ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____Олена ОЛЬХОВСЬКА
(підпис)

«_____»___202_ р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему

«ПРОГРАМНА РЕАЛІЗАЦІЯ СОЦІАЛЬНОЇ МЕРЕЖІ ДЛЯ СТУДЕНТІВ З ВИКОРИСТАННЯМ МОВИ ПРОГРАМУВАННЯ PYTHON»

зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня бакалавра

Виконавець роботи Барабаш Руслан Анатолійович
_____«____» _____202_ р.
(підпис)

Науковий керівник доц., к.ф.-м.н. Черненко Оксана Олексіївна
_____«____»_____202_ р.
(підпис)

Рецензент

ПОЛТАВА 2024

РЕФЕРАТ

Записка: 86 с., 9 рис., 1 таблиця, 1 додаток, 15 джерел.

СОЦІАЛЬНА МЕРЕЖА, СТУДЕНТИ, PYTHON, FLASK

Об’єкт розробки – програмна реалізація соціальної мережі для студентів.

Мета роботи – розробка соціальної мережі для студентів, яка дозволяє користувачам реєструватися, створювати пости, коментувати, ставити лайки, редагувати профіль, обмінюватися повідомленнями та завантажувати медіафайли, використовуючи мову програмування Python та фреймворк Flask.

Методи дослідження – методи програмування на мові Python з використанням фреймворку Flask, бібліотеки SQLAlchemy для роботи з базою даних, а також методи тестування та налагодження веб-додатків.

Розроблено веб-додаток, який дозволяє користувачам реєструватися та авторизуватися, створювати та редагувати профілі з можливістю завантаження аватарів, публікувати пости з медіафайлами, коментувати та ставити лайки до постів, а також обмінюватися повідомленнями між користувачами. Крім того, додаток включає можливість пошуку користувачів за іменем та прізвищем.

Було проведено огляд існуючих соціальних мереж, аналіз їх функціоналу та інтерфейсів. На основі аналізу визначено ключові вимоги до функціональності та дизайну власного додатку. Відбулося проектування архітектури додатку, його реалізація, тестування та оптимізація.

Результати роботи включають розроблену програму з документацією, що містить інструкції для користувачів та рекомендації щодо подальшого розвитку додатку. Висвітлено позитивні та негативні сторони використання Flask у контексті розробки веб-додатків.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕННЬ, СИМВОЛІВ, СКОРОЧЕННЬ, ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	Пояснення умовних позначень, скорочень, символів
Flask	Фреймворк для розробки веб-додатків на Python
SQLite	Легка вбудована база даних, яка використовується для збереження даних додатку
HTML	Мова розмітки гіпертексту, використовується для створення веб-сторінок
CSS	Мова стилізації веб-документів
JavaScript	Мова програмування, яка використовується для додавання інтерактивності на веб-сторінках
SQLAlchemy	ORM-бібліотека для Python, яка полегшує роботу з базами даних
Jinja2	Шаблонізатор для Python, використовується для динамічного створення HTML
Bootstrap	Фреймворк для швидкої розробки адаптивних веб-дизайнів
API	Інтерфейс програмування додатків, дозволяє взаємодію між різними програмами
CRUD	Аббревіатура, що означає створення (Create), читання (Read), оновлення (Update), видалення (Delete) даних
CSRF	Захист від підробки міжсайтових запитів, метод безпеки для захисту від атак на веб-додатки
JWT	JSON Web Token, стандарт для створення токенів доступу, що дозволяють безпечний обмін інформацією

CLI	Командний інтерфейс користувача, інтерфейс для взаємодії з додатком через командний рядок
ORM	Об'єктно-реляційне відображення, технологія, яка дозволяє взаємодіяти з базою даних за допомогою об'єктів
MVC	Архітектурний шаблон Model-View-Controller, який розділяє додаток на три взаємодіючі компоненти: модель, уявлення та контролер
HTTP	Протокол передачі гіпертексту, основний протокол для передачі даних у веб
REST	Стиль архітектури для створення веб-сервісів, що використовує HTTP
SSL/TLS	Протоколи для забезпечення захищеної передачі даних через Інтернет
JSON	Формат обміну даними, який є зручним для читання людиною та машинної обробки
Blueprint	Функція Flask для організації додатку в модулі та створення масштабованої структури проекту
Alembic	Інструмент для міграцій баз даних у проєктах, що використовують SQLAlchemy
WSGI	Веб-серверний шлюзовий інтерфейс, стандарт для взаємодії веб-серверів з веб-додатками у Python

ВСТУП

У сучасному світі, де цифрові технології та Інтернет стали невід'ємною частиною нашого життя, соціальні мережі відіграють важливу роль у повсякденному спілкуванні та обміні інформацією. Соціальні мережі стали не тільки платформою для особистого спілкування, але й важливим інструментом для професійного розвитку, навчання та організації студентського життя.

Актуальним є розробка соціальних мереж, орієнтованих на специфічні групи користувачів, такі як студенти, які потребують платформи для взаємодії, обміну знаннями та співпраці над спільними проектами. Використання сучасних технологій, таких як Python та Flask, дозволяє створювати потужні та надійні рішення, які відповідають потребам студентів.

Об’єктом розробки є соціальна мережа для студентів, яка включає функціонал реєстрації та авторизації користувачів, створення постів, коментування, систему лайків, редагування профілю, обмін повідомленнями та завантаження медіафайлів.

Предметом розробки є система соціальної мережі, реалізована на базі фреймворку Flask з використанням мови програмування Python.

Метою роботи є розробка соціальної мережі для студентів, яка забезпечує зручне спілкування та взаємодію між користувачами, дозволяє обмінюватися інформацією, співпрацювати над навчальними проектами та підтримувати активне соціальне життя.

Методи дослідження включають аналіз існуючих соціальних мереж, їх методів та інструментів, розробку структури проекту, використання бібліотек та фреймворків, проектування та реалізацію основного функціоналу, тестування додатку на різних браузерах, а також аналіз його впливу на зручність користувачів.

Робота складається з вступу, де визначено актуальність, мету, об'єкт і предмет дослідження; інформаційного огляду, який присвячений теоретичним основам розробки веб-додатків, інструментам та технологіям для розробки на Flask, а також прикладам аналогічних соціальних мереж; теоретичної частини, що описує принципи розробки веб-додатків, особливості роботи з Flask та питання

забезпечення безпеки даних; практичної частини, яка включає опис процесу розробки програмного забезпечення, структури проекту, використаних бібліотек та фреймворків, а також функціоналу програми; висновків та додатків.

Розробка соціальної мережі для студентів на базі Flask дозволяє детально вивчити процеси організації та управління користувацькими даними, забезпечення безпеки даних, а також інтеграції комунікаційних інструментів для покращення взаємодії студентів. Це сприяє створенню ефективних інструментів для спілкування та співпраці, що підвищує загальну продуктивність та соціальну активність студентів.

1. ПОСТАНОВКА ЗАДАЧІ

Актуальність теми

Сучасні технології та Інтернет суттєво впливають на різні аспекти життя, включаючи освіту та соціальну взаємодію. Студенти, як активні користувачі цифрових технологій, потребують спеціалізованих платформ для спілкування, обміну знаннями та співпраці над навчальними проектами. В умовах дистанційного навчання та зростаючої популярності онлайн-освіти виникає необхідність у створенні соціальних мереж, орієнтованих саме на студентів. Такі мережі повинні забезпечувати зручний інтерфейс, функціональність та безпеку даних.

Соціальні мережі для студентів дозволяють не лише підтримувати зв'язок з однокурсниками, а й знаходити нових друзів, ділитися навчальними матеріалами, організовувати групи за інтересами та брати участь у спільних проектах. Вони сприяють формуванню активного студентського співтовариства, що підвищує загальну ефективність навчального процесу та соціальної активності студентів.

Мета роботи

Метою даної роботи є розробка соціальної мережі для студентів, яка забезпечує зручне спілкування та взаємодію між користувачами, дозволяє обмінюватися інформацією, співпрацювати над навчальними проектами та підтримувати активне соціальне життя, використовуючи мову програмування Python та фреймворк Flask.

Задачі роботи

Для досягнення поставленої мети необхідно вирішити такі задачі:

1. Провести аналіз існуючих соціальних мереж для студентів, визначити їхні переваги та недоліки.
2. Визначити вимоги до функціональності та дизайну власної соціальної мережі для студентів.
3. Розробити архітектуру додатку, вибрати відповідні інструменти та технології для його реалізації.
4. Реалізувати функціональність додатку, яка включає:
    - Реєстрацію та авторизацію користувачів.

- Створення та редагування профілів з можливістю завантаження аватарів.
- Публікацію постів з можливістю додавання медіафайлів.
- Систему коментування та лайків.
- Обмін повідомленнями між користувачами.
- Пошук користувачів за ім'ям та прізвищем.

5. Забезпечити безпеку даних користувачів та захист від несанкціонованого доступу.
6. Провести тестування та налагодження додатку, перевірити його роботу на різних браузерах.
7. Розробити документацію, яка містить інструкції для користувачів та рекомендації щодо подальшого розвитку додатку.
8. Оцінити ефективність розробленого додатку, визначити його позитивні та негативні сторони, а також можливості для подальшого вдосконалення.

Таким чином, виконання цих задач сприятиме створенню ефективного інструменту для спілкування та співпраці студентів, що підвищить їх продуктивність та соціальну активність.

2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1 Платформи для розробки веб-додатків

Сучасні веб-додатки відіграють ключову роль у цифровому світі, забезпечуючи користувачам доступ до різноманітних послуг та інформації через Інтернет. Розробка таких додатків вимагає використання спеціалізованих платформ, які забезпечують необхідні інструменти та середовище для створення, тестування та розгортання веб-додатків. Розглянемо основні платформи для розробки веб-додатків:

1. Django

Django — це високорівневий веб-фреймворк для Python, що сприяє швидкій розробці та чіткому дизайну веб-додатків. Django надає великий набір вбудованих інструментів, які полегшують розробку, включаючи систему аутентифікації, адміністраторську панель, ORM (Object-Relational Mapping) для роботи з базами даних, шаблони для HTML-сторінок та інше. Основні переваги Django:

- Висока швидкість розробки.
- Наявність вбудованих інструментів та бібліотек.
- Зосередженість на безпеці та масштабованості.
- Підтримка різних баз даних (PostgreSQL, MySQL, SQLite, Oracle).

2. Flask

Flask — це мікрофреймворк для Python, який надає мінімальний набір функцій, дозволяючи розробникам додавати необхідні компоненти та бібліотеки відповідно до вимог проекту. Flask забезпечує високу гнучкість та простоту, що робить його ідеальним для невеликих проектів або прототипів. Основні переваги Flask:

- Простота та легкість у використанні.
- Гнучкість у виборі компонентів та бібліотек.
- Легка інтеграція з іншими технологіями та сервісами.
- Відмінна документація та активна спільнота.

3. Ruby on Rails

Ruby on Rails (або просто Rails) — це веб-фреймворк, написаний на мові програмування Ruby. Rails слідує принципам "Convention over Configuration" (конвенції понад конфігурацію) та "Don't Repeat Yourself" (не повторюй себе), що дозволяє розробникам швидко створювати ефективні та масштабовані веб-додатки. Основні переваги Rails:

- Швидкість розробки завдяки конвенціям.
- Наявність вбудованих інструментів та бібліотек.
- Підтримка RESTful архітектури.
- Активна спільнота та великий набір ресурсів.

4. ASP.NET

ASP.NET — це фреймворк від Microsoft для створення динамічних веб-додатків, веб-сайтів та веб-сервісів. Використовуючи мову програмування C#, ASP.NET дозволяє створювати потужні та масштабовані рішення, інтегруючи їх з іншими продуктами Microsoft. Основні переваги ASP.NET:

- Підтримка різноманітних мов програмування (C#, VB.NET).
- Висока продуктивність та масштабованість.
- Інтеграція з іншими продуктами Microsoft (Azure, SQL Server).
- Система управління контентом (CMS) та інструменти для розробки.

5. Node.js

Node.js — це серверне середовище для виконання JavaScript, що дозволяє розробляти високопродуктивні та масштабовані веб-додатки. Node.js використовує подієво-орієнтовану модель, що робить його ідеальним для додатків з високою кількістю одночасних з'єднань, таких як чати та реальні часи оновлення даних. Основні переваги Node.js:

- Висока продуктивність завдяки подієво-орієнтованій моделі.
- Використання однієї мови (JavaScript) на клієнті та сервері.
- Великий екосистема пакетів (npm).
- Підтримка реальних часів оновлення даних.

6. Spring Boot

Spring Boot — це фреймворк для створення мікросервісів на основі мови програмування Java. Spring Boot спрощує налаштування та розгортання додатків, надаючи попередньо налаштовані шаблони та автоматичну конфігурацію. Основні переваги Spring Boot:

- Прискорене налаштування та розгортання додатків.
- Інтеграція з іншими проектами Spring (Spring Security, Spring Data).
- Підтримка мікросервісної архітектури.
- Широка спільнота та надійна документація.

Вибір платформи для розробки веб-додатків залежить від вимог проекту, масштабу, мов програмування, які використовуються, та інших факторів. Для розробки соціальної мережі для студентів на базі Python було обрано фреймворк Flask завдяки його простоті, гнучкості та можливості швидкого створення прототипів, що дозволяє зосередитися на специфічних вимогах користувачів та швидкій інтеграції додаткових функцій.

2.2 Огляд інструментів і технологій для Flask

Для створення веб-додатків на основі Python і Flask необхідно використовувати різні інструменти та технології, які забезпечують ефективну розробку, тестування та розгортання додатків. Розглянемо основні інструменти та технології, які застосовуються в цьому процесі.

1. Python

Python — це високорівнева мова програмування, яка відома своєю простотою та зручністю для розробників. Вона підтримує різні парадигми програмування, включаючи об'єктно-орієнтоване, функціональне та процедурне програмування. Основні переваги Python:

- Легкість вивчення та читабельність коду.
- Велика стандартна бібліотека.
- Широка підтримка спільноти.
- Великий набір сторонніх бібліотек та фреймворків.

2. Flask

Flask — це мікрофреймворк для веб-розробки на мові Python. Він надає мінімальний набір функцій, дозволяючи розробникам додавати необхідні компоненти та бібліотеки відповідно до вимог проекту. Основні переваги Flask:

- Простота та легкість у використанні.
- Гнучкість у виборі компонентів та бібліотек.
- Легка інтеграція з іншими технологіями та сервісами.
- Відмінна документація та активна спільнота.

3. Jinja2

Jinja2 — це шаблонізатор для Python, який використовується у Flask для динамічного створення HTML-сторінок. Jinja2 дозволяє вбудовувати Python-код у HTML, що забезпечує гнучкість у створенні шаблонів. Основні переваги Jinja2:

- Підтримка умовних операторів та циклів.
- Можливість розширення шаблонів.
- Висока продуктивність.
- Легкість інтеграції з Flask.

4. SQLAlchemy

SQLAlchemy — це ORM (Object-Relational Mapping) бібліотека для Python, яка дозволяє взаємодіяти з базами даних, використовуючи об'єктно-орієнтований підхід. Основні переваги SQLAlchemy:

- Підтримка різних баз даних (PostgreSQL, MySQL, SQLite, Oracle).
- Абстракція роботи з базами даних.
- Потужні можливості для побудови запитів.
- Легкість інтеграції з Flask.

5. Flask-Migrate

Flask-Migrate — це розширення для Flask, яке забезпечує міграцію баз даних за допомогою бібліотеки Alembic. Це дозволяє легко змінювати структуру бази даних під час розробки додатку. Основні переваги Flask-Migrate:

- Простота у використанні.
- Інтеграція з SQLAlchemy.

- Можливість автоматичного створення та застосування міграцій.
- Відмінна документація.

6. Flask-WTF

Flask-WTF — це розширення для Flask, яке забезпечує інтеграцію з бібліотекою WTForms для створення та валідації форм. Основні переваги Flask-WTF:

- Підтримка CSRF-захисту.
- Можливість створення складних форм з валідацією.
- Легкість інтеграції з шаблонами Jinja2.
- Відмінна документація.

7. Flask-Login

Flask-Login — це розширення для Flask, яке забезпечує управління сесіями користувачів та аутентифікацію. Основні переваги Flask-Login:

- Простота у використанні.
- Підтримка "запам'ятати мене" функціональності.
- Легкість інтеграції з іншими розширеннями Flask.
- Відмінна документація.

