МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

МЕЛЬНИК ДАР'Я СЕРГІЇВНА

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
канд. техн. наук, доцент
_____ О. В. Зелінська
«____»_____ 20__ р.

РОЗРОБКА ВЕБДОДАТКУ ДЛЯ БЛОГ-ПЛАТФОРМИ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
П. К. Ніколюк, професор кафедри
інформаційних технологій,
д-р фіз.-мат. наук, професор
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____

Вінниця – 2025

АНОТАЦІЯ

Мельник Д.С «Розробка вебдодатку для блог платформи». Спеціальність 122 «Комп’ютерні науки», освітня програма «Сучасні інформаційні технології та програмування». Донецький національний університет імені Василя Стуса, Вінниця 2025.

У кваліфікаційній (бакалаврській) роботі досліджено та проаналізовано методи створення вебдодатку. За допомогою таких технологій, як мова програмування Python, фреймворк Flask, технології для фронтенду (HTML, CSS, Jinja2), система управління базами даних SQLite, а також для забезпечення безпеки використовуються сторонні бібліотеки, як-от Werkzeug для хешування паролів

Ключові слова: вебдодаток, блог, блог-платформа, інтерактивність,динамічність.

53 ст., 21 рис., 2 дод., 20 джерел.

ABSTRACT

Melnyk D.S. " Development of a Web Application for a Blog Platform ". Specialty 122 "Computer science", educational program "Modern information technologies and programming". Vasyl Stus Donetsk National University, Vinnytsia 2025.

In the qualification (bachelor's) thesis, the methods of recognizing military objects, as well as tools for designing and training similar systems, were researched and analyzed. With the help of such technologies as the Python programming language, the Flask framework is used, along with frontend technologies such as HTML, CSS, and Jinja2, the SQLite database management system, and third-party libraries such as Werkzeug are used to ensure security, including password hashing.

Keywords: Web Application, blog, blog platform, interactivity, dynamic nature.

ЗМІСТ

ВСТУП

Актуальність теми дослідження. Із розвитком інтернет-технологій блогові веб-додатки стали важливою складовою цифрового простору, змінивши спосіб, у який користувачі споживають та створюють інформацію. Сьогодні блоги виконують не лише функцію особистих щоденників, а й перетворилися на потужні інструменти впливу, маркетингу, навчання, громадського висловлювання та взаємодії. Як засіб масової комунікації, блогові платформи активно використовуються як приватними особами, так і компаніями, медіа, освітніми установами й громадськими організаціями. Їхнє поширення засвідчує зростаючу потребу в динамічних і доступних онлайн-середовищах для публікації, обміну і обговорення контенту. У такому контексті актуальність теми розробки сучасного блогового веб-додатка не викликає сумнівів.

Блогові веб-додатки мають бути не лише зручними для кінцевих користувачів, а й достатньо гнучкими для підтримки нових функцій, адаптивними до змінних вимог, безпечними та стабільними в роботі. Розробка такого додатка потребує ґрунтовного підходу до проектування архітектури, вибору технологій, реалізації інтерфейсу та забезпечення якості функціональності. З огляду на це, створення повноцінного веб-додатка для ведення блогу є не лише технічним завданням, а й складним інженерним процесом, що вимагає врахування великої кількості взаємопов’язаних аспектів.

Мета роботи полягає в розробці сучасного, функціонального та динамічного блогового веб-додатка, який дозволятиме користувачам створювати, редагувати, коментувати та поширювати контент, забезпечуючи при цьому інтуїтивно зрозумілий інтерфейс і високий рівень взаємодії. Додаток повинен відповідати вимогам безпеки, мати адаптивний дизайн для різних пристроїв, підтримувати організацію контенту за допомогою категорій і тегів, а також реалізовувати механізми взаємодії між користувачами — такі як коментарі, оцінювання та підписки.

Щоб досягти поставленої мети — створити сучасний, зручний і функціональний вебдодаток для блог-платформи — в роботі було заплановано вирішити низку практичних завдань. У процесі роботи було поставлено такі **завдання**:

- з’ясувати, якими функціями повинна володіти сучасна блог-платформа та що важливо для користувачів;
- вивчити сучасні фреймворки й бібліотеки для створення вебдодатків, порівняти їх і вибрати оптимальні для проєкту;
- розібратися з тим, як краще спроєктувати інтерфейс, щоб ним було легко й приємно користуватись;
- продумати, як реалізувати створення, редагування, коментування та поширення постів;
- побудувати структуру бази даних, яка дозволить зручно працювати з постами, тегами, категоріями, користувачами та їх взаємодіями;
- реалізувати вебдодаток із сучасним адаптивним дизайном, авторизацією та захистом даних.

Об’єктом дослідження виступають веб-додатки, призначені для створення, обробки та керування цифровим контентом.

Предметом дослідження є процес розробки динамічного блогового веб-додатка з використанням сучасних інструментів і технологій, як мова програмування Python, фреймворк Flask, технології для фронтенду (HTML, CSS, Jinja2), система управління базами даних SQLite, а також для забезпечення безпеки використовуються сторонні бібліотеки, як-от Werkzeug для хешування паролів

Створення динамічного вебдодатку для блог-платформи має цінність як у теоретичному, так і в практичному контексті. З теоретичного боку, робота дозволяє краще усвідомити особливості побудови багатофункціональних вебсервісів, розглянути підходи до реалізації користувацької взаємодії, структуризації контенту та безпечної обробки даних.

Практична цінність. Практичне значення полягає у створенні повноцінного інструменту, що може бути застосований для публікації авторського контенту, обміну думками через коментарі, а також організації взаємодії між користувачами. Розроблений додаток може використовуватись як основа для подальших комерційних або навчальних проєктів у сфері веброзробки.

