL: https://www.w3schools.com/bootstrap5/index.php
17. GitHub: Flask Examples. URL: https://github.com/pallets/flask
18. Flask Tutorial. URL: https://www.geeksforgeeks.org/flask-tutorial/
19. Powerschool Software. URL: https://www.powerschool.com/
20. 10 Illustrated examples of Visual Studio. URL: https://www.troyhunt.com/10-illustrated-examples-of-visual
21. Eskooly Application. URL: https://eskooly.com/bb/Desktop/

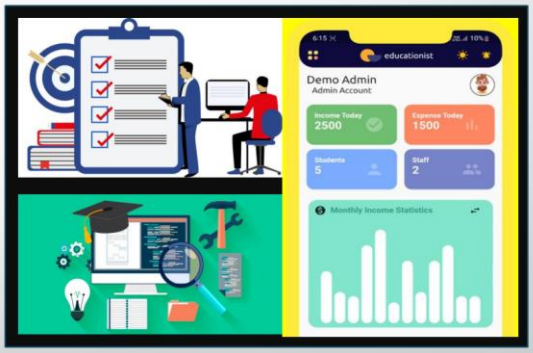
ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (ПРЕЗЕНТАЦІЯ)





Завдання дослідження

1. Проаналізувати існуючі системи обліку студентів, їх переваги та недоліки
2. Обґрунтувати вибір інструментів та технологій для розробки
3. Розробити архітектуру системи
4. Створити функціональний веб-додаток
5. Реалізувати авторизацію, управління студентами та групами
6. Забезпечити безпеку зберігання даних та гнучкість ролей



Аналіз існуючих рішень

1. PowerSchool SIS – складна система для великих ВНЗ
2. eSckooly – має десктопну і мобільну версію, але орієнтована на загальні задачі
3. Educationalist – проста у використанні, але має обмежений функціонал

Назва	Підключення до бази даних	Наявність мультіплатформе ності	Можливість залучення роботодавців	Взаємодія зі студентами/ у чнями	Реалізація пошуку студентів
PowerSchool SIS	+	+	-	+	+
eSckooly	+	-	+	+	+
Educationalist	+	-	-	+	+

Вибір технологій

Bootstrap 5 – популярний CSS-фреймворк для створення адаптивного та зрозумілого інтерфейсу. Завдяки використанню Bootstrap додаток коректно відображається як на комп'ютерах, так і на мобільних пристроях.

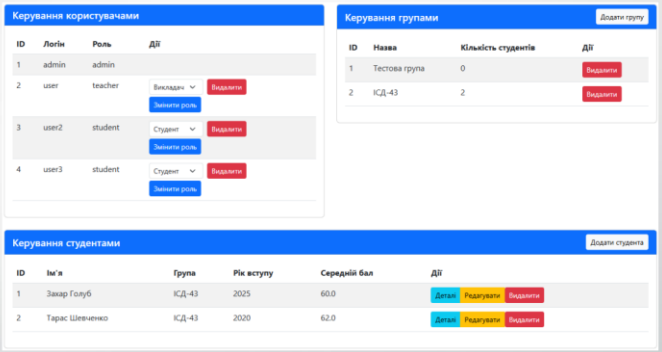
Flask – це мікрофреймворк на мові Python. Його головною перевагою є гнучкість. До того ж, екосистема Flask підтримує розширення для авторизації, роботи з базами даних та обробки форм.

SQLite – вбудована реляційна база даних, яка не потребує окремого сервера для роботи. Вона дуже зручна на етапі розробки й підходить для застосунків із невеликою кількістю користувачів.

Jinja2 – потужний шаблонізатор, вбудований у Flask, що дозволяє генерувати HTML-сторінки з динамічним наповненням.

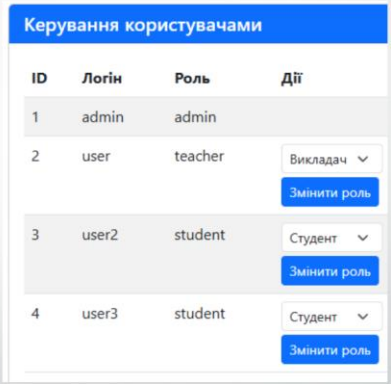
Функціональні можливості додатка

1. Автентифікація та розмежування ролей
2. Додавання та редагування студентів
3. Додавання груп
4. Пошук студентів
5. Детальний перегляд інформації про студента
6. Адаптивність інтерфейсу
7. Сповіщення та валідація



Інтерфейс: Авторизація та ролі

Система підтримує механізм реєстрації та входу користувачів із використанням захищеного зберігання паролів (хешування). Всі користувачі при вході отримують відповідні права доступу, залежно від призначеної ролі: адміністратор, викладач, або студент. Для кожної ролі визначено окремий інтерфейс, що мінімізує ризик випадкового доступу до несанкціонованої інформації.