8. Flask-Mail

Flask-Mail — це розширення для Flask, яке забезпечує надсилання електронних листів через додаток. Основні переваги Flask-Mail:

- Простота налаштування.
- Підтримка різних поштових серверів.
- Легкість інтеграції з додатком Flask.
- Відмінна документація.

9. Bootstrap

Bootstrap — це популярний фреймворк для розробки адаптивних та сучасних веб-інтерфейсів. Він включає готові компоненти для створення форм, кнопок, навігаційних панелей та інших елементів інтерфейсу. Основні переваги Bootstrap:

- Адаптивний дизайн.
- Величезна кількість готових компонентів.

- Легкість інтеграції з HTML та CSS.
- Активна спільнота та велика кількість ресурсів.

10. Git та GitHub

Git — це система контролю версій, яка дозволяє відстежувати зміни у коді та працювати з різними гілками розробки. GitHub — це веб-сервіс для хостингу репозиторіїв Git, який надає інструменти для спільної розробки, відстеження проблем та управління проектами. Основні переваги Git та GitHub:

- Можливість командної роботи.
- Відстеження змін та управління версіями.
- Інтеграція з іншими інструментами для розробки.
- Велика спільнота розробників.

Використання цих інструментів та технологій дозволяє створювати потужні, гнучкі та масштабовані веб-додатки на базі Python та Flask. Вони забезпечують зручність розробки, тестування та розгортання додатків, що відповідають сучасним вимогам до функціональності, безпеки та продуктивності.

2.3 Аналіз аналогічних соціальних мереж

Аналіз існуючих соціальних мереж дозволяє виявити їхні ключові характеристики, переваги та недоліки, а також визначити основні вимоги та напрямки для розробки власної соціальної мережі для студентів. Розглянемо деякі популярні соціальні мережі та їхні особливості.

1. Facebook

Facebook є однією з найпопулярніших соціальних мереж у світі, з мільярдами користувачів. Основні функції Facebook включають створення профілів, публікацію постів, коментарів, фотографій і відео, систему лайків, обмін повідомленнями та створення груп за інтересами. Основні переваги Facebook:

- Величезна кількість користувачів та активна спільнота.
- Різноманітні функціональні можливості.
- Інтеграція з іншими сервісами та додатками.

- Потужні інструменти для аналітики та реклами.

Недоліки:

- Проблеми з конфіденційністю та безпекою даних.
- Перенасиченість функціями, що може ускладнити користування.
- Надмірна кількість реклами.

2. LinkedIn

LinkedIn — це соціальна мережа для професіоналів, яка орієнтована на створення професійних контактів, пошук роботи та обмін професійною інформацією. Основні функції LinkedIn включають створення професійних профілів, публікацію статей, участь у групах за інтересами, систему рекомендацій та обмін повідомленнями. Основні переваги LinkedIn:

- Професійна орієнтація.
- Можливості для пошуку роботи та рекрутингу.
- Інструменти для професійного нетворкінгу.
- Підтримка створення та обміну професійним контентом.

Недоліки:

- Обмежені можливості для неформального спілкування.
- Деякі функції доступні тільки в платній версії.
- Менша аудиторія порівняно з іншими соціальними мережами.

3. Twitter

Twitter — це соціальна мережа для мікроблогінгу, яка дозволяє користувачам публікувати короткі повідомлення (твіти) довжиною до 280 символів. Основні функції Twitter включають публікацію та ретвіти повідомлень, використання хештегів, обмін повідомленнями та підписку на інші акаунти. Основні переваги Twitter:

- Швидке поширення інформації.
- Можливість стежити за новинами та актуальними подіями.
- Простота та зручність використання.
- Можливість взаємодії з відомими особистостями та компаніями.

Недоліки:

- Обмежена довжина повідомлень.
- Високий рівень шуму та спаму.
- Проблеми з модерацією контенту.

4. Instagram

Instagram — це соціальна мережа для обміну фотографіями та відео, яка набрала популярності серед молоді. Основні функції Instagram включають публікацію фотографій та відео, використання фільтрів, систему лайків та коментарів, історії (Stories), обмін повідомленнями та можливість ведення прямих ефірів. Основні переваги Instagram:

- Орієнтація на візуальний контент.
- Зручний та інтуїтивно зрозумілий інтерфейс.
- Можливість взаємодії з великою кількістю користувачів.
- Інтеграція з іншими соціальними мережами.

Недоліки:

- Обмежені можливості для текстового контенту.
- Проблеми з конфіденційністю та безпекою даних.
- Високий рівень конкуренції за увагу користувачів.

5. VKontakte (VK)

VKontakte — це популярна соціальна мережа в країнах СНД, яка надає широкий спектр функцій для спілкування та обміну контентом. Основні функції VK включають створення профілів, публікацію постів, коментарів, фотографій та відео, систему лайків, обмін повідомленнями, створення груп та спільнот, а також музичний і відеосервіс. Основні переваги VK:

- Велика кількість функцій для спілкування та обміну контентом.
- Інтеграція з різними сервісами.
- Велика аудиторія користувачів.
- Можливість створення тематичних спільнот та груп.

Недоліки:

- Проблеми з конфіденційністю та безпекою даних.
- Перенасиченість функціями, що може ускладнити користування.

- Реклама та спам.

Аналіз аналогічних соціальних мереж показує, що для успішної розробки соціальної мережі для студентів необхідно врахувати такі ключові аспекти:

- Забезпечення зручного та інтуїтивно зрозумілого інтерфейсу.
- Реалізація основних функцій для спілкування та обміну контентом (пости, коментарі, лайки, повідомлення).
- Забезпечення конфіденційності та безпеки даних користувачів.
- Інтеграція з іншими сервісами та можливість розширення функціональності.

На основі цього аналізу було визначено основні вимоги до нашої соціальної мережі для студентів, яка повинна забезпечити зручність використання, безпеку даних та підтримку активної взаємодії між користувачами.

2.4 Вибір бази даних та методів зберігання даних

У процесі розробки соціальної мережі для студентів важливим аспектом є вибір бази даних та методів зберігання даних. Вибір бази даних впливає на продуктивність, масштабованість та надійність додатку. Для нашого проекту було обрано використання ORM (Object-Relational Mapping) бібліотеки SQLAlchemy для взаємодії з реляційною базою даних. Розглянемо детальніше вибір бази даних та методів зберігання даних.

1. Вибір бази даних

Основними критеріями для вибору бази даних для нашої соціальної мережі є продуктивність, масштабованість, підтримка транзакцій, надійність та легкість інтеграції з обраними технологіями. Розглянемо деякі популярні реляційні бази даних:

a. PostgreSQL

PostgreSQL — це потужна, відкрита реляційна база даних з розширеними можливостями. Вона забезпечує високий рівень надійності, підтримує складні запити та транзакції, а також має вбудовані механізми для забезпечення цілісності даних. Основні переваги PostgreSQL:

- Підтримка стандарту SQL.
- Розширені можливості для зберігання та обробки даних.
- Висока продуктивність та масштабованість.
- Активна спільнота та регулярні оновлення.

b. MySQL

MySQL — це популярна реляційна база даних з відкритим вихідним кодом, відома своєю швидкістю та надійністю. Вона широко використовується для веб-додатків та має велику кількість інструментів для адміністрування та управління даними. Основні переваги MySQL:

- Висока продуктивність та швидкодія.
- Широка підтримка та інтеграція з багатьма мовами програмування.
- Проста в налаштуванні та використанні.
- Велика кількість документованих прикладів та ресурсів.

c. SQLite

SQLite — це легка реляційна база даних, яка не потребує налаштування серверного програмного забезпечення. Вона ідеально підходить для невеликих проектів або прототипів. Основні переваги SQLite:

- Простота використання та налаштування.
- Вбудована в багато мов програмування.
- Підтримка основних можливостей SQL.
- Низькі вимоги до ресурсів.

Для нашого проекту було обрано PostgreSQL як базу даних, оскільки вона забезпечує високу продуктивність, масштабованість та надійність, що є критичними для соціальної мережі.

2. Методи зберігання даних з використанням SQLAlchemy

SQLAlchemy — це потужна ORM бібліотека для Python, яка забезпечує гнучкість та зручність у роботі з реляційними базами даних. Використання SQLAlchemy дозволяє абстрагуватися від деталей взаємодії з базою даних та працювати з об'єктами Python замість написання SQL-запитів. Основні можливості SQLAlchemy:

- Підтримка різних реляційних баз даних (PostgreSQL, MySQL, SQLite, Oracle).
- Автоматичне відображення об'єктів Python на таблиці бази даних.
- Підтримка складних запитів, фільтрів та агрегацій.
- Механізми для управління транзакціями та зв'язками між таблицями.

3. Основні компоненти SQLAlchemy

a. Declarative Base

Declarative Base — це базовий клас, який використовується для визначення моделей (класів), що відображаються на таблиці бази даних. Кожна модель описує структуру таблиці, включаючи стовпці та їх типи даних.

```
from sqlalchemy.ext.declarative import declarative_base
Base = declarative_base()

class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True)
    username = Column(String, unique=True, nullable=False)
    email = Column(String, unique=True, nullable=False)
    password_hash = Column(String, nullable=False)
```

b. Сесії (Sessions)

Сесії у SQLAlchemy використовуються для взаємодії з базою даних. Вони дозволяють виконувати запити, додавати, оновлювати та видаляти дані.

```
from sqlalchemy.orm import sessionmaker
Session = sessionmaker(bind=engine)
session = Session()

# Додавання нового користувача
new_user = User(username='student', email='student@example.com', password_hash='hashed_password')
session.add(new_user)
session.commit()
```

c. Зв'язки між таблицями (Relationships)

SQLAlchemy підтримує визначення зв'язків між таблицями, таких як один-до-багатьох, багато-до-одного та багато-до-багатьох.

```
class Post(Base):
    __tablename__ = 'posts'
    id = Column(Integer, primary_key=True)
    title = Column(String, nullable=False)
    content = Column(Text, nullable=False)
    user_id = Column(Integer, ForeignKey('users.id'))
    user = relationship('User', back_populates='posts')


User.posts = relationship('Post', order_by=Post.id, back_populates='user')
```

4. Використання Flask-Migrate для управління міграціями

Flask-Migrate забезпечує управління міграціями бази даних, дозволяючи легко додавати та змінювати структуру таблиць без втрати даних. Це особливо корисно в процесі розробки, коли зміни у структурі бази даних є частими.

```
from flask_migrate import Migrate
migrate = Migrate(app, db)
```

Вибір PostgreSQL як бази даних та використання SQLAlchemy для взаємодії з нею забезпечують надійну та масштабовану основу для розробки соціальної мережі для студентів. Використання Flask-Migrate дозволяє легко управляти змінами у структурі бази даних, що спрощує процес розробки та підтримки додатку.

3. ТЕОРЕТИЧНА ЧАСТИНА

3.1 Принципи розробки веб-додатків

Розробка веб-додатків є складним процесом, який вимагає врахування багатьох аспектів, включаючи архітектуру, безпеку, продуктивність та зручність використання. Дотримання основних принципів розробки допомагає створювати ефективні, надійні та масштабовані додатки. Розглянемо ключові принципи розробки веб-додатків.

1. Архітектура веб-додатків

Архітектура веб-додатків визначає структуру системи та взаємодію між її компонентами. Існує кілька популярних архітектурних підходів:

a. Клієнт-серверна архітектура

У цьому підході клієнтська частина (front-end) та серверна частина (back-end) розділені. Клієнт відправляє запити до сервера, а сервер обробляє ці запити та повертає відповіді. Основні переваги:

- Розподіл навантаження між клієнтом та сервером.
- Можливість використання різних технологій для клієнта та сервера.
- Підтримка масштабованості.

b. Трьохрівнева архітектура

Цей підхід включає три рівні: клієнтський, серверний та рівень бази даних. Сервер обробляє бізнес-логіку та взаємодіє з базою даних. Основні переваги:

- Розподіл відповідальності між рівнями.
- Підвищена безпека та масштабованість.
- Можливість окремого оновлення рівнів.

c. Мікросервісна архітектура

Цей підхід передбачає розбиття додатку на окремі сервіси, кожен з яких відповідає за певну функціональність. Основні переваги:

- Легкість масштабування окремих сервісів.
- Підвищена стійкість до відмов.
- Можливість використання різних мов програмування та технологій для

кожного сервісу.

2. Безпека веб-додатків

Забезпечення безпеки є одним з найважливіших аспектів розробки веб-додатків. Основні принципи безпеки включають:

a. Аутентифікація та авторизація

Аутентифікація підтверджує особу користувача, а авторизація визначає, які ресурси та дії доступні для цього користувача. Для забезпечення безпеки використовуються такі технології, як JWT (JSON Web Tokens), OAuth, SAML та інші.

b. Захист від атак

Веб-додатки повинні бути захищені від різних видів атак, таких як XSS (Cross-Site Scripting), CSRF (Cross-Site Request Forgery), SQL-ін'єкції та інші. Для цього застосовуються відповідні методи захисту, такі як валідація та санітизація введених даних, використання захисних заголовків та інші.

c. Шифрування даних

Дані, які передаються між клієнтом та сервером, повинні бути захищені за допомогою шифрування (HTTPS). Також важливо шифрувати чутливі дані, що зберігаються в базі даних.

3. Продуктивність та масштабованість

Продуктивність та масштабованість є критичними для веб-додатків з великою кількістю користувачів. Основні підходи для забезпечення продуктивності:

a. Кешування

Кешування дозволяє зменшити навантаження на сервер та базу даних шляхом збереження часто використовуваних даних у швидкодоступному сховищі (кеші). Кешування може здійснюватися на різних рівнях, включаючи клієнтський, серверний та рівень бази даних.

b. Оптимізація запитів

Оптимізація SQL-запитів, використання індексів та інших методів дозволяє підвищити продуктивність взаємодії з базою даних. Важливо мінімізувати кількість запитів та забезпечити їх ефективне виконання.

c. Балансування навантаження

Для забезпечення масштабованості використовується балансування навантаження між кількома серверами. Це дозволяє рівномірно розподілити запити користувачів та підвищити стійкість до відмов.

4. Зручність використання (юзабіліті)

Зручність використання веб-додатку є важливим фактором для залучення та утримання користувачів. Основні принципи зручності використання:

a. Інтуїтивно зрозумілий інтерфейс

Інтерфейс повинен бути простим та зрозумілим для користувачів. Важливо забезпечити логічну структуру додатку, зрозумілі навігаційні елементи та доступ до основних функцій.

b. Швидкість завантаження сторінок

Швидкість завантаження сторінок впливає на задоволеність користувачів. Важливо оптимізувати ресурси (зображення, скрипти, стилі) для швидкого завантаження та використання асинхронного завантаження контенту.

c. Адаптивний дизайн

Адаптивний дизайн забезпечує коректне відображення веб-додатку на різних пристроях (десктопах, планшетах, смартфонах). Це дозволяє користувачам зручно працювати з додатком незалежно від пристрою.

5. Тестування та налагодження

Тестування є невід'ємною частиною процесу розробки веб-додатків. Основні види тестування:

a. Юніт-тестування

Юніт-тестування дозволяє перевірити окремі компоненти додатку на коректність їх роботи. Для цього використовуються фреймворки для тестування, такі як pytest для Python.

b. Інтеграційне тестування

Інтеграційне тестування перевіряє взаємодію між різними компонентами додатку, що дозволяє виявити проблеми в інтеграції.

c. Тестування користувацького інтерфейсу

Тестування користувацького інтерфейсу (UI-тестування) дозволяє перевірити, як користувачі взаємодіють з додатком. Для цього використовуються інструменти для автоматизації тестування, такі як Selenium.

d. Нагрузочне тестування

Нагрузочне тестування дозволяє перевірити, як додаток поводиться під великим навантаженням, та виявити можливі проблеми з продуктивністю.

Дотримання основних принципів розробки веб-додатків дозволяє створювати ефективні, надійні та зручні для користувачів додатки. Важливо враховувати архітектуру, безпеку, продуктивність, масштабованість та зручність використання на всіх етапах розробки. Тестування та налагодження є ключовими етапами, які забезпечують якість кінцевого продукту.

3.2 Особливості Flask

Flask — це мікрофреймворк для веб-розробки на мові програмування Python, який відзначається своєю простотою, гнучкістю та легкістю вивчення. Він ідеально підходить для створення як невеликих додатків, так і масштабних веб-систем. У цьому розділі розглянемо основні особливості та можливості Flask.