Структура кваліфікаційної роботи:

Робота складається із вступу, трьох розділів, висновку, використаних джерел із 20 пунктів, включаючи офіційну документацію обраних технологій, довідники та посібники, 15 рисунків. Обсяг роботи - 81 сторінка.

РОЗДІЛ 1
АНАЛІЗ СУЧАСНОГО СТАНУ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Види інформаційних об'єктів та огляд популярних платформ вебдодатків для ведення блогу

Інформаційні об'єкти є фундаментальними складовими будь-якого сучасного веб-додатка, адже саме вони визначають, які дані зберігаються, як вони обробляються та взаємодіють між собою. У контексті блогових платформ інформаційні об’єкти не лише відображають технічну структуру системи, а й значною мірою формують користувацький досвід, визначаючи, як саме буде виглядати і функціонувати інтерфейс, які можливості будуть доступні користувачеві, та наскільки зручною і логічною буде навігація.

Одним із ключових інформаційних об'єктів у блогах є користувач. Він є ініціатором та споживачем контенту, і саме навколо його дій вибудовується логіка додатка. Об’єкт "користувач" зазвичай включає ідентифікаційні дані (логін, email, пароль у зашифрованому вигляді), а також додаткову інформацію — ім’я, аватар, дату реєстрації, біографію, соціальні зв’язки. У складніших системах також можуть зберігатися рольові моделі, які визначають рівень доступу: звичайний користувач, редактор, адміністратор. Це дозволяє більш гнучко керувати правами на створення, редагування чи видалення контенту.

Ще одним центральним об’єктом є публікація або пост, що містить основний контент блогу. Структура поста може бути досить складною, включаючи заголовок, текст, зображення, дату створення, дату оновлення, ім’я автора, категорію, набір тегів, кількість переглядів, статус (опубліковано, в чернетках, видалено) тощо. У деяких випадках пости можуть мати прикріплені медіафайли, документи або навіть елементи інтерактивного контенту (наприклад, опитування або відеовставки). Важливо, що кожен пост існує не сам по собі, а у взаємозв’язку з іншими об’єктами, зокрема з користувачами, категоріями та коментарями.

Коментарі — ще один важливий тип інформаційного об’єкта, який забезпечує динаміку на платформі та сприяє створенню спільноти навколо контенту. Кожен коментар зазвичай включає текст повідомлення, дату і час його публікації, автора коментаря, а також зв’язок з конкретним постом. В деяких реалізаціях також підтримуються вкладені коментарі (відповіді), що формує дерево обговорень. Це дозволяє користувачам вести дискусії безпосередньо під конкретними думками, зберігаючи контекст спілкування.

Не менш важливими для організації контенту є об’єкти категорій і тегів. Вони виконують роль класифікаторів, що допомагають структурувати інформацію на платформі. Категорії, як правило, є ієрархічними — тобто можуть містити підкатегорії, що дозволяє побудувати глибоку структуру тем. Теги, навпаки, використовуються більш вільно — користувач може додати до поста будь-яку кількість тегів, які відображають ключові поняття або теми, згадані у тексті. Це значно спрощує пошук і навігацію, дозволяючи користувачам швидше знаходити релевантний контент.

Окрему категорію становлять медіаоб'єкти — зображення, відео, аудіофайли, документи, які додаються до постів або коментарів. Вони можуть зберігатися безпосередньо в базі даних або на зовнішньому файловому сховищі, але у будь-якому випадку потребують метаданих: назва файлу, тип, розмір, дата завантаження, посилання на автора або публікацію. Робота з такими об'єктами також потребує особливої уваги до безпеки та оптимізації — наприклад, створення мініатюр зображень або автоматичне стиснення великих файлів.

У розвиненіших системах також можуть бути передбачені такі об’єкти, як повідомлення (notifications), приватні листування між користувачами, збережені чернетки, історія редагувань постів, системні логи, а також налаштування профілю чи інтерфейсу. Кожен із них доповнює загальну функціональність платформи та робить її більш персоналізованою і зручною для користувачів.

Важливим аспектом у проектуванні таких об'єктів є їхня взаємодія. Наприклад, між постами та користувачами існує зв’язок "один до багатьох", оскільки один користувач може мати кілька публікацій, а кожен коментар завжди

належить до конкретного поста. У складніших сценаріях зв’язки можуть бути багато до багатьох — як у випадку з тегами, які можуть бути пов’язані з численними постами одночасно. Грамотне моделювання цих зв’язків є запорукою ефективної роботи додатка, забезпечуючи як швидкий доступ до даних, так і їх узгодженість та цілісність.

Таким чином, інформаційні об'єкти є не просто одиницями даних, а цілісними сутностями, навколо яких вибудовується вся логіка додатка. Їх правильне проектування дозволяє створити масштабовану, адаптивну та зручну у використанні платформу, яка може ефективно обробляти велику кількість даних та забезпечувати якісний досвід для користувачів. Успішна реалізація блогового веб-додатка значною мірою залежить саме від того, наскільки точно й продумано були змодельовані ці об’єкти на початковому етапі розробки.

Ведення блогів стало невід'ємною частиною сучасного інтернету. Це дозволяє людям ділитися своїми думками, досвідом та знаннями з глобальною аудиторією. Існує багато платформ та веб-додатків, що дозволяють створювати та керувати блогами, кожна з яких має свої унікальні особливості, переваги та недоліки.