Інтерфейс: Авторизація та ролі

Адміністратор

Має повний контроль над даними в системі. Основні можливості:

1. Керування користувачами:
2. Керування студентами:
3. Керування групами:
4. Моніторинг активності користувачів:
5. Пошук студентів за ПІБ.

Викладач

Має доступ до груп, і функціонал для взаємодії зі студентами:

1. Перегляд списку студентів своєї групи.
2. Редагування характеристик студентів (наприклад, опис академічних досягнень чи поведінки).
3. Внесення або оновлення середнього балу.
4. Пошук студентів за ПІБ.

Студент

В особистому кабінеті студента Кожен студент після входу може побачити власні дані в особистому кабінеті: Ім'я, прізвище, дата народження, група. Середній бал, характеристика, які можуть оновлюватися викладачем.

Проєктування бази даних

У розробленій моделі бази даних виділено кілька основних сутностей, які утворюють каркас всієї системи. Кожна з них відповідає певній реальній ролі або об'єкту у процесі функціонування закладу освіти.

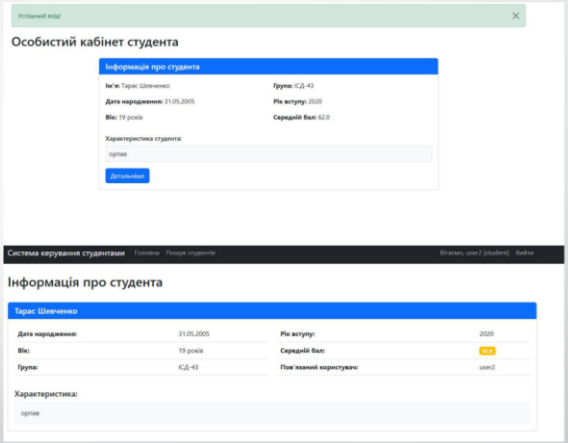
1. User (Користувач)
   Центральна таблиця, що зберігає облікові записи всіх користувачів — незалежно від того, чи є вони студентами, викладачами чи адміністраторами.
   Ця таблиця лежить в основі системи авторизації та дозволяє керувати правами доступу, визначаючи, які дії доступні конкретному користувачу.
2. Student (Студент)
   Містить детальну інформацію про студентів, у тому числі особисті й навчальні показники.
3. Group (Група)
   У таблиці groups зберігаються назви всіх академічних груп, до яких належать студенти.
4. Teacher (Викладач)
   Описує викладачів, які мають змогу взаємодіяти з групами студентів.
5. GroupAssignment (Призначення груп викладачам)
   Це службова таблиця, яка реалізує зв'язок багато-до-багатьох між викладачами та навчальними групами.
   Ця таблиця дозволяє призначити одного викладача до кількох груп, або кількох викладачів до однієї групи — наприклад, у випадку командного викладання чи змінного навантаження.

Проєктування бази даних

Схема зв'язків між таблицями
Загальна логіка структури бази даних демонструє такі типи зв'язків:
Один-до-одного — між users і students, users і teachers (через user_id);
Один-до-багатьох — між groups і students;
Багато-до-багатьох — між teachers і groups, реалізований через проміжну таблицю group_assignments.
Ця структура повністю задовольняє вимоги системи до зберігання та обробки даних, дозволяє швидко здійснювати вибірки, фільтрування, агрегацію та статистичний аналіз.
Діаграма структури бази даних

Результати виконаної роботи

Реалізовано повнофункціональний веб-додаток
У межах кваліфікаційної роботи створено веб-додаток, який охоплює всі основні функції системи обліку студентів:
– додавання, редагування, видалення та перегляд даних про студентів;
– створення та керування навчальними групами;
– авторизація користувачів з розмежуванням ролей (адміністратор, викладач, студент);
– функціонал пошуку, сповіщення про дії та повідомлення про помилки.
Завдяки цьому додаток може повноцінно використовуватись у навчальному процесі.