1. Легкість використання

Flask пропонує мінімальний набір необхідних інструментів, що дозволяє швидко розпочати розробку веб-додатків. Його структура дуже проста, що знижує поріг входження для нових розробників. Це забезпечує швидкий цикл розробки та зменшує складність проекту.

2. Гнучкість та модульність

Flask не нав'язує розробникам жорстких правил і дозволяє використовувати будь-які компоненти та бібліотеки, необхідні для проекту. Це дозволяє створювати гнучкі рішення, адаптовані до конкретних потреб. Завдяки великій кількості розширень можна легко додати функціональність, яка відсутня в базовій комплектації.

3. Простота маршрутизації

Маршрутизація у Flask реалізується за допомогою декораторів, що робить процес визначення URL-запитів інтуїтивно зрозумілим. Це дозволяє легко керувати маршрутами та забезпечує чітку структуру додатку.

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return "Hello, World!"

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

4. Шаблонізація з використанням Jinja2

Flask використовує Jinja2 як шаблонізатор, що дозволяє створювати динамічні HTML-сторінки. Jinja2 підтримує використання змінних, умовних операторів та циклів у шаблонах, що забезпечує гнучкість та динамічність інтерфейсів.

```
<!doctype html>
<html>
  <head>
    <title>{{ title }}</title>
  </head>
  <body>
    <h1>{{ heading }}</h1>
    <p>{{ content }}</p>
  </body>
</html>
```

5. Підтримка ORM через SQLAlchemy

Flask підтримує інтеграцію з різними ORM для роботи з базами даних. Найпопулярніше розширення — SQLAlchemy, яке забезпечує зручний спосіб

взаємодії з базами даних та управління даними.

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///site.db'
db = SQLAlchemy(app)

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(150), nullable=False, unique=True)
    email = db.Column(db.String(120), unique=True, nullable=False)

    def __repr__(self):
        return f"User('{self.username}', '{self.email}')"
```

6. Підтримка розширень

Flask має велику кількість розширень, які дозволяють додавати додаткові функції до додатку, такі як Flask-WTF для роботи з формами, Flask-Login для аутентифікації користувачів, Flask-Mail для надсилання електронних листів та Flask-Migrate для управління міграціями баз даних.

7. Обробка помилок

Flask забезпечує зручний механізм обробки помилок, що дозволяє налаштовувати спеціальні сторінки для різних типів помилок та винятків. Це допомагає створювати більш дружні до користувача додатки з інформативними повідомленнями про помилки.

```
@app.errorhandler(404)
def not_found_error(error):
    return render_template('404.html'), 404

@app.errorhandler(500)
```

```
def internal_error(error):
    db.session.rollback()
    return render_template('500.html'), 500
```

8. Вбудований веб-сервер та режим налагодження

Flask постачається з вбудованим веб-сервером, що ідеально підходить для розробки та тестування. Він також має режим налагодження, який дозволяє автоматично перезавантажувати додаток при внесенні змін у код та надає детальну інформацію про помилки.

9. Аутентифікація та авторизація

Flask-Login — це розширення, яке дозволяє легко реалізувати аутентифікацію та авторизацію користувачів. Воно підтримує функціонал "запам'ятати мене", обмеження доступу до певних маршрутів та багато іншого.

10. Документація та підтримка спільноти

Flask має добре задокументовану офіційну документацію та велику підтримку спільноти. Це забезпечує легкий доступ до ресурсів та прикладів, що допомагають у вирішенні проблем та реалізації різних функціональних можливостей.

Flask є потужним та гнучким інструментом для розробки веб-додатків, що поєднує простоту використання з багатими можливостями розширення. Він дозволяє розробникам швидко створювати як невеликі додатки, так і складні веб-системи, надаючи всі необхідні інструменти для досягнення цієї мети. Використання Flask забезпечує швидкий цикл розробки, високу продуктивність та можливість адаптації до специфічних вимог проекту.

3.3 Безпека даних у веб-додатках

Забезпечення безпеки даних у веб-додатках є одним з найважливіших аспектів розробки. Невідповідність стандартам безпеки може призвести до втрати конфіденційної інформації, фінансових збитків та пошкодження репутації. Веб-додатки є ціллю для різноманітних атак, таких як XSS (Cross-Site Scripting), CSRF (Cross-Site Request Forgery), SQL-ін'єкції та інші. У цьому розділі розглянемо

основні підходи та практики забезпечення безпеки даних у веб-додатках.

1. Аутентифікація та авторизація

Аутентифікація та авторизація є ключовими аспектами забезпечення безпеки веб-додатків:

a. Аутентифікація

Аутентифікація підтверджує особу користувача. Використання надійних механізмів аутентифікації, таких як двофакторна аутентифікація (2FA), допомагає захистити облікові записи користувачів від несанкціонованого доступу. Популярні методи аутентифікації включають:

- **JWT (JSON Web Tokens)**: використовується для передачі інформації про користувача між клієнтом та сервером.
- **OAuth**: протокол для аутентифікації та авторизації, що дозволяє використовувати облікові записи зовнішніх провайдерів (наприклад, Google, Facebook).
- **SAML (Security Assertion Markup Language)**: протокол для аутентифікації, що дозволяє обмінюватися інформацією про користувача між різними системами.

b. Авторизація

Авторизація визначає, які дії та ресурси доступні для аутентифікованого користувача. Це забезпечує контроль доступу та запобігає несанкціонованому використанню ресурсів додатку. Розмежування прав доступу здійснюється за допомогою ролей та дозволів.

2. Захист від атак

Веб-додатки повинні бути захищені від різних видів атак:

a. SQL-ін'єкції

SQL-ін'єкції дозволяють зловмисникам виконувати довільні SQL-запити до бази даних. Для захисту від SQL-ін'єкцій слід використовувати підготовлені вирази та ORM (Object-Relational Mapping), такі як SQLAlchemy.

 # Приклад захисту від SQL-ін'єкції за допомогою підготовлених виразів

*cursor.execute("SELECT * FROM users WHERE username = %s", (username,))*

b. XSS (Cross-Site Scripting)

XSS дозволяє зловмисникам впроваджувати шкідливий код у веб-сторінки, які відвідують інші користувачі. Для захисту від XSS необхідно виконувати санітизацію та екранування введених даних.

```
from flask import escape

@app.route('/greet')
def greet():
    name = request.args.get('name', '')
    return f'Hello, {escape(name)}!'
```

c. CSRF (Cross-Site Request Forgery)

CSRF дозволяє зловмисникам виконувати дії від імені аутентифікованого користувача. Для захисту від CSRF використовуються токени, які передаються разом з формами або AJAX-запитами.

```
from flask_wtf.csrf import CSRFProtect

csrf = CSRFProtect(app)
```

3. Шифрування даних

Шифрування даних допомагає захистити інформацію від несанкціонованого доступу під час передачі та зберігання:

a. Шифрування під час передачі

Використання HTTPS забезпечує шифрування даних, які передаються між клієнтом та сервером, за допомогою SSL/TLS протоколів. Це запобігає перехопленню та модифікації даних під час їх передачі.

b. Шифрування при зберіганні

Шифрування чутливих даних, таких як паролі користувачів, зберігає їх у безпечному вигляді. Для цього використовуються алгоритми хешування, такі як bcrypt або Argon2.

```
from werkzeug.security import generate_password_hash, check_password_hash
```

hashed_password = generate_password_hash('password123', method='bcrypt')

4. Управління сесіями

Сесії дозволяють зберігати стан користувача між запитами. Для забезпечення безпеки сесій необхідно використовувати надійні механізми управління сесіями та захищати їх від викрадення:

a. Використання захищених сесій

Фреймворки, такі як Flask, забезпечують механізми управління сесіями, що дозволяють зберігати дані користувачів у захищеному вигляді.

b. Налаштування cookies

Cookies, що використовуються для зберігання сесій, повинні бути налаштовані з параметрами HttpOnly та Secure, що запобігає доступу до них з JavaScript та передачі через незахищені з'єднання.

```
app.config.update(
    SESSION_COOKIE_HTTPONLY=True,
    SESSION_COOKIE_SECURE=True,
)
```

5. Логи та моніторинг

Моніторинг та ведення логів допомагають виявляти та реагувати на інциденти безпеки:

a. Логування

Ведення логів про події у додатку (вхід користувачів, зміни даних, помилки) дозволяє відстежувати підозрілі активності та аналізувати інциденти.

b. Моніторинг

Моніторинг додатку в реальному часі допомагає швидко виявляти атаки та реагувати на них. Використання спеціалізованих інструментів для моніторингу дозволяє забезпечити високий рівень безпеки.

Забезпечення безпеки даних у веб-додатках є критично важливим для захисту користувачів та їхньої інформації. Дотримання найкращих практик, таких як аутентифікація та авторизація, захист від атак, шифрування даних, управління сесіями, логування та моніторинг, дозволяє створювати надійні та захищені веб-

додатки.

3.4 Архітектура та дизайн веб-додатків

Архітектура та дизайн веб-додатків визначають структуру системи, взаємодію між її компонентами та користувацький досвід. Вибір правильної архітектури та дизайну впливає на продуктивність, масштабованість, зручність використання та підтримку додатка. Розглянемо основні принципи та підходи до архітектури та дизайну веб-додатків.

1. Архітектурні підходи

Існує кілька популярних архітектурних підходів, які можуть бути використані для створення веб-додатків:

a. Монолітна архітектура

Монолітна архітектура передбачає, що всі компоненти додатку (клієнтська частина, серверна логіка, база даних) інтегровані в один великий блок. Основні переваги монолітної архітектури:

- Простота розробки та розгортання.
- Легкість налагодження та тестування.
- Менші вимоги до інфраструктури.

Недоліки:

- Складність масштабування.
- Проблеми з підтримкою та оновленням великих проектів.
- Вразливість до змін: зміни в одній частині системи можуть вплинути на інші частини.

b. Мікросервісна архітектура

Мікросервісна архітектура розбиває додаток на окремі сервіси, кожен з яких відповідає за конкретну функціональність. Кожен сервіс працює незалежно та може бути розгорнутий окремо. Основні переваги мікросервісної архітектури:

- Легкість масштабування окремих сервісів.
- Можливість незалежного розгортання та оновлення.

- Підвищена стійкість до відмов: проблеми в одному сервісі не впливають на роботу інших.

Недоліки:

- Ускладнення розробки та підтримки.
- Вимоги до оркестрації сервісів та управління ними.
- Потреба у забезпеченні взаємодії між сервісами.

c. Serverless архітектура

Serverless архітектура передбачає виконання функцій без управління серверами. Функції виконуються на хмарних платформах, таких як AWS Lambda або Google Cloud Functions. Основні переваги serverless архітектури:

- Відсутність потреби в управлінні серверами.
- Плата тільки за виконання функцій.
- Легкість масштабування.

Недоліки:

- Обмеження по часу виконання функцій.
- Залежність від хмарного провайдера.
- Складність у налаштуванні та відлагодженні.

2. Дизайн користувацького інтерфейсу (UI)

Дизайн користувацького інтерфейсу визначає, як користувачі взаємодіють з веб-додатком. Важливо забезпечити інтуїтивність, зручність та естетичну привабливість інтерфейсу. Основні принципи дизайну UI:

a. Простота та зрозумілість

Інтерфейс повинен бути простим та зрозумілим для користувачів. Важливо уникати перевантаження інтерфейсу зайвою інформацією та елементами.

b. Адаптивний дизайн

Адаптивний дизайн забезпечує коректне відображення додатка на різних пристроях (десктопах, планшетах, смартфонах). Це досягається за допомогою медіазапитів у CSS та гнучких макетів.

c. Узгодженість

Узгодженість інтерфейсу забезпечує однаковий вигляд та поведінку елементів

на всіх сторінках додатка. Це включає використання єдиних кольорів, шрифтів, кнопок та інших компонентів.

d. Візуальна ієрархія

Візуальна ієрархія допомагає користувачам зрозуміти важливість та взаємозв'язок між елементами інтерфейсу. Це досягається за допомогою розміру, кольору, контрасту та розташування елементів.

3. Підходи до проектування баз даних

Проектування баз даних є важливим аспектом архітектури веб-додатків. Основні підходи до проектування баз даних:

a. Нормалізація

Нормалізація передбачає розподіл даних по таблицях таким чином, щоб зменшити надлишковість та уникнути аномалій оновлення. Це забезпечує цілісність та узгодженість даних.

b. Денормалізація

Денормалізація передбачає об'єднання даних в одну таблицю для підвищення продуктивності запитів. Це може призвести до надлишковості даних, але знижує витрати на виконання складних запитів.

c. Використання індексів

Індекси підвищують продуктивність запитів до бази даних, дозволяючи швидко знаходити необхідні дані. Важливо правильно проектувати індекси для досягнення оптимальної продуктивності.

4. Безпека та захист даних

Забезпечення безпеки та захисту даних є критичним аспектом архітектури веб-додатків. Основні принципи безпеки:

a. Захист від SQL-ін'єкцій

Використання підготовлених виразів та ORM для взаємодії з базою даних запобігає SQL-ін'єкціям.

b. Використання HTTPS

Шифрування даних під час передачі за допомогою HTTPS забезпечує захист даних від перехоплення та модифікації.

c. Аутентифікація та авторизація

Використання надійних методів аутентифікації та авторизації запобігає несанкціонованому доступу до ресурсів додатка.

5. Тестування та налагодження

Тестування є важливим етапом розробки веб-додатків, що забезпечує якість та надійність системи. Основні види тестування:

a. Юніт-тестування

Юніт-тестування перевіряє окремі компоненти додатка на коректність їх роботи. Для цього використовуються фреймворки, такі як pytest для Python.

b. Інтеграційне тестування

Інтеграційне тестування перевіряє взаємодію між різними компонентами додатка, виявляючи проблеми в інтеграції.

c. Тестування користувацького інтерфейсу (UI)

UI-тестування перевіряє, як користувачі взаємодіють з додатком. Для цього використовуються інструменти для автоматизації тестування, такі як Selenium.

d. Нагрузочне тестування

Нагрузочне тестування перевіряє, як додаток поводиться під великим навантаженням, та виявляє можливі проблеми з продуктивністю.

Архітектура та дизайн веб-додатків мають ключове значення для створення ефективних, надійних та зручних для користувачів систем. Вибір правильної архітектури, забезпечення безпеки, продуманий дизайн інтерфейсу та належне тестування дозволяють створювати додатки, які відповідають сучасним вимогам та очікуванням користувачів.

4. ПРАКТИЧНА ЧАСТИНА

4.1. Опис розробки програмного забезпечення

У цьому розділі буде описано процес розробки соціальної мережі для студентів з використанням мови програмування Python та фреймворку Flask. Ми розглянемо структуру проекту, ключові компоненти коду та реалізовані функції.

1. Структура проекту

Проект організовано у вигляді стандартної структури Flask-додатку:

```
student_social_network/
├── __pycache__/
├── instance/
├── migrations/
├── static/
│   ├── img/
│   └── styles.css
├── templates/
│   ├── create_post.html
│   ├── edit_post.html
│   ├── edit_profile.html
│   ├── home.html
│   ├── index.html
│   ├── login.html
│   ├── messages.html
│   ├── news.html
│   ├── post.html
│   ├── register.html
│   ├── search.html
│   ├── send_message.html
│   └── user_profile.html
```

```
├── venv/
├── app.py
├── config.py
├── database.py
├── models.py
└── routes.py
```

2. Основні файли проекту

a. app.py

Головний файл додатку, який ініціалізує Flask-додаток, підключає розширення та налаштовує маршрутизацію.

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager
from flask_migrate import Migrate
from config import DevelopmentConfig

# Імпорт бази даних з зовнішнього файлу
from database import db

login_manager = LoginManager()

def create_app(config_class=DevelopmentConfig):
    app = Flask(__name__)
    app.config.from_object(config_class)

    # Ініціалізація розширень Flask з поточним додатком
    db.init_app(app)
    login_manager.init_app(app)
    migrate = Migrate(app, db)
```

```
    # Налаштування сторінки для неавторизованих користувачів
    login_manager.login_view = 'login'  # Вказуємо ім'я функції маршруту 'login'

    # Функція для завантаження даних користувача
    @login_manager.user_loader
    def load_user(user_id):
        from models import User
        return db.session.get(User, int(user_id))  # Використовуємо метод get для отримання користувача

    # Реєстрація маршрутів і моделей
    with app.app_context():
        from models import User
        from routes import init_routes
        init_routes(app)

        # Створення таблиць у базі даних, якщо вони ще не створені
        if db.engine.url.drivername == 'sqlite':
            db.create_all()

    return app

app = create_app()

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

b. config.py

Файл конфігурації для налаштування параметрів додатку.