Twitter – це соціальна мережа, що дозволяє користувачам публікувати короткі повідомлення (твіти) довжиною до 280 символів. Вона відома своєю швидкістю поширення інформації та широкою аудиторією. Основні функції: публікація коротких текстових повідомлень, додавання фотографій, відео та посилань, функції ретвітів, лайків та коментарів, хештеги для організації контенту. Переваги: швидке поширення інформації, велика аудиторія та висока взаємодія, можливість використання хештегів для збільшення видимості. Недоліки: обмеження на довжину повідомлень, складність у веденні довготривалих дискусій, висока конкуренція за увагу користувачів.

Instagram – це соціальна мережа, орієнтована на публікацію фотографій та відео. Вона є ідеальною платформою для візуальних блогів, де основний акцент робиться на зображеннях та відеоконтенті. Основні функції: публікація фотографій та відео, історії (Stories), що зникають через 24 години, IGTV для

довгих відео, можливість додавання тексту, хештегів та геоміток до публікацій. Переваги: візуальний контент привертає більше уваги, висока взаємодія через лайки, коментарі та підписки, можливість монетизації через рекламу та співпрацю з брендами. Недоліки: високі вимоги до якості контенту, менші можливості для довгих текстових постів, алгоритми можуть обмежувати видимість контенту.

Facebook – це одна з найбільших соціальних мереж у світі, що дозволяє користувачам публікувати текстові повідомлення, фотографії, відео, створювати події та групи. Основні функції: публікація текстових постів, фотографій та відео, створення сторінок та груп для спільнот за інтересами, можливість організації та реклами подій, інтеграція з іншими сервісами та додатками. Переваги: велика аудиторія користувачів, широкі можливості для взаємодії з підписниками, можливість таргетованої реклами. Недоліки: зниження органічного охоплення постів через алгоритми, висока конкуренція за увагу користувачів, складність у захисті приватності та даних користувачів.

WordPress – це одна з найпопулярніших платформ для створення блогів і сайтів різного рівня складності. Вона є системою управління контентом (CMS), яка надає широкі можливості для кастомізації. Основні функції: створення та редагування постів, використання шаблонів дизайну, підтримка плагінів для розширення функціональності, налаштування SEO, підтримка мультимедійного контенту. Переваги: повний контроль над контентом і дизайном, велика кількість безкоштовних і платних тем і плагінів, активна спільнота розробників. Недоліки: потреба в базових технічних знаннях для налаштування, складність у підтримці безпеки, особливо на самостійно хостингових версіях (WordPress.org).

Blogger – це безкоштовна платформа для ведення блогів, яка належить компанії Google. Вона дозволяє швидко створити блог без необхідності володіння технічними навичками. Основні функції: створення публікацій, редагування шаблонів, інтеграція з Google Analytics, можливість монетизації через Google AdSense. Переваги: проста у використанні, повна інтеграція з сервісами Google, безкоштовний хостинг. Недоліки: обмежені можливості для

кастомізації, менша кількість шаблонів та плагінів у порівнянні з WordPress, обмежений контроль над технічною частиною.

Medium – це платформа, що поєднує в собі блогінг та соціальні функції, орієнтована на якісний текстовий контент. Вона часто використовується журналістами, письменниками та експертами в різних галузях. Основні функції: простий редактор для публікації тексту, можливість створення колекцій статей, інтеграція з соціальними мережами, підписка на авторів. Переваги: фокус на контент, мінімалістичний інтерфейс, вбудована аудиторія платформи, можливість заробітку через партнерську програму Medium. Недоліки: обмежена кастомізація дизайну, відсутність власного домену у безкоштовному варіанті, залежність від політики платформи.

Tumblr – це мікроблогінгова платформа, що поєднує можливості ведення блогів і соціальних мереж. Вона популярна серед молоді та креативних користувачів. Основні функції: публікація тексту, зображень, відео, гіфок, аудіо та посилань, можливість стежити за іншими блогами, система тегів для навігації. Переваги: простота використання, активна спільнота, широкий спектр підтримуваного контенту. Недоліки: обмежені можливості кастомізації, менш професійний вигляд у порівнянні з іншими платформами, зменшення активності користувачів останніми роками.

Ghost – це сучасна платформа з відкритим кодом, створена спеціально для професійного блогінгу та онлайн-публікацій. Вона орієнтована на швидкість, мінімалізм і зручність для авторів. Основні функції: сучасний редактор для написання контенту, підтримка підписок і платного доступу до матеріалів, SEO-інструменти, інтеграція з зовнішніми сервісами. Переваги: висока швидкість, чистий інтерфейс, можливість монетизації контенту, підтримка Markdown. Недоліки: складніша установка у порівнянні з Blogger чи Medium, обмежена кількість шаблонів і плагінів, переважно платне хостингування.

Загалом, вибір платформи для ведення блогу залежить від потреб користувача, технічного досвіду, типу контенту та бажаного рівня контролю над виглядом і функціональністю. У рамках розробки власного веб-додатка важливо

враховувати сильні сторони кожної з перелічених платформ, щоб створити конкурентоспроможне, зручне й функціональне рішення для користувачів.

1.2 Аналіз технічних вимог та потреб користувачів до сучасних блог-платформ

Аналіз очікувань користувачів є одним із ключових етапів при створенні ефективного та конкурентоспроможного веб-додатка. В умовах стрімкого розвитку цифрових технологій користувачі стають дедалі більш вимогливими, очікуючи від онлайн-сервісів не лише функціональності, а й зручності, швидкості та інтуїтивно зрозумілого інтерфейсу. Важливо розуміти, що сучасний користувач має доступ до великої кількості альтернатив, тому саме якість взаємодії з платформою часто стає вирішальним фактором у виборі певного сервісу. У цьому контексті очікування користувачів охоплюють не тільки технічні аспекти, а й емоційні – відчуття задоволення від використання, довіру до ресурсу та готовність рекомендувати його іншим.