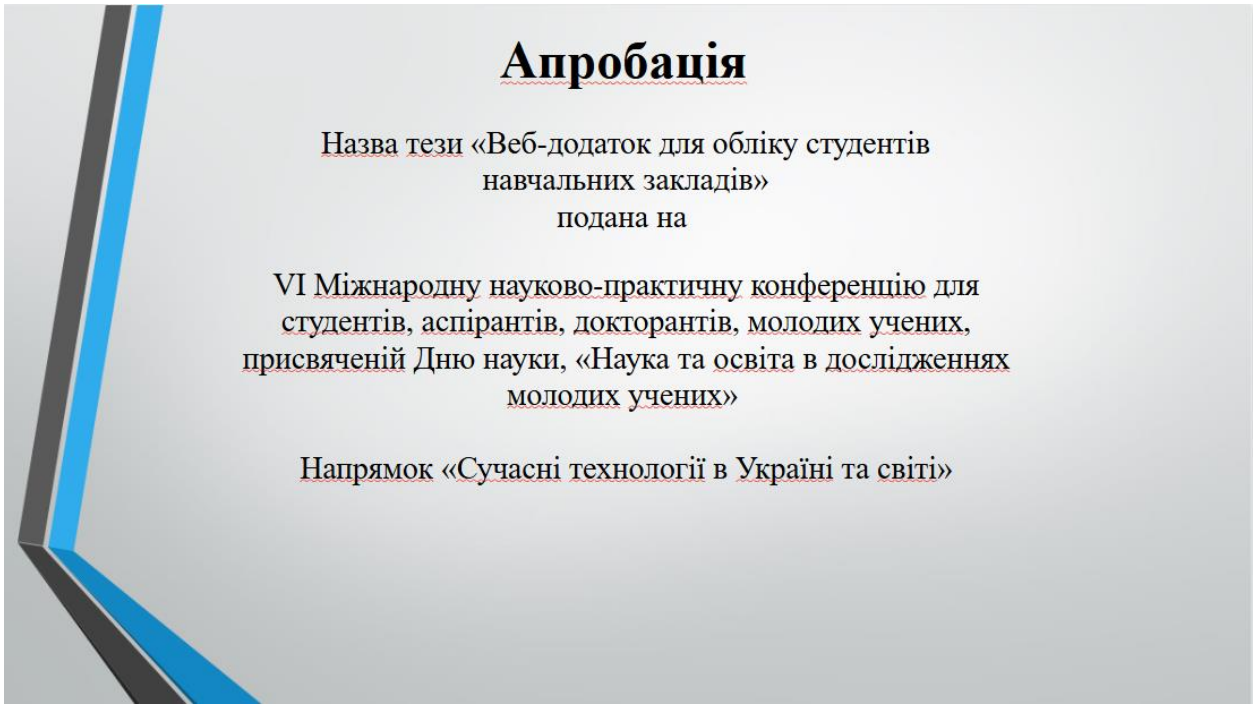


Висновки та перспективи розвитку

Система спрощує роботу адміністрації та викладачів
Завдяки автоматизації основних процесів обліку студентів зменшується навантаження на адміністративний персонал. Користувачі можуть швидко додавати, редагувати та переглядати інформацію, що економить час і зменшує кількість помилок.
Підвищується прозорість та ефективність обліку
Кожен студент має доступ до власної інформації, а викладачі – до даних своїх груп. Це дозволяє краще контролювати академічну успішність та оперативно реагувати на зміни.

Перспективи розвитку системи
Проєкт має потенціал до подальшого вдосконалення:

1. Електронний журнал – фіксація оцінок і відвідуваності;
2. Інтеграція з Moodle – синхронізація з платформами дистанційного навчання;
3. Хмарне зберігання – забезпечення доступу з будь-якого пристрою;
4. Аналітика та статистика – автоматичне формування звітів і прогнозів успішності.


Апробація
Назва тези «Веб-додаток для обліку студентів навчальних закладів»
подана на
VI Міжнародну науково-практичну конференцію для студентів, аспірантів, докторантів, молодих учених, присвяченій Дню науки, «Наука та освіта в дослідженнях молодих учених»
Напрямок «Сучасні технології в Україні та світі»