```
import os
```

```
class Config:
    SECRET_KEY = os.environ.get('SECRET_KEY', 'secret')
        SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL',
'sqlite:///site.db')
    SQLALCHEMY_TRACK_MODIFICATIONS = False
      ENV = os.environ.get('FLASK_ENV', 'production')  # Задаємо значення за
замовчуванням
    UPLOAD_FOLDER = 'static/img'

class DevelopmentConfig(Config):
    DEBUG = True
    ENV = 'development'  # Явно вказуємо, що це розробницьке середовище

class TestingConfig(Config):
    TESTING = True
    SQLALCHEMY_DATABASE_URI = 'sqlite:///test.db'
    ENV = 'testing'

class ProductionConfig(Config):
    DEBUG = False
    ENV = 'production'
```

c. database.py

Файл для ініціалізації бази даних SQLAlchemy.

```
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()
```

d. models.py

Файл, який містить моделі бази даних та форми.

```
from datetime import datetime
from flask_login import UserMixin
```

```
from database import db
from flask_wtf import FlaskForm
from wtforms import BooleanField, PasswordField, StringField, SubmitField, TextAreaField, FileField
from wtforms.validators import DataRequired, Length, EqualTo, Optional, Regexp
from flask_wtf.file import FileField, FileAllowed

class User(UserMixin, db.Model):
    __tablename__ = 'user'
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(100), unique=True, nullable=False)
    password = db.Column(db.String(200), nullable=False)
    first_name = db.Column(db.String(100), nullable=False)
    last_name = db.Column(db.String(100), nullable=False)
    about_me = db.Column(db.String(250))
      group_name = db.Column(db.String(50), nullable=True)  # Нове поле для назви групи
      group_number = db.Column(db.String(50), nullable=True)  # Нове поле для номера групи
    avatar = db.Column(db.String(120), nullable=True)
    messages_sent = db.relationship('Message',
                                    foreign_keys='Message.sender_id',
                                    backref='author', lazy='dynamic')
    messages_received = db.relationship('Message',
                                        foreign_keys='Message.recipient_id',
                                        backref='recipient', lazy='dynamic')
    posts = db.relationship('Post', backref='author', lazy='dynamic')

    # Відносини будуть оновлені після визначення класу Post
```

```
class Post(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    body = db.Column(db.String(140))
    timestamp = db.Column(db.DateTime, index=True, default=datetime.utcnow)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))
    comments = db.relationship('Comment', backref='post', lazy='dynamic')
    media = db.Column(db.String(255))

# Тепер можна безпечно визначити асоціативну таблицю
likes = db.Table('likes',
    db.Column('user_id', db.Integer, db.ForeignKey('user.id'), primary_key=True),
    db.Column('post_id', db.Integer, db.ForeignKey('post.id'), primary_key=True),
    db.Column('timestamp', db.DateTime, index=True, default=datetime.utcnow)
)

# Оновлення відносин у User
User.liked_posts = db.relationship('Post', secondary=likes,
                                   primaryjoin=(likes.c.user_id == User.id),
                                   secondaryjoin=(likes.c.post_id == Post.id),
                                   backref=db.backref('liked_by', lazy='dynamic'),
                                   lazy='dynamic')

User.like_post = lambda self, post: self.liked_posts.append(post) if not self.has_liked_post(post) else None
User.unlike_post = lambda self, post: self.liked_posts.remove(post) if self.has_liked_post(post) else None
User.has_liked_post = lambda self, post: db.session.query(likes).filter_by(user_id=self.id, post_id=post.id).count() > 0

class Comment(db.Model):
```

```
    id = db.Column(db.Integer, primary_key=True)
    body = db.Column(db.String(140))
    timestamp = db.Column(db.DateTime, index=True, default=datetime.utcnow)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))
    post_id = db.Column(db.Integer, db.ForeignKey('post.id'))
        user = db.relationship('User', backref='user_comments')  # Змінено на 'user_comments'

    def __repr__(self):
        return f'<Comment {self.body}>'

class Message(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    sender_id = db.Column(db.Integer, db.ForeignKey('user.id'))
    recipient_id = db.Column(db.Integer, db.ForeignKey('user.id'))
    body = db.Column(db.String(140))
    timestamp = db.Column(db.DateTime, index=True, default=datetime.utcnow)

    def __repr__(self):
        return '<Message {}>'.format(self.body)

class PostForm(FlaskForm):
    body = StringField('Say something', validators=[DataRequired()])
      media = FileField('Attach a photo or video', validators=[FileAllowed(['jpg', 'jpeg', 'png', 'gif', 'mp4'], 'Only images and videos are allowed!')])
    submit = SubmitField('Submit')

class EditPostForm(FlaskForm):
    body = TextAreaField('Текст', validators=[DataRequired()])
    submit = SubmitField('Оновити')
```

```
class CommentForm(FlaskForm):
    body = StringField('Коментар', validators=[DataRequired()])
    submit = SubmitField('Надіслати')

class EditProfileForm(FlaskForm):
    avatar = FileField('Змінити аватар', validators=[FileAllowed(['jpg', 'png'],
'Тільки зображення!')])
    submit = SubmitField('Зберегти зміни')
    first_name = StringField('Ім’я', validators=[DataRequired()])
    last_name = StringField('Прізвище', validators=[DataRequired()])
    username = StringField('Логін', validators=[DataRequired()])
    about_me = TextAreaField('Про мене', validators=[Length(min=0, max=140)])
    group_name = StringField('Назва групи', validators=[Optional()])
    group_number = StringField('Номер групи', validators=[Optional()])
    change_password = BooleanField('Змінити пароль')
    old_password = PasswordField('Старий пароль', validators=[Optional()])
    new_password = PasswordField('Новий пароль', validators=[Optional()])
    confirm = PasswordField('Підтвердіть новий пароль', validators=[
      EqualTo('new_password', message='Паролі мають співпадати')])
    submit = SubmitField('Зберегти зміни')

class MessageForm(FlaskForm):
    message = TextAreaField('Повідомлення', validators=[DataRequired(),
Length(min=0, max=140)])
    submit = SubmitField('Відправити')

class SearchForm(FlaskForm):
    query = StringField('Шукати', validators=[DataRequired()])
    submit = SubmitField('Пошук')
```

```
class RegistrationForm(FlaskForm):
    username = StringField('Username', validators=[
        DataRequired(),
        Length(min=3, max=20),
            Regexp(r'^[A-Za-z0-9]+$', message="Логін повинен містити тільки англійські букви та цифри.")
    ])
    first_name = StringField('First Name', validators=[
        DataRequired(),
        Length(min=1, max=50),
           Regexp(r'^[\u0400-\u04FF]+$', message="Ім'я повинно містити тільки кирилицю.")
    ])
    last_name = StringField('Last Name', validators=[
        DataRequired(),
        Length(min=1, max=50),
            Regexp(r'^[\u0400-\u04FF]+$', message="Прізвище повинно містити тільки кирилицю.")
    ])
    password = PasswordField('Password', validators=[
        DataRequired(),
        Length(min=8, message="Пароль повинен бути мінімум 8 символів."),
                Regexp(r'^(?=.*[A-Za-z])(?=.*\d)[A-Za-z\d]{8,}$',  message="Пароль повинен містити букви та цифри.")
    ])
    confirm_password = PasswordField('Confirm Password', validators=[
        DataRequired(),
        EqualTo('password', message='Паролі повинні співпадати.')
    ])
```

```
    submit = SubmitField('Register')


class LoginForm(FlaskForm):
    username = StringField('Username', validators=[
        DataRequired(),
            Regexp(r'^[A-Za-z0-9]+$', message="Логін повинен містити тільки англійські букви та цифри.")
    ])
    password = PasswordField('Password', validators=[DataRequired()])
    submit = SubmitField('Log In')
```

e. routes.py

Файл, який містить маршрути додатку.

```
from datetime import datetime
from os import abort
import os
from flask import render_template, request, redirect, url_for, flash
from werkzeug.security import generate_password_hash, check_password_hash
from flask_login import login_user, logout_user, login_required, current_user
from models import EditPostForm, EditProfileForm, LoginForm, PostForm, User, Post, Comment, CommentForm, Message, MessageForm, SearchForm, RegistrationForm
from flask import current_app as app
from database import db
from werkzeug.utils import secure_filename


def init_routes(app):
    @app.route('/register', methods=['GET', 'POST'])
    def register():
        if current_user.is_authenticated:
            return redirect(url_for('index'))
```

```
    form = RegistrationForm()
    if form.validate_on_submit():
        first_name = form.first_name.data
        last_name = form.last_name.data
        username = form.username.data
        password = form.password.data

        existing_user = User.query.filter_by(username=username).first()
        if existing_user:
            flash('Цей логін вже зайнятий. Спробуйте інший.', 'warning')
            return redirect(url_for('register'))

        hashed_password = generate_password_hash(password)
                new_user = User(first_name=first_name, last_name=last_name,
username=username, password=hashed_password)
        db.session.add(new_user)
        db.session.commit()

        flash('Реєстрація успішна. Тепер ви можете увійти.', 'success')
        return redirect(url_for('login'))

    return render_template('register.html', form=form)

@app.route('/login', methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('index'))

    form = LoginForm()
    if form.validate_on_submit():
```

```
            username = form.username.data
            password = form.password.data

            user = User.query.filter_by(username=username).first()

            if user and check_password_hash(user.password, password):
                login_user(user, remember=True)  # Optionally handle 'remember me' functionality
                next_page = request.args.get('next')
                return redirect(next_page) if next_page else redirect(url_for('index'))
            else:
                flash('Неправильний логін або пароль.', 'danger')  # Adding bootstrap class for styling

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

    @app.route('/logout')
    @login_required
    def logout():
        logout_user()
        return redirect(url_for('login'))

    @app.route('/')
    @app.route('/index')
    @login_required
    def index():
        posts = Post.query.order_by(Post.timestamp.desc()).all()
        return render_template('index.html', posts=posts)

    @app.route('/news')
```

```
@login_required
def news():
    posts = Post.query.order_by(Post.created_at.desc()).all()
    return render_template('news.html', posts=posts)

@app.route('/create_post', methods=['GET', 'POST'])
@login_required
def create_post():
    form = PostForm()
    if form.validate_on_submit():
        post = Post(body=form.body.data, author=current_user)
        if form.media.data:
            media_file = form.media.data
            filename = secure_filename(media_file.filename)
            filepath = os.path.join(app.config['UPLOAD_FOLDER'], filename)
            media_file.save(filepath)
            post.media = filename  # Зберігаємо лише ім'я файлу
        db.session.add(post)
        db.session.commit()
        flash('Ваш пост було успішно опубліковано!')
        return redirect(url_for('index'))
    return render_template('create_post.html', form=form)

@app.route('/edit_post/<int:post_id>', methods=['GET', 'POST'])
@login_required
def edit_post(post_id):
    post = Post.query.get_or_404(post_id)
    if post.author != current_user:
        flash('You do not have permission to edit this post.')
        return redirect(url_for('index'))
```

```
    form = EditPostForm()
    if form.validate_on_submit():
        post.body = form.body.data
        db.session.commit()
        flash('Your post has been updated.')
        return redirect(url_for('post', post_id=post.id))

    form.body.data = post.body
            return render_template('edit_post.html', title='Редагування Поста', form=form, post=post)

@app.route('/delete_post/<int:post_id>', methods=['POST'])
@login_required
def delete_post(post_id):
    post = Post.query.get_or_404(post_id)
    if post.author != current_user:
        abort(403)  # Забороняємо видалення, якщо користувач не є автором посту
    db.session.delete(post)
    db.session.commit()
    flash('Ваш пост було успішно видалено!', 'success')
    return redirect(url_for('index'))

@app.route('/post/<int:post_id>', methods=['GET', 'POST'])
@login_required
def post(post_id):
    post = Post.query.get_or_404(post_id)
    form = CommentForm()
    if form.validate_on_submit():
```

```
                comment = Comment(body=form.body.data, user_id=current_user.id,
post_id=post.id)
                db.session.add(comment)
                db.session.commit()
                flash('Ваш коментар було додано.')
                return redirect(url_for('post', post_id=post.id))
            comments = post.comments.order_by(Comment.timestamp.asc())
                        return render_template('post.html', post=post, form=form,
comments=comments)

        @app.route('/like/<int:post_id>', methods=['POST'])
        @login_required
        def like(post_id):
            post = Post.query.get_or_404(post_id)
            current_user.like_post(post)
            db.session.commit()
            return redirect(request.referrer)

        @app.route('/unlike/<int:post_id>', methods=['POST'])
        @login_required
        def unlike(post_id):
            post = Post.query.get_or_404(post_id)
            current_user.unlike_post(post)
            db.session.commit()
            return redirect(request.referrer)

        @app.route('/user/<username>')
        @login_required
        def user_profile(username):
            user = User.query.filter_by(username=username).first_or_404()
```

```
                                                                    posts =
Post.query.filter_by(user_id=user.id).order_by(Post.timestamp.desc()).all()
        return render_template('user_profile.html', user=user, posts=posts)

    @app.route('/edit_profile', methods=['GET', 'POST'])
    @login_required
    def edit_profile():
        form = EditProfileForm(obj=current_user)
        if form.validate_on_submit():
            current_user.username = form.username.data
            current_user.first_name = form.first_name.data
            current_user.last_name = form.last_name.data
            current_user.group_name = form.group_name.data
            current_user.group_number = form.group_number.data
            current_user.about_me = form.about_me.data

            # Обробка завантаження аватару
            if form.avatar.data:
                avatar_file = form.avatar.data
                if hasattr(avatar_file, 'filename'):
                    filename = secure_filename(avatar_file.filename)
                    save_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
                    # Створіть директорію, якщо вона не існує
                    if not os.path.exists(app.config['UPLOAD_FOLDER']):
                        os.makedirs(app.config['UPLOAD_FOLDER'])
                    avatar_file.save(save_path)
                    current_user.avatar = filename

            db.session.commit()
            flash('Зміни профілю було збережено.', 'success')
```

```
            return redirect(url_for('user_profile', username=current_user.username))

        return render_template('edit_profile.html', form=form)

    @app.route('/messages')
    @login_required
    def messages():
        # Отримання діалогів з останніми повідомленнями
        subquery = db.session.query(
                                        Message.sender_id,    Message.recipient_id,
db.func.max(Message.timestamp).label('last_msg_time')
        ).filter(
                (Message.sender_id == current_user.id) | (Message.recipient_id ==
current_user.id)
        ).group_by(
            Message.sender_id, Message.recipient_id
        ).subquery()

        last_messages = db.session.query(Message).join(
            subquery, db.or_(
                db.and_(
                    Message.sender_id == subquery.c.sender_id,
                    Message.recipient_id == subquery.c.recipient_id,
                    Message.timestamp == subquery.c.last_msg_time
                ),
                db.and_(
                    Message.sender_id == subquery.c.recipient_id,
                    Message.recipient_id == subquery.c.sender_id,
                    Message.timestamp == subquery.c.last_msg_time
                )
```

```
            )
        ).order_by(Message.timestamp.desc())

        return render_template('messages.html', messages=last_messages)

    @app.route('/send_message/<username>', methods=['GET', 'POST'])
    @login_required
    def send_message(username):
        user = User.query.filter_by(username=username).first_or_404()
        form = MessageForm()
        if form.validate_on_submit():
            message = Message(sender_id=current_user.id, recipient_id=user.id,
body=form.message.data)
            db.session.add(message)
            db.session.commit()
            flash('Your message has been sent.')
            return redirect(url_for('send_message', username=username))

        messages = Message.query.filter(
            db.or_(
                db.and_(Message.sender_id == current_user.id, Message.recipient_id
== user.id),
                db.and_(Message.sender_id == user.id, Message.recipient_id ==
current_user.id)
            )
        ).order_by(Message.timestamp.asc())
            return render_template('send_message.html', form=form, recipient=user,
messages=messages)

    @app.route('/search', methods=['GET', 'POST'])
```

```
@login_required
def search():
    form = SearchForm()
    users = []
    if form.validate_on_submit():
        query = form.query.data
        # Змінено фільтрацію для пошуку за іменем або прізвищем
        users = User.query.filter(
            db.or_(
                User.first_name.ilike(f'%{query}%'),
                User.last_name.ilike(f'%{query}%')
            )
        ).all()
        if not users:
            flash('Немає користувачів з таким іменем або прізвищем.', 'info')
    return render_template('search.html', form=form, users=users)
```

3. Основні функціональні можливості

a. Реєстрація та авторизація користувачів

Користувачі можуть реєструватися та авторизуватися у системі, що дозволяє їм створювати особисті профілі та публікувати контент.

b. Створення та редагування постів

Користувачі можуть створювати нові пости, редагувати та видаляти їх. Пости можуть містити текст та зображення.

c. Редагування профілю

Користувачі можуть редагувати свій профіль, включаючи зміну імені, електронної пошти та зображення профілю.

4. Використані технології та інструменти

Для розробки соціальної мережі були використані наступні технології та інструменти:

- **Python**: основна мова програмування для розробки бекенду.

- **Flask**: фреймворк для веб-розробки.
- **SQLAlchemy**: ORM для роботи з базою даних.
- **WTForms**: бібліотека для створення та валідації форм.
- **Flask-Login**: розширення для управління аутентифікацією користувачів.
- **Flask-Migrate**: інструмент для управління міграціями бази даних.
- **Jinja2**: шаблонізатор для створення динамічних HTML-сторінок.
- **Bootstrap**: фреймворк для створення адаптивного дизайну.

Цей розділ описує процес розробки програмного забезпечення для соціальної мережі для студентів. Використання Flask та супутніх інструментів дозволяє створити потужний, гнучкий та масштабований веб-додаток, який забезпечує всі необхідні функції для спілкування та співпраці студентів.

4.2. Реалізація функціональності для користувача

Соціальна мережа для студентів реалізує декілька ключових функціональних можливостей для користувачів. У цьому розділі буде детально розглянуто реалізацію основних функцій, включаючи реєстрацію та авторизацію користувачів, створення та управління постами, редагування профілю, обмін повідомленнями та пошук користувачів.

1. Реєстрація та авторизація користувачів

Користувачі можуть реєструватися, заповнивши реєстраційну форму, яка перевіряє введені дані на відповідність встановленим вимогам, включаючи унікальність імені користувача та електронної пошти. Після успішної реєстрації користувач може увійти до системи за допомогою форми входу, яка перевіряє правильність введених облікових даних.