Ще одним важливим аспектом є персоналізація досвіду, адже користувачі прагнуть відчувати, що платформа враховує їхні індивідуальні потреби, вподобання та стиль взаємодії. Це може виявлятися у вигляді рекомендацій на основі попередніх дій, налаштування тем оформлення чи можливості адаптувати інтерфейс під власні звички. Крім того, значну роль відіграє безпека та захист персональних даних – користувачі очікують, що їх інформація буде зберігатися конфіденційно та не використовуватиметься без дозволу.

Для того щоб задовольнити ці очікування, розробникам необхідно постійно підтримувати діалог із цільовою аудиторією. Вивчення зворотного зв’язку, аналіз поведінкових патернів, використання аналітики та залучення фокус-груп допомагає отримати реальну картину того, як сприймається платформа з боку користувачів. Водночас важливо не просто збирати дані, а й правильно їх інтерпретувати, формуючи конкретні рішення щодо покращення функціональності, дизайну чи продуктивності.

Варто також звернути увагу на зміну очікувань з плином часу. Те, що вважалося інноваційним кілька років тому, сьогодні може сприйматися як базова функція. Тому розробка веб-додатка повинна передбачати гнучкість і можливість масштабування, щоб відповідати новим викликам та вимогам. Успішна платформа – це та, що не лише відповідає поточним потребам користувачів, а й здатна передбачати їхні майбутні запити, пропонуючи інноваційні рішення ще до того, як вони стануть масовими очікуваннями.

У результаті системний і глибокий аналіз очікувань користувачів сприяє створенню веб-додатка, який буде не лише технічно досконалим, а й по-справжньому зручним, привабливим та конкурентним на ринку. Це забезпечує високий рівень залученості користувачів, сприяє формуванню лояльної спільноти навколо платформи та відкриває нові можливості для розвитку в умовах постійної цифрової еволюції.

Сучасні блог-платформи повинні забезпечувати базовий набір функцій, які дозволяють користувачам створювати, редагувати та публікувати контент без додаткових складнощів. Основні мінімальні вимоги до функціоналу блог-платформи включають:

1. Реєстрація та автентифікація користувачів: блог-платформа повинна мати можливість реєстрації нових користувачів, а також авторизації для вже зареєстрованих. Важливо, щоб підтримувались різні методи автентифікації, зокрема через електронну пошту та паролі, а також соціальні мережі (Facebook, Google, GitHub).
2. Створення та редагування контенту: платформа повинна дозволяти користувачам створювати статті, додавати текст, зображення, відео та інші медіа-елементи. Для цього часто використовуються WYSIWYG (What You See Is What You Get) редактори, які забезпечують простоту в написанні та форматуванні контенту без потреби в знаннях HTML.
3. Публікація та модерація контенту: після того, як користувач написав статтю, блог-платформа повинна дозволяти публікацію цього контенту, а також надавати можливість для модерації (у разі необхідності). Це може

бути важливим, якщо на платформі є кілька авторів або необхідність контролювати якість публікацій.

4. Категоризація та теги: для зручності користувачів та пошукових систем, платформа повинна підтримувати категорії та теги для кожної статті. Це дозволяє структурувати контент і робить його більш зручним для навігації.
5. Коментарі: більшість блог-платформ мають можливість додавання коментарів під статтями. Це сприяє інтерактивності та обміну думками між авторами та читачами. Важливо, щоб коментарі також були піддані модерації, аби запобігти спаму та непотрібному контенту.
6. Пошукова система по блогу: користувачі повинні мати можливість швидко знаходити контент через пошук. Це включає в себе індексацію статей за ключовими словами, категоріями та тегами.
7. Адаптивний дизайн: блог-платформа повинна бути зручною для використання на різних пристроях, таких як десктопи, планшети та мобільні телефони. Адаптивний дизайн або мобільна версія є важливою складовою сучасних платформ, оскільки значна частина трафіку походить від мобільних користувачів.

Безпека є однією з ключових вимог до будь-якої сучасної блог-платформи. Зважаючи на популярність блогів і великий обсяг особистої та корпоративної інформації, блог-платформи повинні мати захисні механізми для забезпечення конфіденційності та цілісності даних:

1. Шифрування даних: вся інформація, що передається між клієнтом і сервером, повинна бути зашифрована за допомогою протоколу HTTPS. Це забезпечує захист від перехоплення даних і є обов'язковим для будь-якої сучасної платформи.
2. Захист від SQL-ін'єкцій: блог-платформа повинна бути захищена від SQL-ін'єкцій, що є однією з найпоширеніших атак на веб-додатки. Це досягається через використання підготовлених запитів (prepared statements) та ORM (Object-Relational Mapping).

3. Механізми автентифікації та авторизації: для захисту облікових записів користувачів важливо реалізувати безпечні методи автентифікації, такі як двофакторна автентифікація (2FA) або автентифікація через OAuth.
4. Захист від CSRF (Cross-Site Request Forgery): платформа повинна включати механізми захисту від CSRF атак, які можуть бути використані для виконання небажаних дій користувача без його відома. Це реалізується через токени безпеки.
5. Регулярні оновлення та патчі безпеки: платформа повинна регулярно оновлюватися для виправлення вразливостей безпеки та забезпечення актуальних патчів, які знижують ризики атак.

Продуктивність блог-платформи визначається її здатністю обробляти великий потік запитів та забезпечувати швидке завантаження сторінок для користувачів. Платформа повинна відповідати наступним вимогам:

1. Швидкість завантаження сторінок: блог-платформа повинна мати оптимізовану структуру сторінок для швидкого завантаження. Це можна досягти за допомогою кешування контенту, мінімізації HTTP-запитів, використання CDN (Content Delivery Network) для швидкої доставки контенту з різних регіонів.
2. Масштабованість: блог-платформа повинна бути здатна обробляти збільшення навантаження з ростом кількості користувачів та контенту. Це включає горизонтальне масштабування (додавання нових серверів) та вертикальне масштабування (збільшення потужностей окремих серверів).
3. Кешування даних: використання кешування дозволяє значно зменшити час відповіді для часто запитуваних даних, таких як статті, коментарі, популярні теги. Кешування може здійснюватися як на рівні браузера, так і на сервері (наприклад, через Redis або Memcached).
4. Оптимізація бази даних: оскільки блог-платформи часто працюють з великими обсягами текстових даних, оптимізація бази даних є критично важливою. Це включає індексацію, нормалізацію даних, правильне

проектування схеми та використання запитів, які мінімізують навантаження на сервер.

Оптимізація для пошукових систем є важливою складовою успіху блогу, оскільки вона забезпечує видимість контенту для користувачів і сприяє залученню органічного трафіку. Веб-платформа для блогу повинна підтримувати наступні вимоги SEO:

1. Чисті URL-адреси: URL кожної статті повинна бути легко читається і включати ключові слова, що стосуються контенту (наприклад, example.com/seo-optimization-tips). Це допомагає не тільки користувачам, а й пошуковим системам зрозуміти, про що йдеться в статті.
2. Мета-теги: платформа повинна дозволяти авторам задавати мета-теги для кожної статті, зокрема мета-опис, ключові слова та заголовки, які використовуються для підвищення видимості в пошукових системах.
3. Мобільна оптимізація: оскільки мобільний трафік складає значну частину всього трафіку в інтернеті, блоги повинні бути повністю адаптовані до мобільних пристроїв. Це є одним з ключових факторів ранжування в пошукових системах.
4. Швидкість завантаження: пошукові системи (особливо Google) враховують швидкість завантаження сторінок як фактор ранжування. Тому важливо оптимізувати час завантаження сторінки для покращення SEO.

1.3 Проблеми та виклики, пов’язані з реалізацією блогів

Реалізація та підтримка сучасних блог-платформ включає безліч важливих аспектів, таких як забезпечення конфіденційності користувачів, захист авторських прав, ефективна модерація контенту та боротьба зі спамом і фейками. Ці проблеми не лише впливають на ефективність і репутацію платформи, а й ставлять перед розробниками та адміністраторами блогів серйозні виклики, які потребують постійної уваги та вдосконалення підходів. Розглянемо ці питання детальніше.

Конфіденційність є одним з найбільш важливих аспектів при створенні блог-платформи, адже будь-яка платформа, яка збирає особисті дані користувачів, повинна забезпечувати їхній захист і дотримання правил конфіденційності. Важливими моментами, які можуть призвести до проблем у цьому контексті, є:

1. Збір особистих даних: для того, щоб користувачі могли реєструватися на платформі, публікувати коментарі та вести свої блоги, платформа часто збирає персональну інформацію, таку як ім'я, електронна пошта, адреса тощо. Однак це створює ризик для порушення конфіденційності, якщо ці дані не захищені належним чином. У разі витоку або неправомірного доступу до такої інформації репутація платформи може бути серйозно піддана шкоді.
2. Збереження даних: користувачі можуть очікувати, що їхні особисті дані будуть зберігатися конфіденційно та не будуть передаватися третім особам без їхнього дозволу. Якщо блог-платформа порушує ці принципи, це може призвести до юридичних наслідків, зокрема штрафів відповідно до законів про захист персональних даних, таких як GDPR у Європейському Союзі.
3. Ризики для приватності: у сучасному світі конфіденційність користувачів може бути порушена не тільки через витоки даних, але й через механізми трекінгу та збору метаданих (наприклад, через cookies або IP-адреси). Веб-блог може зібрати надлишкову інформацію про користувачів, що ставить під загрозу їхню приватність. Тому блог-платформи повинні чітко інформувати користувачів про те, які дані вони збирають і як ці дані будуть використовуватися.
4. Безпека даних: захист даних користувачів — це не лише питання збору і зберігання, а й питання їхнього захисту від зовнішніх атак. Блог-платформи повинні бути належним чином захищені від хакерських атак, які можуть призвести до викрадення персональних даних. Важливо використовувати сучасні технології шифрування та захисту серверів, такі

як HTTPS, а також регулярно оновлювати програмне забезпечення, щоб зменшити ризик компрометації даних.

Інший важливий аспект, що виникає при реалізації блогів — це питання авторських прав. Оскільки більшість контенту на блог-платформах є результатом творчої діяльності користувачів, виникає ряд проблем, пов'язаних з порушенням авторських прав. Основні проблеми включають:

1. Порушення авторських прав: блогери можуть випадково або навмисно використовувати чужі матеріали, такі як зображення, відео, тексти, музику, без належного дозволу від власника авторських прав. Це може призвести до юридичних наслідків для автора та платформи, адже такі дії можуть порушувати законодавство щодо авторського права, зокрема, Закон про авторське право та суміжні права.
2. Використання нелегального контенту: чи то через неуважність, чи через спроби залучити більше трафіку, деякі блогери можуть використовувати контент без дозволу його творців, що стає проблемою для платформи. Веб-платформи повинні розробити механізми для виявлення та видалення нелегального контенту, що порушує авторські права.
3. Захист контенту користувачів: платформа також повинна забезпечити захист власних контенту користувачів. Це може включати механізми для запобігання крадіжці контенту, наприклад, за допомогою водяних знаків, обмежень на копіювання матеріалів або використання технологій для відстеження незаконного копіювання.
4. Ліцензування контенту: для зменшення ризику порушень авторських прав платформа може сприяти легальному використанню контенту через впровадження механізмів ліцензування. Наприклад, платформа може надавати авторам можливість публікувати свій контент під певними умовами ліцензій, як-от Creative Commons, що дозволяє іншим використовувати матеріали за певних обмежень.