```
@app.route('/register', methods=['GET', 'POST'])
def register():
    if current_user.is_authenticated:
        return redirect(url_for('index'))
```

```
    form = RegistrationForm()
    if form.validate_on_submit():
        first_name = form.first_name.data
        last_name = form.last_name.data
        username = form.username.data
        password = form.password.data

        existing_user = User.query.filter_by(username=username).first()
        if existing_user:
            flash('Цей логін вже зайнятий. Спробуйте інший.', 'warning')
            return redirect(url_for('register'))

        hashed_password = generate_password_hash(password)
        new_user = User(first_name=first_name, last_name=last_name,
username=username, password=hashed_password)
        db.session.add(new_user)
        db.session.commit()

        flash('Реєстрація успішна. Тепер ви можете увійти.', 'success')
        return redirect(url_for('login'))

    return render_template('register.html', form=form)

@app.route('/login', methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('index'))

    form = LoginForm()
    if form.validate_on_submit():
```

```
        username = form.username.data
        password = form.password.data

        user = User.query.filter_by(username=username).first()

        if user and check_password_hash(user.password, password):
            login_user(user, remember=True)
            next_page = request.args.get('next')
            return redirect(next_page) if next_page else redirect(url_for('index'))
        else:
            flash('Неправильний логін або пароль.', 'danger')

    return render_template('login.html', form=form)
```

2. Створення та управління постами

Користувачі можуть створювати нові пости, редагувати та видаляти свої пости. Пости можуть містити текст та медіафайли (зображення чи відео), які зберігаються у відповідній директорії на сервері.

Створення поста:

```
@app.route('/create_post', methods=['GET', 'POST'])
@login_required
def create_post():
    form = PostForm()
    if form.validate_on_submit():
        post = Post(body=form.body.data, author=current_user)
        if form.media.data:
            media_file = form.media.data
            filename = secure_filename(media_file.filename)
            filepath = os.path.join(app.config['UPLOAD_FOLDER'], filename)
            media_file.save(filepath)
            post.media = filename
```

```
        db.session.add(post)
        db.session.commit()
        flash('Ваш пост було успішно опубліковано!')
        return redirect(url_for('index'))
    return render_template('create_post.html', form=form)
```

Редагування поста:

```
@app.route('/edit_post/<int:post_id>', methods=['GET', 'POST'])
@login_required
def edit_post(post_id):
    post = Post.query.get_or_404(post_id)
    if post.author != current_user:
        flash('You do not have permission to edit this post.')
        return redirect(url_for('index'))

    form = EditPostForm()
    if form.validate_on_submit():
        post.body = form.body.data
        db.session.commit()
        flash('Your post has been updated.')
        return redirect(url_for('post', post_id=post.id))

    form.body.data = post.body
    return render_template('edit_post.html', title='Редагування Поста', form=form,
post=post)
```

Видалення поста:

```
@app.route('/delete_post/<int:post_id>', methods=['POST'])
@login_required
def delete_post(post_id):
    post = Post.query.get_or_404(post_id)
    if post.author != current_user:
```

```
        abort(403)
    db.session.delete(post)
    db.session.commit()
    flash('Ваш пост було успішно видалено!', 'success')
    return redirect(url_for('index'))
```

3. Редагування профілю

Користувачі можуть редагувати свої профілі, змінюючи інформацію, таку як ім'я користувача, електронну пошту, біографію та завантажуючи нові аватари.

```
@app.route('/edit_profile', methods=['GET', 'POST'])
@login_required
def edit_profile():
    form = EditProfileForm(obj=current_user)
    if form.validate_on_submit():
        current_user.username = form.username.data
        current_user.first_name = form.first_name.data
        current_user.last_name = form.last_name.data
        current_user.group_name = form.group_name.data
        current_user.group_number = form.group_number.data
        current_user.about_me = form.about_me.data

        if form.avatar.data:
            avatar_file = form.avatar.data
            if hasattr(avatar_file, 'filename'):
                filename = secure_filename(avatar_file.filename)
                save_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
                if not os.path.exists(app.config['UPLOAD_FOLDER']):
                    os.makedirs(app.config['UPLOAD_FOLDER'])
                avatar_file.save(save_path)
                current_user.avatar = filename
```

```
            db.session.commit()
            flash('Зміни профілю було збережено.', 'success')
            return redirect(url_for('user_profile', username=current_user.username))

        return render_template('edit_profile.html', form=form)
```

4. Обмін повідомленнями

Користувачі можуть надсилати та отримувати приватні повідомлення. Повідомлення зберігаються у базі даних та відображаються на сторінці повідомлень.

```
    @app.route('/send_message/<username>', methods=['GET', 'POST'])
    @login_required
    def send_message(username):
        user = User.query.filter_by(username=username).first_or_404()
        form = MessageForm()
        if form.validate_on_submit():
            message = Message(sender_id=current_user.id, recipient_id=user.id,
body=form.message.data)
            db.session.add(message)
            db.session.commit()
            flash('Your message has been sent.')
            return redirect(url_for('send_message', username=username))

        messages = Message.query.filter(
            db.or_(
                db.and_(Message.sender_id == current_user.id, Message.recipient_id ==
user.id),
                db.and_(Message.sender_id == user.id, Message.recipient_id ==
current_user.id)
            )
        ).order_by(Message.timestamp.asc())
        return render_template('send_message.html', form=form, recipient=user,
```

```
messages=messages)
```

5. Пошук користувачів

Користувачі можуть шукати інших користувачів за іменем або прізвищем. Результати пошуку відображаються у вигляді списку користувачів, які відповідають запиту.

```
@app.route('/search', methods=['GET', 'POST'])
@login_required
def search():
    form = SearchForm()
    users = []
    if form.validate_on_submit():
        query = form.query.data
        users = User.query.filter(
            db.or_(
                User.first_name.ilike(f'%{query}%'),
                User.last_name.ilike(f'%{query}%')
            )
        ).all()
        if not users:
            flash('Немає користувачів з таким іменем або прізвищем.', 'info')
    return render_template('search.html', form=form, users=users)
```

6. Відображення профілів користувачів

Користувачі можуть переглядати профілі інших користувачів, де відображаються їхні пости та інформація про них.

```
@app.route('/user/<username>')
@login_required
def user_profile(username):
    user = User.query.filter_by(username=username).first_or_404()
    posts = Post.query.filter_by(user_id=user.id).order_by(Post.timestamp.desc()).all()
```

```
    return render_template('user_profile.html', user=user, posts=posts)
```

7. Відображення коментарів та лайків

Користувачі можуть залишати коментарі до постів інших користувачів та ставити лайки.

Додавання коментарів:

```
@app.route('/post/<int:post_id>', methods=['GET', 'POST'])
@login_required
def post(post_id):
    post = Post.query.get_or_404(post_id)
    form = CommentForm()
    if form.validate_on_submit():
        comment = Comment(body=form.body.data, user_id=current_user.id, post_id=post.id)
        db.session.add(comment)
        db.session.commit()
        flash('Ваш коментар було додано.')
        return redirect(url_for('post', post_id=post.id))
    comments = post.comments.order_by(Comment.timestamp.asc())
    return render_template('post.html', post=post, form=form, comments=comments)
```

Лайки та дизлайки:

```
@app.route('/like/<int:post_id>', methods=['POST'])
@login_required
def like(post_id):
    post = Post.query.get_or_404(post_id)
    current_user.like_post(post)
    db.session.commit()
    return redirect(request.referrer)

@app.route('/unlike/<int:post_id>', methods=['POST'])
@login_required
```

```
def unlike(post_id):
    post = Post.query.get_or_404(post_id)
    current_user.unlike_post(post)
    db.session.commit()
    return redirect(request.referrer)
```

Реалізація функціональності для користувачів у соціальній мережі для студентів включає створення зручного та інтуїтивно зрозумілого інтерфейсу для реєстрації та авторизації, створення та управління постами, редагування профілю, обміну повідомленнями, пошуку користувачів та взаємодії з постами через коментарі та лайки. Використання Flask та супутніх бібліотек дозволяє створити потужний, гнучкий та масштабований веб-додаток, що відповідає потребам користувачів.

4.3. Опис програми

Соціальна мережа для студентів є веб-додатком, розробленим з використанням Flask, який надає користувачам різноманітні функціональні можливості для спілкування, співпраці та обміну інформацією. Основна мета програми полягає в тому, щоб забезпечити студентів платформою для обміну думками, ідеями, ресурсами та спільної роботи над проектами. У цьому розділі буде описано основні компоненти програми та їх функціональні можливості.

1. Архітектура програми

Програма має класичну клієнт-серверну архітектуру, де клієнтська частина відповідає за взаємодію з користувачем, а серверна частина — за обробку запитів, управління даними та взаємодію з базою даних. Основні компоненти архітектури включають:

- **Клієнтська частина**: HTML-сторінки, CSS для стилізації та JavaScript для інтерактивності.
- **Серверна частина**: Flask-додаток, що обробляє HTTP-запити, управляє сесіями користувачів та виконує бізнес-логіку.

- **База даних**: SQLAlchemy ORM для взаємодії з реляційною базою даних (SQLite).

2. Основні компоненти програми

Програма складається з декількох основних компонентів, кожен з яких виконує певну функцію:

a. Реєстрація та авторизація користувачів

Функціонал реєстрації та авторизації дозволяє користувачам створювати нові облікові записи та входити до системи. Реєстраційна форма перевіряє унікальність імені користувача та електронної пошти, а також відповідність пароля встановленим вимогам. Після успішної реєстрації користувач може увійти до системи за допомогою електронної пошти та пароля.

b. Створення та управління постами

Користувачі можуть створювати нові пости, редагувати та видаляти їх. Пости можуть містити текст та зображення, що дозволяє користувачам ділитися думками, ідеями та ресурсами. Всі пости відображаються на головній сторінці, а також на сторінці профілю користувача.

c. Редагування профілю

Користувачі можуть редагувати свій профіль, включаючи зміну імені користувача, електронної пошти та завантаження зображення профілю. Це дозволяє користувачам підтримувати актуальну інформацію про себе.

d. Обмін повідомленнями

Функціонал обміну повідомленнями дозволяє користувачам надсилати та отримувати приватні повідомлення. Це сприяє більш тісній співпраці та обміну інформацією між користувачами.

e. Пошук користувачів

Користувачі можуть шукати інших користувачів за іменем або прізвищем, що полегшує знайомство та спілкування між студентами, які можуть мати спільні інтереси або працювати над схожими проектами.

f. Відображення профілів користувачів

Кожен користувач має власний профіль, де відображаються його особиста

інформація та всі створені ним пости. Інші користувачі можуть переглядати ці профілі, що сприяє кращому знайомству та співпраці.

3. Інтерфейс користувача

Інтерфейс користувача реалізований за допомогою шаблонів Jinja2 та стилізований з використанням CSS і фреймворку Bootstrap. Це забезпечує зручність використання та адаптивність інтерфейсу на різних пристроях.

4. Структура шаблонів

Шаблони HTML забезпечують відображення даних та інтерактивність інтерфейсу. Основні шаблони включають:

- **base.html**: базовий шаблон, який містить спільні для всіх сторінок елементи, такі як навігаційна панель та стилі.
- **index.html**: головна сторінка, де відображаються всі пости.
- **register.html**: форма реєстрації користувачів.
- **login.html**: форма входу до системи.
- **profile.html**: сторінка профілю користувача.
- **create_post.html**: форма створення нового поста.
- **edit_post.html**: форма редагування існуючого поста.
- **messages.html**: сторінка для перегляду та надсилання повідомлень.
- **search.html**: форма пошуку користувачів.
- **user_profile.html**: сторінка профілю іншого користувача.

5. Структура CSS

Файл styles.css містить стилі для різних елементів інтерфейсу, забезпечуючи їх привабливий вигляд та зручність використання.

6. База даних

Для зберігання даних користувачів, постів, повідомлень та іншої інформації використовується база даних SQLite, з якою взаємодіє SQLAlchemy ORM. Це забезпечує зручне управління даними та їх цілісність.

7. Логіка роботи програми

Всі маршрути (routes) додатку визначені у файлі routes.py, де обробляються HTTP-запити, перевіряються дані форм та виконується бізнес-логіка програми.

8. Використані бібліотеки та інструменти

Для розробки програми були використані наступні бібліотеки та інструменти:

- **Python**: основна мова програмування.
- **Flask**: мікрофреймворк для веб-розробки.
- **SQLAlchemy**: ORM для взаємодії з базою даних.
- **WTForms**: бібліотека для створення та валідації форм.
- **Flask-Login**: розширення для управління сесіями користувачів.
- **Flask-Migrate**: інструмент для управління міграціями бази даних.
- **Jinja2**: шаблонізатор для створення динамічних HTML-сторінок.
- **Bootstrap**: фреймворк для створення адаптивного дизайну.

Описана програма соціальної мережі для студентів надає користувачам усі необхідні інструменти для ефективного спілкування, співпраці та обміну інформацією. Використання Flask та супутніх бібліотек дозволяє створити гнучку, масштабовану та безпечну платформу для студентів, забезпечуючи зручний інтерфейс та багатий функціонал.

4.4. Інструкція для користувача

Цей розділ містить детальні інструкції для користувачів щодо встановлення, налаштування та використання веб-додатку соціальної мережі для студентів. Інструкція охоплює всі основні функції програми, включаючи реєстрацію та авторизацію, створення та управління постами, редагування профілю, обмін повідомленнями та пошук користувачів.

Встановлення та налаштування додатку

1. **Встановлення додатку:**
    - **Скачайте та розпакуйте проект:** Завантажте архів з кодом додатку з надійного джерела та розпакуйте його у зручному місці.
    - **Налаштування віртуального оточення:** Створіть віртуальне оточення за допомогою команди **python -m venv venv** та активуйте його:
        - Для Windows: **venv\Scripts\activate**

- Для Linux/Mac: **source venv/bin/activate**

- **Встановлення залежностей:** Встановіть необхідні залежності, виконавши команду: **pip install -r requirements.txt**.
- **Ініціалізація бази даних:** Ініціалізуйте базу даних за допомогою команд:

flask db init

flask db migrate -m "Initial migration"

flask db upgrade

- **Запуск додатку:** Запустіть додаток командою: **flask run**. Додаток буде доступний за адресою **http://127.0.0.1:5000**.

Використання додатку

2. **Реєстрація та авторизація:**
   - **Реєстрація нового користувача:** Відкрийте сторінку реєстрації, натиснувши кнопку "Реєстрація" на головній сторінці. Заповніть форму реєстрації, вказавши ім'я користувача та пароль. (див. рис. 4.1) Після успішної реєстрації ви будете перенаправлені на сторінку входу.