Модерація контенту є важливим елементом підтримки здорової атмосфери на будь-якій блог-платформі. Вона допомагає уникати появи спаму,

непристойних матеріалів та фейків, а також зберігає репутацію платформи серед користувачів. Проблеми та виклики, пов'язані з модерацією контенту, включають:

1. Спам: спамери можуть використовувати блог-платформи для розповсюдження реклами або шкідливих посилань. Вони можуть публікувати коментарі з промоційними матеріалами, що знижує якість платформи та створює непотрібний трафік. Модератори повинні розробляти автоматизовані механізми для виявлення спаму (наприклад, через використання алгоритмів для виявлення спам-посилань) і вручну перевіряти повідомлення користувачів.
2. Неетичний контент: блог-платформи повинні забезпечити механізми для боротьби з непристойним контентом — образливими, дискримінаційними або неприязними коментарями, а також матеріалами, які порушують моральні або правові норми. Це може включати системи скарг від користувачів, а також автоматичні фільтри для виявлення неприязних або вульгарних виразів.
3. Боротьба з фейковими новинами: одним з великих викликів є боротьба з фейковими новинами, які можуть поширюватися через блоги. Оскільки багато блогерів мають власну аудиторію, недостовірна інформація може мати серйозні наслідки. Для цього платформи повинні впроваджувати механізми для перевірки достовірності публікацій, наприклад, співпрацювати з фактчекінговими організаціями або розробляти алгоритми для виявлення фейків.
4. Модерація в реальному часі: оскільки контент на платформі постійно змінюється і оновлюється, важливо забезпечити ефективну модерацію в реальному часі. Це може включати автоматичні системи для попередньої перевірки контенту, а також налаштування правил для ручної модерації, коли це необхідно.

Використання ботів та фейкових акаунтів для поширення фальшивої інформації чи спаму також є великою проблемою для блог-платформ. Для боротьби з такими явищами необхідно:

1. Верифікація акаунтів: веб-платформи можуть впровадити додаткові методи верифікації акаунтів, наприклад, через перевірку через соціальні мережі або за допомогою мобільного телефону.
2. Системи CAPTCHA: використання технологій CAPTCHA допомагає запобігти автоматичному створенню акаунтів або публікаціям, здійснюваним ботами.

1.4 Формулювання мети та постановка задачі

Розробка блогового веб-додатка є комплексним процесом, що передбачає створення цифрової платформи, яка дозволяє користувачам не лише ділитися власними думками та досвідом, а й активно взаємодіяти між собою, формуючи спільноти за інтересами. Основна ідея такої платформи полягає у створенні середовища, де кожен може створювати, редагувати, публікувати та обговорювати контент, маючи при цьому комфортні, безпечні й ефективні інструменти для взаємодії.

У сучасних умовах, коли інформаційне поле надзвичайно насичене, а користувачі мають високі очікування щодо якості та швидкодії цифрових продуктів, особливу увагу необхідно приділяти не лише функціоналу, а й загальному користувацькому досвіду. Це означає, що платформа повинна бути не тільки технічно досконалою, але й інтуїтивно зрозумілою, доступною з різних пристроїв і адаптованою до потреб різноманітної аудиторії.

Однією з основних задач при розробці блогового веб-додатка є створення системи управління контентом, яка б дозволяла користувачам без зайвих труднощів публікувати власні матеріали — від коротких текстів до повноцінних статей із мультимедійним супроводом. Система має забезпечувати можливість додавання зображень, відео, гіперпосилань, форматування тексту, а також класифікацію контенту за темами чи тегами для кращої навігації та

структурування інформації. Особливе значення тут має простота створення та редагування контенту — платформа повинна бути доступною навіть тим, хто не має технічного досвіду, забезпечуючи при цьому широкі можливості для самовираження.

Редагування та управління вже створеними публікаціями є не менш важливою задачею. Користувач повинен мати змогу легко оновлювати свої записи, виправляти помилки, видаляти застарілу або непотрібну інформацію, а також переглядати матеріал перед публікацією, щоб мати впевненість у його якості. Така функціональність сприяє підвищенню якості контенту на платформі та зменшенню кількості випадкових або неповних публікацій.

Інтеграція інструментів для соціальної взаємодії є ще одним ключовим завданням у межах реалізації такого додатка. Коментарі, оцінки публікацій (лайки, дизлайки), можливість підписки на авторів, повідомлення про нові пости чи відповіді — все це створює середовище для спілкування, залучення користувачів і підтримки активності на платформі. Чим більше можливостей взаємодії буде передбачено, тим більше шансів, що користувачі не просто будуть повертатися до блогу, а й активно формуватимуть навколо нього спільноту, генеруючи новий контент і обговорення.

Особливої уваги потребує питання безпеки та захисту персональних даних. Усі дані, що передаються через платформу — реєстраційна інформація, електронні адреси, паролі, вміст повідомлень, — повинні бути надійно захищені. Реалізація механізмів аутентифікації, шифрування, фільтрації шкідливого вмісту, а також запобігання спаму та зловживанням є критично важливою частиною задачі. Блоговий веб-додаток повинен відповідати сучасним вимогам інформаційної безпеки та бути захищеним від найбільш розповсюджених загроз, таких як SQL-ін’єкції, XSS-атаки, фішинг чи злом сесій.