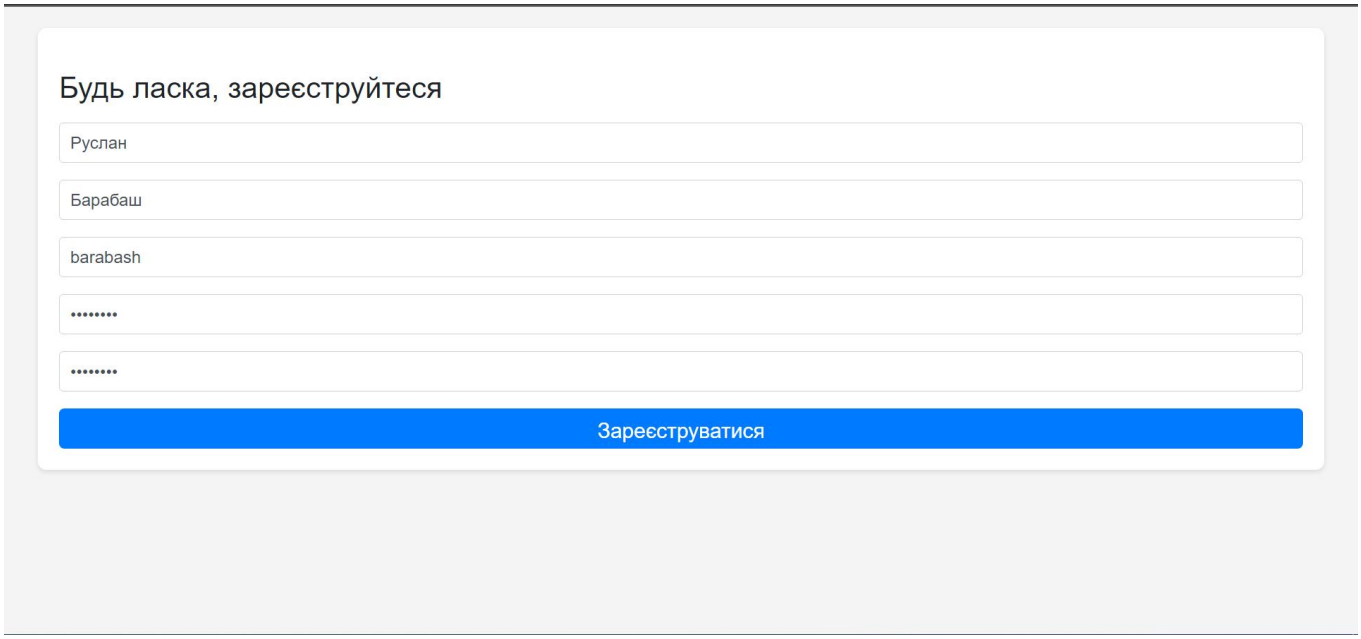


Рисунок 4.1 – Реєстрація користувача

- **Вхід у систему:** Відкрийте сторінку входу, натиснувши кнопку

"Увійти" на головній сторінці. Введіть своє ім'я користувача та пароль для входу в систему. (див. рис. 4.2) Після успішного входу ви будете перенаправлені на головну сторінку з переліком постів.

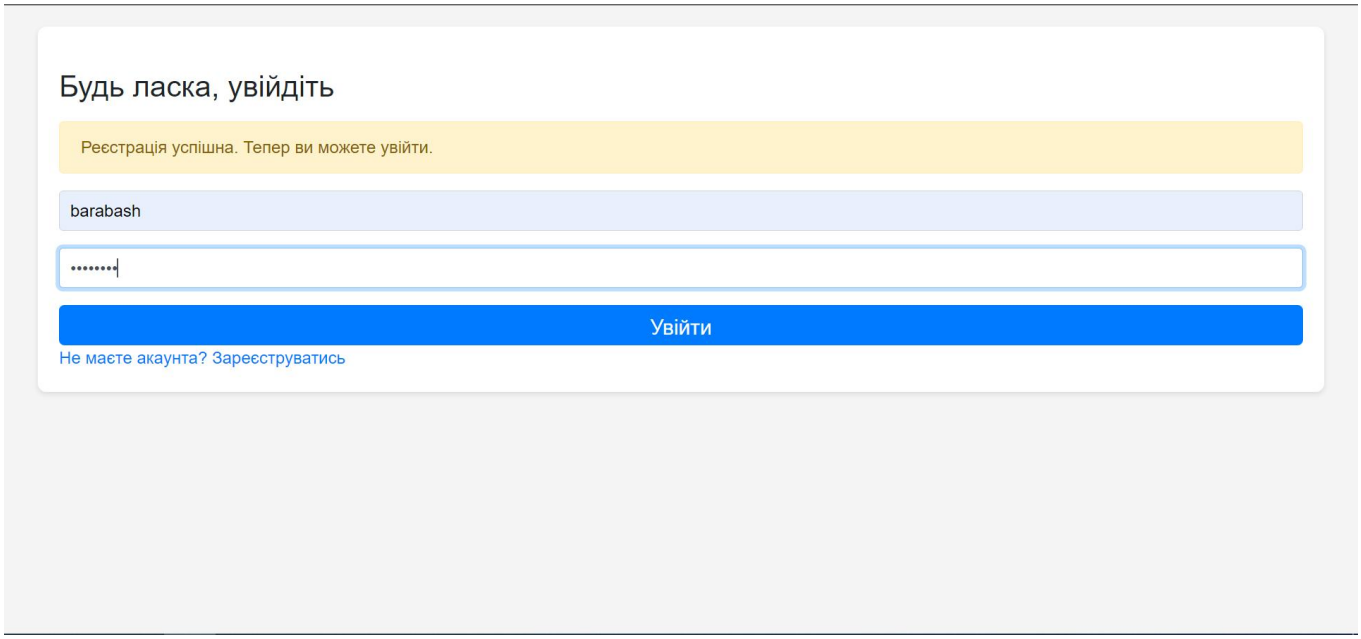


Рисунок 4-2 – Логін користувача

- **Вихід з системи:** Для виходу з системи натисніть кнопку "Вийти" у верхньому правому куті навігаційної панелі. Після виходу ви будете перенаправлені на сторінку входу.

3. **Управління постами:**
   - **Додавання поста:** Натисніть кнопку "Створити пост" на головній сторінці або в меню навігації. Заповніть форму додавання поста, вказавши заголовок та вміст. Натисніть кнопку "Опублікувати пост" для збереження поста. (див. рис. 4.3) Після успішного додавання ви будете перенаправлені на головну сторінку.

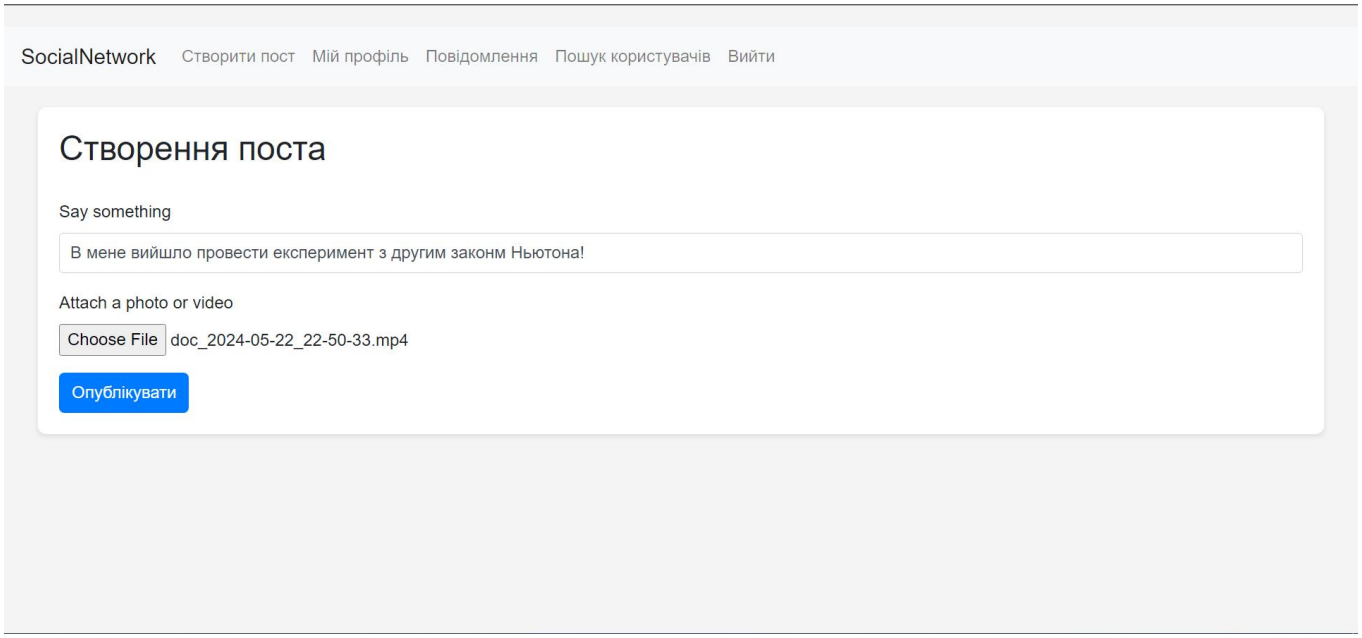


Рисунок 4.3 – Створення поста

- **Редагування поста:** На головній сторінці або в профілі користувача натисніть на пост, який ви хочете редагувати. (див. рис. 4.4) Натисніть кнопку "Редагувати" на сторінці поста, внесіть необхідні зміни у форму та натисніть кнопку "Оновити пост". Після успішного редагування ви будете перенаправлені на сторінку поста.

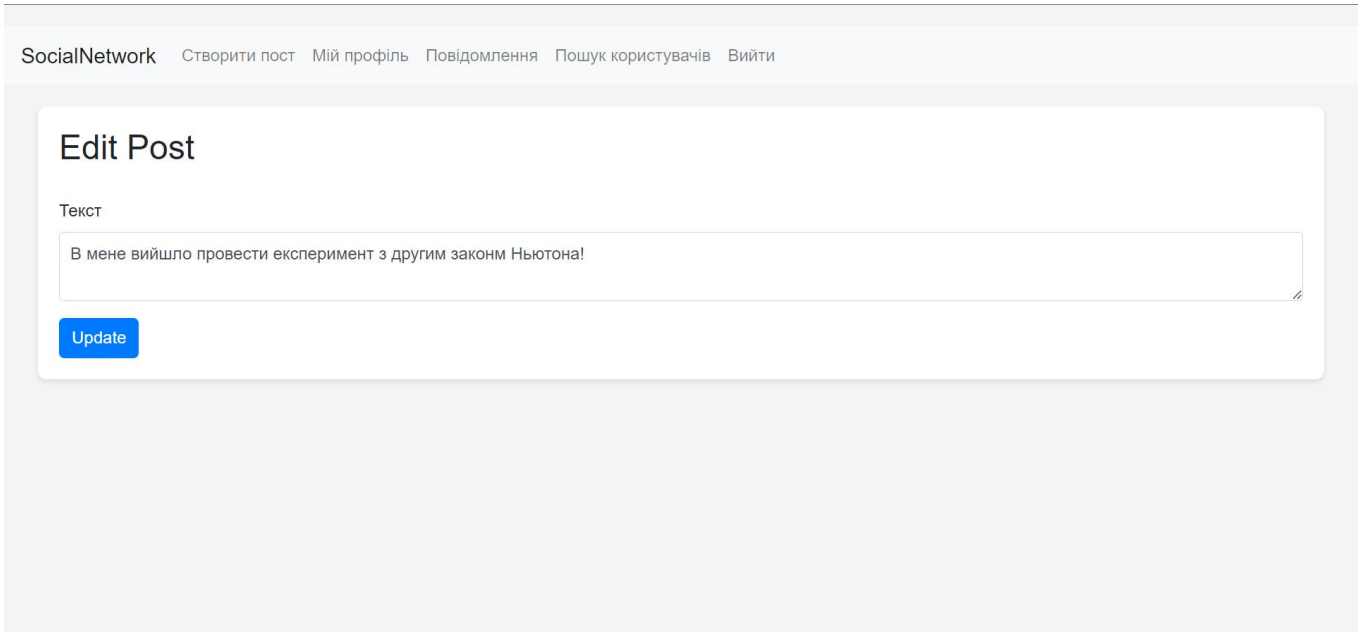


Рисунок 4.4 – Редагувати пост

- **Видалення поста:** На сторінці поста натисніть кнопку "Видалити".

Підтвердіть видалення у спливаючому вікні. (див. рис. 4.5) Після видалення поста ви будете перенаправлені на головну сторінку.

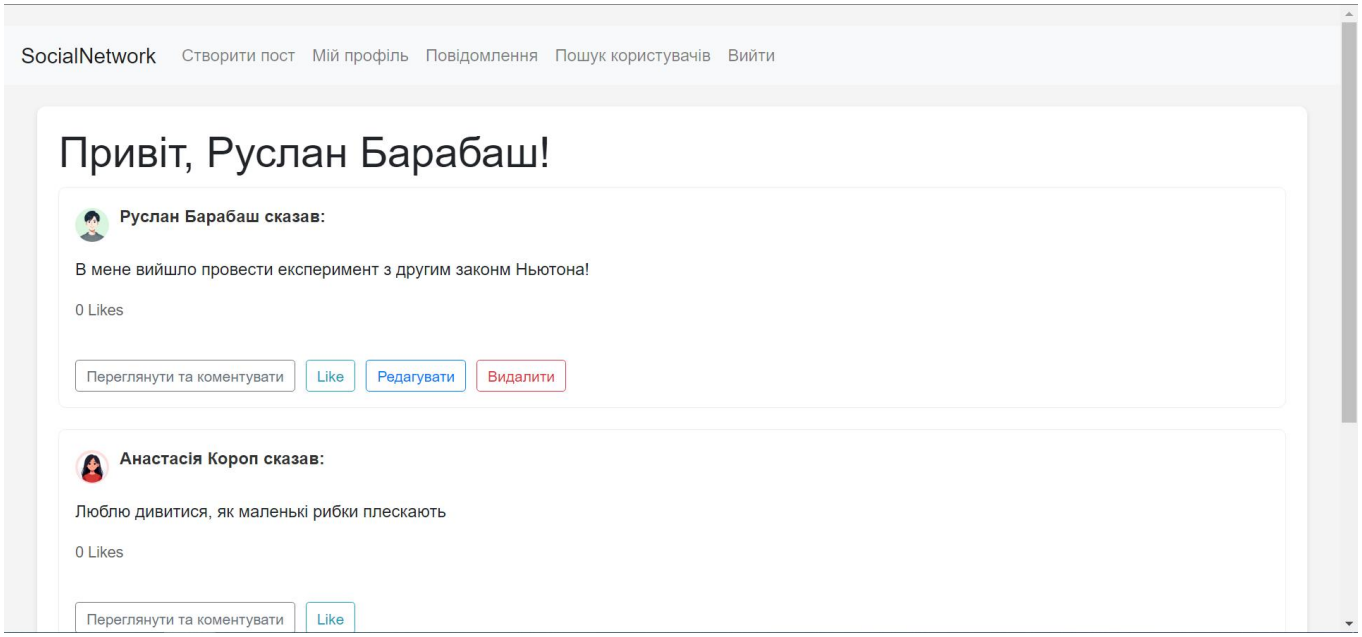


Рисунок 4.5 – Видалення поста

4. **Редагування профілю:**
   - **Редагування профілю:** На головній сторінці натисніть на ваше ім'я користувача у верхньому правому куті та виберіть "Профіль". Натисніть кнопку "Редагувати профіль", внесіть необхідні зміни у форму та натисніть кнопку "Зберегти зміни". (див. рис. 4.6-4.7) Після успішного редагування ви будете перенаправлені на сторінку вашого профілю.

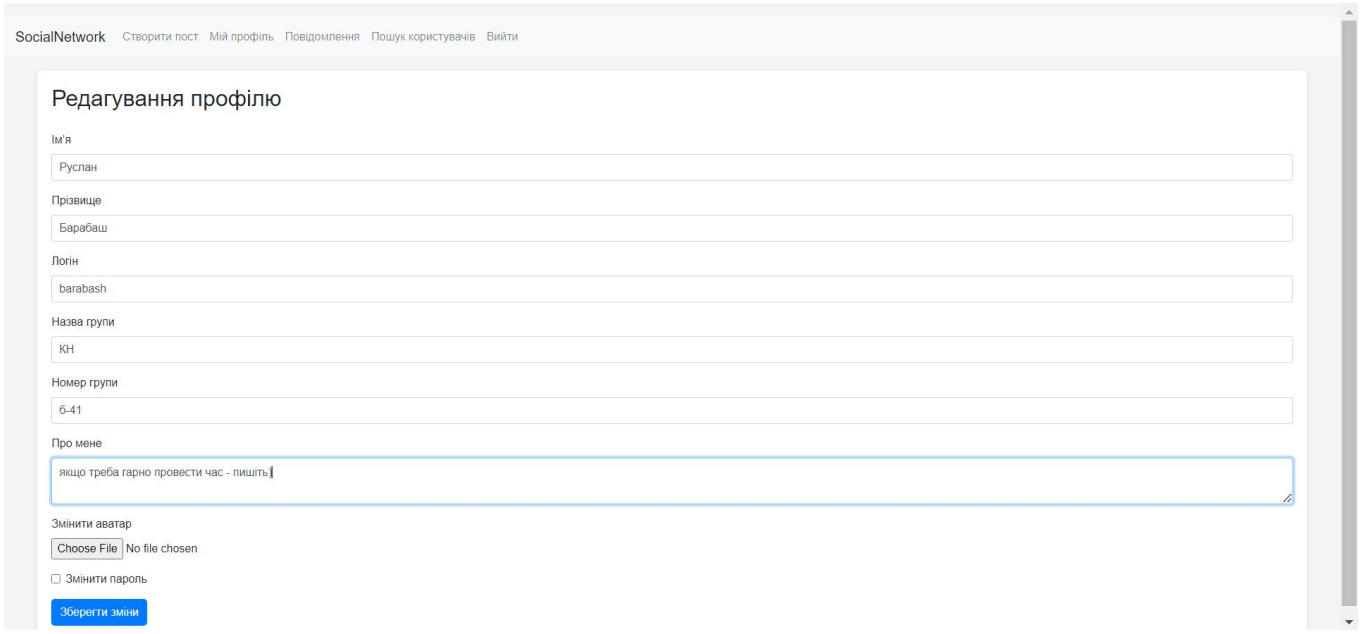
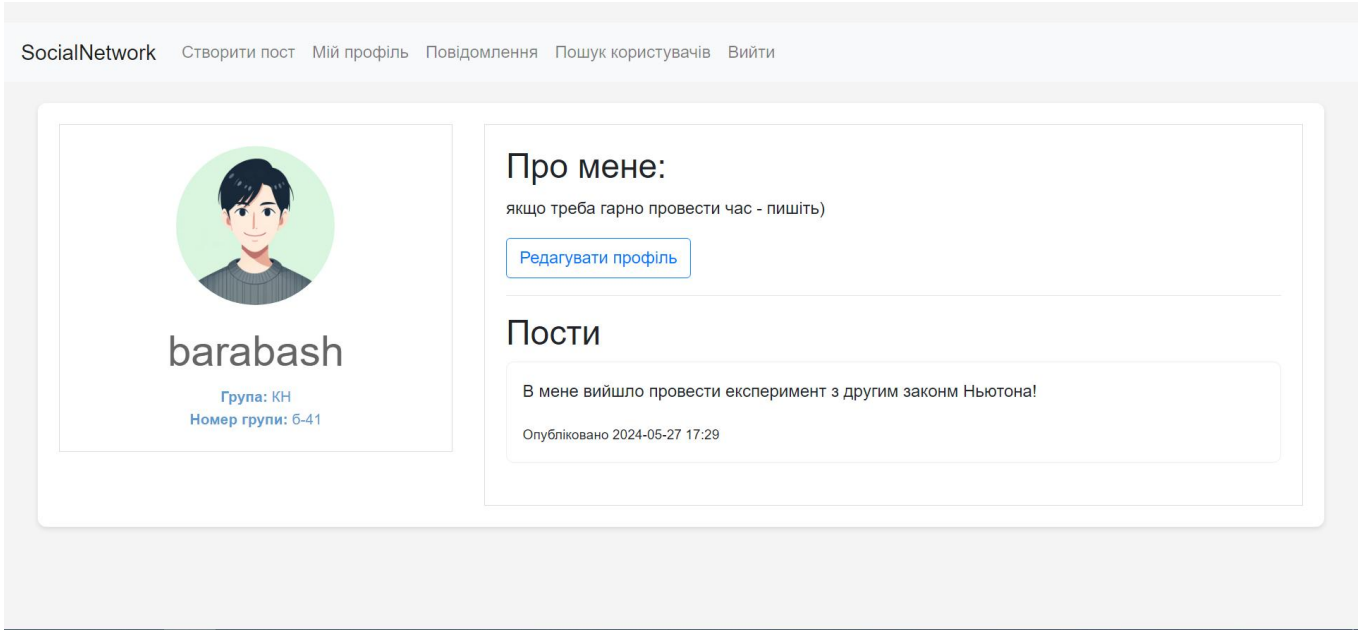
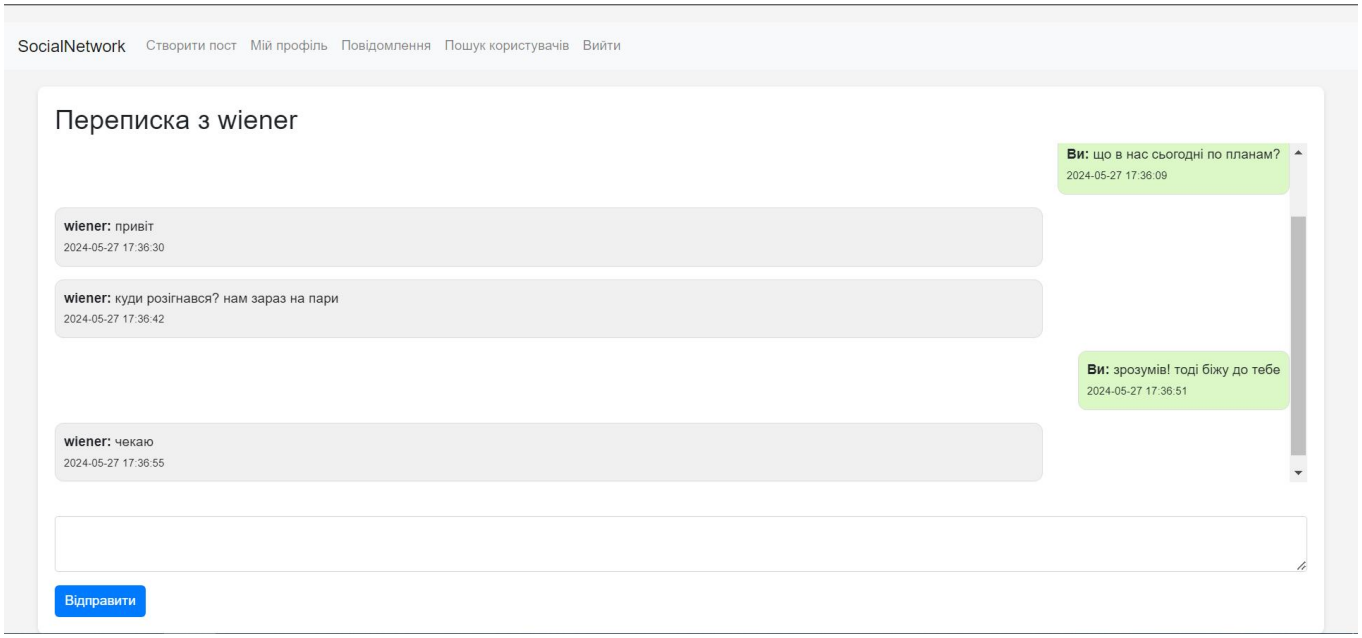


Рисунок 4.6 – Редагування профілю



4.7 – Профіль користувача

5. **Обмін повідомленнями:**
   - **Надсилання повідомлень:** Перейдіть на профіль користувача, якому ви хочете надіслати повідомлення. Натисніть кнопку "Надіслати повідомлення", заповніть форму та натисніть "Надіслати". (див. рис. 4.8) Після успішного надсилання повідомлення ви будете перенаправлені на сторінку повідомлень.



4.8 – Листування

6. **Пошук користувачів:**
    - **Пошук користувачів:** Натисніть кнопку "Пошук" у меню навігації. Заповніть поле пошуку та натисніть кнопку "Пошук". (див. рис. 4.9) Результати пошуку будуть відображені на сторінці.

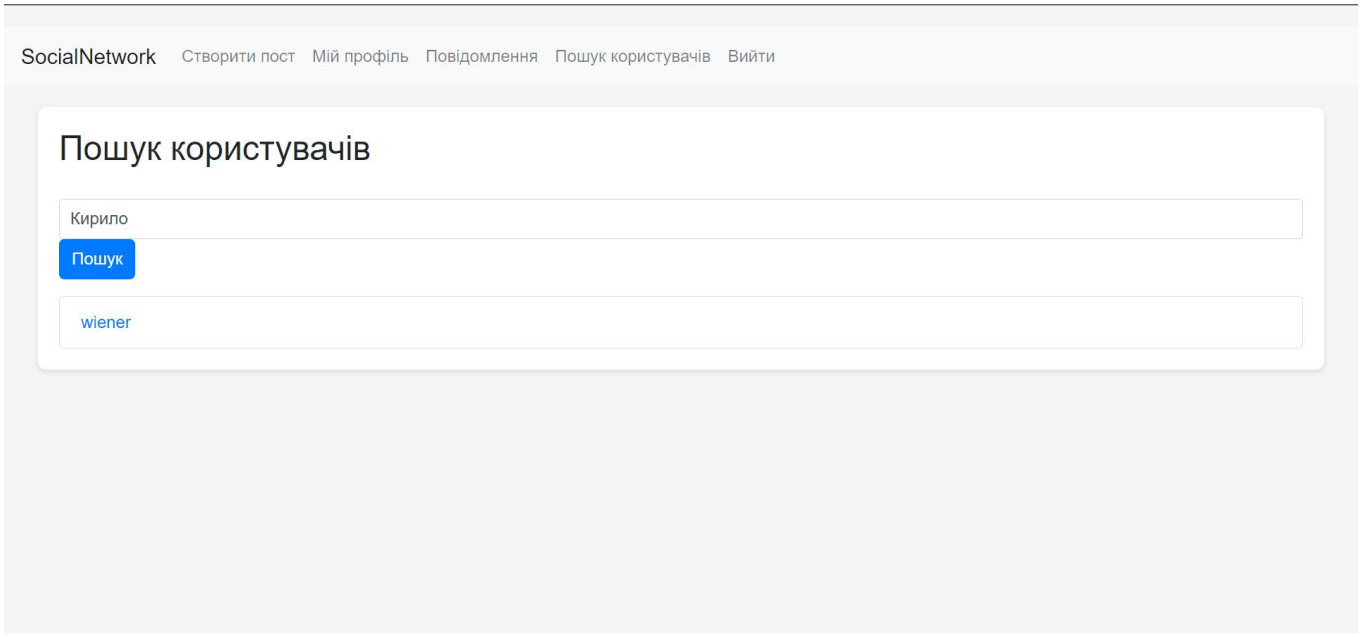


Рисунок 4.9 – Пошук користувачів

Ця інструкція надає користувачам необхідні знання для встановлення, налаштування та використання соціальної мережі для студентів. Дотримуючись вказаних кроків та рекомендацій, користувачі можуть ефективно взаємодіяти один з одним, обмінюватися інформацією та співпрацювати над спільними проектами.

ВИСНОВКИ

Під час виконання дипломної роботи був детально розглянутий теоретичний матеріал з теми «Програмна реалізація соціальної мережі для студентів з використанням мови програмування Python». Було проведено огляд сучасних методів та інструментів розробки веб-додатків, а також розроблено веб-додаток соціальної мережі, що включає широкий спектр функцій для забезпечення ефективної комунікації та взаємодії між студентами.

Результати роботи:

1. Проведено інформаційний огляд літературних джерел та робіт з розробки веб-додатків на платформі Flask та інших аналогічних платформах.
2. Виділено позитивні та негативні сторони різних методів розробки веб-додатків та соціальних мереж.
3. Обґрунтовано вибір технологій для розробки веб-додатку, включаючи мову програмування Python, фреймворк Flask, базу даних SQLite та різноманітні інструменти для фронтенд-розробки.
4. Розроблено веб-додаток соціальної мережі, який включає функції реєстрації та автентифікації користувачів, створення та редагування постів, коментування, ставлення лайків, обмін повідомленнями, редагування профілю та завантаження медіафайлів.
5. Реалізовано та оптимізовано модулі для керування постами та повідомленнями, що забезпечують ефективне зберігання та обробку даних у базі даних.
6. Створено інтерфейс користувача, який забезпечує зручний спосіб взаємодії з додатком, включаючи можливість додавання, редагування, видалення постів та перегляду профілів інших користувачів.
7. Проведено відладку та виправлення помилок у веб-додатку, використовуючи інструменти налагодження та тестування.
8. Проведено тестування веб-додатку для перевірки коректності його роботи, зручності використання та стійкості проти потенційних загроз.

9. Розроблено інструктаж для користувачів щодо встановлення, налаштування та використання веб-додатку соціальної мережі.

Робота дозволила глибше зрозуміти принципи розробки веб-додатків на базі Flask та забезпечення безпеки даних у веб-додатках. Було розроблено практичний інструмент, який може бути використаний для підвищення ефективності комунікації між студентами, а також для навчальних та дослідницьких цілей у сфері розробки веб-додатків.

СПИСОК ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Miguel Grinberg. Flask Documentation. Доступно онлайн за адресою: https://flask.palletsprojects.com/en/2.0.x/
2. Krishna Rungta. Learn Flask Framework. Guru99, 2022. Доступно онлайн за адресою: https://www.guru99.com/flask-tutorial.html
3. Miguel Grinberg. Flask Web Development: Developing Web Applications with Python. O'Reilly Media, 2018. 258 с.
4. Michael Herman. Mastering Flask Web Development. Packt Publishing, 2018. 400 с.
5. Daniel Feldroy, Audrey Feldroy. Two Scoops of Django 3.x: Best Practices for the Django Web Framework. Feldroy, 2020. 725 с.
6. Eric Matthes. Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to Programming. No Starch Press, 2019. 544 с.
7. William S. Vincent. Python Web Development with Flask. 2018. 200 с.
8. Julien Danjou. The Hacker's Guide to Python. 2014. 264 с.
9. Luciano Ramalho. Fluent Python: Clear, Concise, and Effective Programming. O'Reilly Media, 2015. 792 с.
10. Jake VanderPlas. Python Data Science Handbook: Essential Tools for Working with Data. O'Reilly Media, 2016. 548 с.
11. J. Burton Browning, Marty Alchin. Pro Python 3: Features and Tools for Professional Development. Apress, 2019. 468 с.
12. Brad Miller, David Ranum. Problem Solving with Algorithms and Data Structures Using Python. Franklin, Beedle & Associates Inc., 2011. 428 с.
13. Tony Gaddis. Starting Out with Python. Pearson, 2018. 744 с.
14. Mark Lutz. Learning Python, 5th Edition. O'Reilly Media, 2013. 1648 с.
15. Ольховська О. В. Методичні рекомендації до виконання кваліфікаційної роботи для студентів спеціальності 122 Комп’ютерні науки освітня програма «Комп’ютерні науки» ступеня бакалавра / О. В. Ольховська, О. О. Черненко. – Полтава : ПУЕТ, 2022. – 67 с. – 1 електрон. опт. диск (CVD-ROM).

ДОДАТОК А.

config.py