Також важливо враховувати масштабованість та стабільність платформи. У разі зростання кількості користувачів чи обсягу контенту система повинна залишатися стабільною, швидкою у завантаженні й чіткою в роботі. Для цього потрібно продумати оптимальну архітектуру, ефективне кешування,

використання CDN для швидкої доставки контенту, а також резервне копіювання та моніторинг працездатності.

Окремо варто згадати про важливість адаптивного дизайну, який дозволить комфортно користуватись платформою як з комп’ютера, так і з мобільного пристрою. З урахуванням того, що значна частина користувачів споживає контент саме зі смартфонів, цей аспект не можна ігнорувати. Зручна мобільна версія не лише забезпечує кращий досвід, а й сприяє розширенню аудиторії додатка.

Отже, загальна постановка задачі полягає не лише у створенні інструменту для публікації контенту, а у формуванні цілісного цифрового середовища, де користувачі можуть самовиражатися, взаємодіяти між собою, відчувати безпеку та комфорт під час користування, і зберігати зацікавленість у тривалому використанні платформи. Комплексний підхід до розв’язання всіх зазначених завдань є запорукою успішної реалізації блогового веб-додатка та його подальшого розвитку.

РОЗДІЛ 2
ВИБІР ТЕХНІЧНИХ ЗАСОБІВ ДЛЯ РОЗРОБКИ ДОДАТКУ ДЛЯ ВЕДЕННЯ БЛОГУ

2.1 Основи функціонування веб-додатків, процес взаємодії між клієнтом та сервером

Сучасні веб-додатки є складними інтерактивними системами, що забезпечують користувачам можливість доступу до різноманітних функцій через браузер, без потреби встановлення додаткового програмного забезпечення. Основи їх функціонування ґрунтуються на поєднанні технологічних рішень, архітектурних підходів та механізмів взаємодії, які забезпечують стабільність, безпеку, продуктивність та зручність у використанні. Щоб повноцінно розуміти принципи роботи веб-додатків, доцільно розглянути їх ключові складові, серед яких: архітектура, технологічний стек, комунікація між компонентами, питання безпеки, забезпечення користувацького досвіду, а також розгортання та масштабування додатків.

Архітектура веб-додатка визначає загальну структуру системи, її логічне розділення на компоненти, а також способи їх взаємодії. Від вибору архітектури залежить ефективність функціонування додатка, його масштабованість, можливість супроводу, адаптації до змінних умов та розширення функціональності.

Найпоширенішими архітектурними моделями є:

- Клієнт-серверна модель — базова модель, яка передбачає, що клієнт (зазвичай веб-браузер) надсилає запити до сервера, а сервер обробляє ці запити та повертає відповіді у вигляді HTML-сторінок, JSON-даних або інших форматів. Такий підхід дозволяє розділити логіку обробки даних та їх подання.
- Монолітна архітектура — класичний підхід до побудови веб-додатків, коли всі функціональні частини (інтерфейс, логіка, база даних) реалізовані в межах єдиної програми. Моноліт простіше реалізувати на початкових

етапах, однак зі зростанням складності додатка ускладнюється його обслуговування та масштабування.

- Архітектура мікросервісів — сучасна розподілена модель, в якій додаток складається з набору незалежних сервісів, кожен з яких виконує окрему бізнес-функцію. Мікросервіси взаємодіють між собою через API, що дозволяє досягти гнучкості, масштабованості, а також спрощує незалежне розгортання та оновлення компонентів.

Технологічний стек веб-додатка — це сукупність мов програмування, фреймворків, бібліотек та інструментів, які використовуються для реалізації його функціональності. Його можна умовно поділити на кілька основних частин:

- Фронтенд (клієнтська частина) — відповідає за зовнішній вигляд та взаємодію користувача з системою. До основних технологій належать:
  - HTML (HyperText Markup Language) — мова розмітки сторінок;
  - CSS (Cascading Style Sheets) — описує стильове оформлення елементів;
  - JavaScript — мова програмування для створення динаміки на сторінці;
  - Фреймворки та бібліотеки: React, Angular, Vue.js — дозволяють будувати масштабовані інтерактивні інтерфейси з компонентною архітектурою.
- Бекенд (серверна частина**)** — реалізує логіку обробки запитів, управління базами даних, автентифікацію тощо. Можуть використовуватись такі мови та фреймворки:
  - Python (Django, Flask);
  - JavaScript (Node.js + Express);
  - PHP (Laravel);
  - Java (Spring Boot);
  - Ruby (Rails).
- Бази даних — використовуються для зберігання контенту, облікових записів, коментарів тощо:

- Реляційні (PostgreSQL, MySQL) — базуються на таблицях та SQL.
- Нереляційні (MongoDB, Redis) — гнучкіші у структурі зберігання даних, особливо ефективні для роботи з великими обсягами інформації.

У зв’язку зі збільшенням кількості кібератак, забезпечення безпеки веб-додатка є пріоритетним завданням. До основних механізмів належать:

- Автентифікація — перевірка особи користувача (наприклад, через логін і пароль, токени, OAuth 2.0).
- Авторизація — визначення прав доступу до ресурсів після автентифікації.
- Захист від атак:
  - SQL-ін'єкції — впровадження шкідливого SQL-коду.
  - XSS (Cross-Site Scripting) — впровадження шкідливого скрипта у сторінку.
  - CSRF (Cross-Site Request Forgery) — спроби несанкціонованих дій від імені користувача.
- Шифрування даних — зазвичай реалізується через HTTPS та внутрішні методи шифрування для конфіденційної інформації.

Веб-додатки функціонують завдяки ефективному обміну даними між клієнтською і серверною частинами. Ця взаємодія може бути реалізована через:

- API (Application Programming Interface) — набір правил для запиту і передачі даних. REST та GraphQL є найпоширенішими рішеннями.
- Асинхронна комунікація — використовується AJAX, Fetch API або Axios для завантаження даних у фоновому режимі без оновлення сторінки.
- Технології реального часу — WebSockets, Server-Sent Events (SSE) дозволяють забезпечити миттєву доставку повідомлень, наприклад у чатах або системах сповіщень.