```
import os

class Config:
    SECRET_KEY = os.environ.get('SECRET_KEY', 'secret')
    SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL', 'sqlite:///site.db')
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    ENV = os.environ.get('FLASK_ENV', 'production')  # Задаємо значення за замовчуванням
    UPLOAD_FOLDER = 'static/img'

class DevelopmentConfig(Config):
    DEBUG = True
    ENV = 'development'  # Явно вказуємо, що це розробницьке середовище

class TestingConfig(Config):
    TESTING = True
    SQLALCHEMY_DATABASE_URI = 'sqlite:///test.db'
    ENV = 'testing'

class ProductionConfig(Config):
    DEBUG = False
    ENV = 'production'
```

models.py

```
from datetime import datetime
from flask_login import UserMixin
from database import db
from flask_wtf import FlaskForm
from wtforms import BooleanField, PasswordField, StringField, SubmitField, TextAreaField, FileField
from wtforms.validators import DataRequired, Length, EqualTo, Optional, Regexp
from flask_wtf.file import FileField, FileAllowed

class User(UserMixin, db.Model):
    __tablename__ = 'user'
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(100), unique=True, nullable=False)
    password = db.Column(db.String(200), nullable=False)
    first_name = db.Column(db.String(100), nullable=False)
    last_name = db.Column(db.String(100), nullable=False)
    about_me = db.Column(db.String(250))
    group_name = db.Column(db.String(50), nullable=True)  # Нове поле для назви групи
    group_number = db.Column(db.String(50), nullable=True)  # Нове поле для номера групи
    avatar = db.Column(db.String(120), nullable=True)
    messages_sent = db.relationship('Message',
                                    foreign_keys='Message.sender_id',
                                    backref='author', lazy='dynamic')
    messages_received = db.relationship('Message',
                                        foreign_keys='Message.recipient_id',
```

```
                                backref='recipient', lazy='dynamic')
    posts = db.relationship('Post', backref='author', lazy='dynamic')

    # Відносини будуть оновлені після визначення класу Post

class Post(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    body = db.Column(db.String(140))
    timestamp = db.Column(db.DateTime, index=True, default=datetime.utcnow)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))
    comments = db.relationship('Comment', backref='post', lazy='dynamic')
    media = db.Column(db.String(255))

# Тепер можна безпечно визначити асоціативну таблицю
likes = db.Table('likes',
    db.Column('user_id', db.Integer, db.ForeignKey('user.id'), primary_key=True),
    db.Column('post_id', db.Integer, db.ForeignKey('post.id'), primary_key=True),
    db.Column('timestamp', db.DateTime, index=True, default=datetime.utcnow)
)

# Оновлення відносин у User
User.liked_posts = db.relationship('Post', secondary=likes,
                        primaryjoin=(likes.c.user_id == User.id),
                        secondaryjoin=(likes.c.post_id == Post.id),
                        backref=db.backref('liked_by', lazy='dynamic'),
                        lazy='dynamic')

User.like_post = lambda self, post: self.liked_posts.append(post) if not self.has_liked_post(post) else None
User.unlike_post = lambda self, post: self.liked_posts.remove(post) if self.has_liked_post(post) else None
User.has_liked_post = lambda self, post: db.session.query(likes).filter_by(user_id=self.id,
post_id=post.id).count() > 0

class Comment(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    body = db.Column(db.String(140))
    timestamp = db.Column(db.DateTime, index=True, default=datetime.utcnow)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))
    post_id = db.Column(db.Integer, db.ForeignKey('post.id'))
    user = db.relationship('User', backref='user_comments')  # Змінено на 'user_comments'

    def __repr__(self):
        return f'<Comment {self.body}>'

class Message(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    sender_id = db.Column(db.Integer, db.ForeignKey('user.id'))
    recipient_id = db.Column(db.Integer, db.ForeignKey('user.id'))
    body = db.Column(db.String(140))
    timestamp = db.Column(db.DateTime, index=True, default=datetime.utcnow)
```

```
    def __repr__(self):
        return '<Message {}>'.format(self.body)

class PostForm(FlaskForm):
    body = StringField('Say something', validators=[DataRequired()])
    media = FileField('Attach a photo or video', validators=[FileAllowed(['jpg', 'jpeg', 'png', 'gif', 'mp4'],
'Only images and videos are allowed!')])
    submit = SubmitField('Submit')

class EditPostForm(FlaskForm):
    body = TextAreaField('Текст', validators=[DataRequired()])
    submit = SubmitField('Оновити')

class CommentForm(FlaskForm):
    body = StringField('Коментар', validators=[DataRequired()])
    submit = SubmitField('Надіслати')

class EditProfileForm(FlaskForm):
    avatar = FileField('Змінити аватар', validators=[FileAllowed(['jpg', 'png'], 'Тільки зображення!')])
    submit = SubmitField('Зберегти зміни')
    first_name = StringField('Ім’я', validators=[DataRequired()])
    last_name = StringField('Прізвище', validators=[DataRequired()])
    username = StringField('Логін', validators=[DataRequired()])
    about_me = TextAreaField('Про мене', validators=[Length(min=0, max=140)])
    group_name = StringField('Назва групи', validators=[Optional()])
    group_number = StringField('Номер групи', validators=[Optional()])
    change_password = BooleanField('Змінити пароль')
    old_password = PasswordField('Старий пароль', validators=[Optional()])
    new_password = PasswordField('Новий пароль', validators=[Optional()])
    confirm = PasswordField('Підтвердіть новий пароль', validators=[
        EqualTo('new_password', message='Паролі мають співпадати')])
    submit = SubmitField('Зберегти зміни')

class MessageForm(FlaskForm):
    message = TextAreaField('Повідомлення', validators=[DataRequired(), Length(min=0, max=140)])
    submit = SubmitField('Відправити')

class SearchForm(FlaskForm):
    query = StringField('Шукати', validators=[DataRequired()])
    submit = SubmitField('Пошук')

class RegistrationForm(FlaskForm):
    username = StringField('Username', validators=[
        DataRequired(),
        Length(min=3, max=20),
        Regexp(r'^[A-Za-z0-9]+$', message="Логін повинен містити тільки англійські букви та цифри.")
    ])
    first_name = StringField('First Name', validators=[
        DataRequired(),
        Length(min=1, max=50),
```

```
        Regexp(r'^[\u0400-\u04FF]+$', message="Ім'я повинно містити тільки кирилицю.")
    ])
    last_name = StringField('Last Name', validators=[
        DataRequired(),
        Length(min=1, max=50),
        Regexp(r'^[\u0400-\u04FF]+$', message="Прізвище повинно містити тільки кирилицю.")
    ])
    password = PasswordField('Password', validators=[
        DataRequired(),
        Length(min=8, message="Пароль повинен бути мінімум 8 символів."),
            Regexp(r'^(?=.*[A-Za-z])(?=.*\d)[A-Za-z\d]{8,}$', message="Пароль повинен містити букви та цифри.")
    ])
    confirm_password = PasswordField('Confirm Password', validators=[
        DataRequired(),
        EqualTo('password', message='Паролі повинні співпадати.')
    ])
    submit = SubmitField('Register')

class LoginForm(FlaskForm):
    username = StringField('Username', validators=[
        DataRequired(),
        Regexp(r'^[A-Za-z0-9]+$', message="Логін повинен містити тільки англійські букви та цифри.")
    ])
    password = PasswordField('Password', validators=[DataRequired()])
    submit = SubmitField('Log In')
```

routes.py

```
from datetime import datetime
from os import abort
import os
from flask import render_template, request, redirect, url_for, flash
from werkzeug.security import generate_password_hash, check_password_hash
from flask_login import login_user, logout_user, login_required, current_user
from models import EditPostForm, EditProfileForm, LoginForm, PostForm, User, Post, Comment, CommentForm, Message, MessageForm, SearchForm, RegistrationForm
from flask import current_app as app
from database import db
from werkzeug.utils import secure_filename

def init_routes(app):
    @app.route('/register', methods=['GET', 'POST'])
    def register():
        if current_user.is_authenticated:
            return redirect(url_for('index'))

        form = RegistrationForm()
        if form.validate_on_submit():
            first_name = form.first_name.data
            last_name = form.last_name.data
```

```
            username = form.username.data
            password = form.password.data

            existing_user = User.query.filter_by(username=username).first()
            if existing_user:
                flash('Цей логін вже зайнятий. Спробуйте інший.', 'warning')
                return redirect(url_for('register'))

            hashed_password = generate_password_hash(password)
                    new_user = User(first_name=first_name, last_name=last_name, username=username,
password=hashed_password)
            db.session.add(new_user)
            db.session.commit()

            flash('Реєстрація успішна. Тепер ви можете увійти.', 'success')
            return redirect(url_for('login'))

        return render_template('register.html', form=form)

    @app.route('/login', methods=['GET', 'POST'])
    def login():
        if current_user.is_authenticated:
            return redirect(url_for('index'))

        form = LoginForm()
        if form.validate_on_submit():
            username = form.username.data
            password = form.password.data

            user = User.query.filter_by(username=username).first()

            if user and check_password_hash(user.password, password):
                login_user(user, remember=True)  # Optionally handle 'remember me' functionality
                next_page = request.args.get('next')
                return redirect(next_page) if next_page else redirect(url_for('index'))
            else:
                flash('Неправильний логін або пароль.', 'danger')  # Adding bootstrap class for styling

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

    @app.route('/logout')
    @login_required
    def logout():
        logout_user()
        return redirect(url_for('login'))

    @app.route('/')
    @app.route('/index')
    @login_required
    def index():
```

```
    posts = Post.query.order_by(Post.timestamp.desc()).all()
    return render_template('index.html', posts=posts)

@app.route('/news')
@login_required
def news():
    posts = Post.query.order_by(Post.created_at.desc()).all()
    return render_template('news.html', posts=posts)

@app.route('/create_post', methods=['GET', 'POST'])
@login_required
def create_post():
    form = PostForm()
    if form.validate_on_submit():
        post = Post(body=form.body.data, author=current_user)
        if form.media.data:
            media_file = form.media.data
            filename = secure_filename(media_file.filename)
            filepath = os.path.join(app.config['UPLOAD_FOLDER'], filename)
            media_file.save(filepath)
            post.media = filename  # Зберігаємо лише ім'я файлу
        db.session.add(post)
        db.session.commit()
        flash('Ваш пост було успішно опубліковано!')
        return redirect(url_for('index'))
    return render_template('create_post.html', form=form)

@app.route('/edit_post/<int:post_id>', methods=['GET', 'POST'])
@login_required
def edit_post(post_id):
    post = Post.query.get_or_404(post_id)
    if post.author != current_user:
        flash('You do not have permission to edit this post.')
        return redirect(url_for('index'))

    form = EditPostForm()
    if form.validate_on_submit():
        post.body = form.body.data
        db.session.commit()
        flash('Your post has been updated.')
        return redirect(url_for('post', post_id=post.id))

    form.body.data = post.body
    return render_template('edit_post.html', title='Редагування Поста', form=form, post=post)

@app.route('/delete_post/<int:post_id>', methods=['POST'])
@login_required
def delete_post(post_id):
    post = Post.query.get_or_404(post_id)
    if post.author != current_user:
```

```
        abort(403)  # Забороняємо видалення, якщо користувач не є автором посту
    db.session.delete(post)
    db.session.commit()
    flash('Ваш пост було успішно видалено!', 'success')
    return redirect(url_for('index'))

@app.route('/post/<int:post_id>', methods=['GET', 'POST'])
@login_required
def post(post_id):
    post = Post.query.get_or_404(post_id)
    form = CommentForm()
    if form.validate_on_submit():
        comment = Comment(body=form.body.data, user_id=current_user.id, post_id=post.id)
        db.session.add(comment)
        db.session.commit()
        flash('Ваш коментар було додано.')
        return redirect(url_for('post', post_id=post.id))
    comments = post.comments.order_by(Comment.timestamp.asc())
    return render_template('post.html', post=post, form=form, comments=comments)

@app.route('/like/<int:post_id>', methods=['POST'])
@login_required
def like(post_id):
    post = Post.query.get_or_404(post_id)
    current_user.like_post(post)
    db.session.commit()
    return redirect(request.referrer)

@app.route('/unlike/<int:post_id>', methods=['POST'])
@login_required
def unlike(post_id):
    post = Post.query.get_or_404(post_id)
    current_user.unlike_post(post)
    db.session.commit()
    return redirect(request.referrer)

@app.route('/user/<username>')
@login_required
def user_profile(username):
    user = User.query.filter_by(username=username).first_or_404()
    posts = Post.query.filter_by(user_id=user.id).order_by(Post.timestamp.desc()).all()
    return render_template('user_profile.html', user=user, posts=posts)

@app.route('/edit_profile', methods=['GET', 'POST'])
@login_required
def edit_profile():
    form = EditProfileForm(obj=current_user)
    if form.validate_on_submit():
        current_user.username = form.username.data
        current_user.first_name = form.first_name.data
```

```
        current_user.last_name = form.last_name.data
        current_user.group_name = form.group_name.data
        current_user.group_number = form.group_number.data
        current_user.about_me = form.about_me.data

        # Обробка завантаження аватару
        if form.avatar.data:
            avatar_file = form.avatar.data
            if hasattr(avatar_file, 'filename'):
                filename = secure_filename(avatar_file.filename)
                save_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
                # Створіть директорію, якщо вона не існує
                if not os.path.exists(app.config['UPLOAD_FOLDER']):
                    os.makedirs(app.config['UPLOAD_FOLDER'])
                avatar_file.save(save_path)
                current_user.avatar = filename

        db.session.commit()
        flash('Зміни профілю було збережено.', 'success')
        return redirect(url_for('user_profile', username=current_user.username))

    return render_template('edit_profile.html', form=form)

@app.route('/messages')
@login_required
def messages():
    # Отримання діалогів з останніми повідомленнями
    subquery = db.session.query(
        Message.sender_id, Message.recipient_id, db.func.max(Message.timestamp).label('last_msg_time')
    ).filter(
        (Message.sender_id == current_user.id) | (Message.recipient_id == current_user.id)
    ).group_by(
        Message.sender_id, Message.recipient_id
    ).subquery()

    last_messages = db.session.query(Message).join(
        subquery, db.or_(
            db.and_(
                Message.sender_id == subquery.c.sender_id,
                Message.recipient_id == subquery.c.recipient_id,
                Message.timestamp == subquery.c.last_msg_time
            ),
            db.and_(
                Message.sender_id == subquery.c.recipient_id,
                Message.recipient_id == subquery.c.sender_id,
                Message.timestamp == subquery.c.last_msg_time
            )
        )
    ).order_by(Message.timestamp.desc())
```

```
    return render_template('messages.html', messages=last_messages)

@app.route('/send_message/<username>', methods=['GET', 'POST'])
@login_required
def send_message(username):
    user = User.query.filter_by(username=username).first_or_404()
    form = MessageForm()
    if form.validate_on_submit():
        message = Message(sender_id=current_user.id, recipient_id=user.id, body=form.message.data)
        db.session.add(message)
        db.session.commit()
        flash('Your message has been sent.')
        return redirect(url_for('send_message', username=username))

    messages = Message.query.filter(
        db.or_(
            db.and_(Message.sender_id == current_user.id, Message.recipient_id == user.id),
            db.and_(Message.sender_id == user.id, Message.recipient_id == current_user.id)
        )
    ).order_by(Message.timestamp.asc())
    return render_template('send_message.html', form=form, recipient=user, messages=messages)

@app.route('/search', methods=['GET', 'POST'])
@login_required
def search():
    form = SearchForm()
    users = []
    if form.validate_on_submit():
        query = form.query.data
        # Змінено фільтрацію для пошуку за іменем або прізвищем
        users = User.query.filter(
            db.or_(
                User.first_name.ilike(f'%{query}%'),
                User.last_name.ilike(f'%{query}%')
            )
        ).all()
        if not users:
            flash('Немає користувачів з таким іменем або прізвищем.', 'info')
    return render_template('search.html', form=form, users=users)
```

app.py

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager
from flask_migrate import Migrate
from config import DevelopmentConfig

# Імпорт бази даних з зовнішнього файлу
from database import db
```

```
login_manager = LoginManager()

def create_app(config_class=DevelopmentConfig):
    app = Flask(__name__)
    app.config.from_object(config_class)

    # Ініціалізація розширень Flask з поточним додатком
    db.init_app(app)
    login_manager.init_app(app)
    migrate = Migrate(app, db)

    # Налаштування сторінки для неавторизованих користувачів
    login_manager.login_view = 'login'  # Вказуємо ім'я функції маршруту 'login'

    # Функція для завантаження даних користувача
    @login_manager.user_loader
    def load_user(user_id):
        from models import User
            return db.session.query(User).get(int(user_id))  # Використовуємо метод get для отримання користувача

    # Реєстрація маршрутів і моделей
    with app.app_context():
        from models import User
        from routes import init_routes
        init_routes(app)

        # Створення таблиць у базі даних, якщо вони ще не створені
        if db.engine.url.drivername == 'sqlite':
            db.create_all()

    return app

app = create_app()

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

database.py

```
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()
```