Інтерфейс веб-додатка має бути не лише функціональним, а й привабливим для користувача. Це забезпечується:

- Фронтенд-фреймворками, які дозволяють створювати SPA (Single Page Applications), де сторінка не перезавантажується повністю, а оновлюються лише її частини.
- Зручним UI/UX-дизайном — передбачає продумане розміщення елементів, логічну навігацію, інтерактивні елементи (форми, кнопки, меню, анімації).
- Динамічним контентом, який може оновлюватися на основі дій користувача чи змін на сервері без перезавантаження.

Після завершення розробки веб-додаток необхідно правильно розгорнути та забезпечити його працездатність у різних умовах:

- Контейнери — інструменти на кшталт Docker дозволяють ізолювати середовище додатка та спростити його розгортання.
- Оркестрація — з використанням Kubernetes можна автоматизувати управління великою кількістю контейнерів, масштабувати додаток та забезпечити високу доступність.
- Масштабування:
  - Горизонтальне — додавання нових екземплярів сервісу.
  - Вертикальне — збільшення ресурсів (CPU, RAM) на одному сервері.
- Інструменти моніторингу — Prometheus, Grafana, New Relic тощо — допомагають відстежувати навантаження, виявляти помилки і підтримувати стабільну роботу системи.

Узагальнюючи, можна стверджувати, що основи функціонування веб-додатків охоплюють широкий спектр технічних і концептуальних питань, які мають бути враховані під час розробки. Збалансоване поєднання правильної архітектури, сучасного технологічного стеку, безпеки, інтерактивності та стратегії розгортання є необхідною умовою для створення стабільного та ефективного веб-додатка, що відповідає вимогам сучасних користувачів і реалій ринку.

Процес взаємодії між клієнтом і сервером є фундаментальною основою функціонування сучасних веб-додатків. Цей процес реалізується за допомогою

клієнт-серверної моделі, яка передбачає чітке розділення обов'язків між двома сторонами: клієнтом (frontend) та сервером (backend). Така архітектура забезпечує масштабованість, надійність, зручність розробки та експлуатації, а також високу продуктивність під час обробки численних запитів користувачів.

Клієнт — це кінцевий пристрій або програмне забезпечення (найчастіше — веб-браузер, мобільний застосунок або десктопна програма), з яким безпосередньо взаємодіє користувач. Основна функція клієнта полягає у забезпеченні інтерфейсу користувача та ініціації запитів до сервера. Ключові складові клієнтської частини:

• Інтерфейс користувача (UI): реалізований за допомогою HTML для структурування контенту та CSS для оформлення і стилізації сторінок. Для формування динамічного контенту використовується шаблонізатор Jinja2, який дозволяє відображати змінні, умови та цикли безпосередньо в HTML, забезпечуючи зручну інтеграцію з серверною логікою на Flask.

• Обробка дій користувача: основна взаємодія відбувається через стандартні елементи HTML-форм. Дані, введені користувачем, надсилаються на сервер через HTTP-запити (GET або POST), де обробляються у Flask. Після цього оновлені дані повертаються у відповідь, і сторінка перезавантажується або оновлюється з новим вмістом.

Сервер — це програмне та апаратне середовище, яке приймає запити від клієнтів, обробляє їх та формує відповідь. Він реалізує бізнес-логіку, взаємодіє з базами даних, перевіряє права доступу та забезпечує безпеку. Основні функції сервера:

- Обробка запитів: серверна частина, написана на такій мові, як Python, отримує запит від клієнта, аналізує його та виконує відповідну бізнес-логіку — від простої перевірки даних до складних обчислень або звернень до сторонніх API.
- Взаємодія з базою даних: серверна частина працює з базою даних SQLite, яка зберігає всю необхідну інформацію — зокрема дані користувачів, публікацій, коментарів тощо. При запиті, наприклад, до профілю

користувача, Flask надсилає відповідний SQL-запит до бази, отримує результати та передає їх шаблону Jinja2 для подальшого відображення на веб-сторінці.

- Забезпечення безпеки:
  - Автентифікація — перевірка особи користувача, наприклад, через логін/пароль, OAuth або JWT-токени.
  - Авторизація — перевірка прав доступу до певних ресурсів.
  - Шифрування — передача конфіденційних даних через HTTPS.
- Формування та надсилання відповіді: після обробки запиту сервер генерує відповідь, яка повертається клієнту. Це може бути:
  - JSON-об'єкт із даними;
  - HTML-сторінка;
  - повідомлення про помилку або підтвердження дії.

Взаємодія між клієнтом та сервером відбувається через мережу Інтернет із використанням стандартних протоколів:

- HTTP/HTTPS: основні протоколи, які забезпечують передачу даних між клієнтом і сервером. HTTPS додає шифрування для захисту інформації.
- Синхронна комунікація: клієнт надсилає запит і чекає відповіді — наприклад, традиційне завантаження сторінки.

2.2 Порівняльний аналіз технологій для створення веб-додатків

Під час розробки будь-якого програмного продукту надзвичайно важливо зважено підійти до вибору інструментів, з якими буде працювати команда. Це не лише питання зручності чи популярності – від обраних технологій залежить, наскільки легко вдасться реалізувати задум, як швидко просуватиметься робота, і наскільки просто буде підтримувати та вдосконалювати систему у майбутньому.

У межах проєкту для створення серверної частини (тобто тієї частини, яка відповідає за логіку програми, роботу з базою даних і взаємодію між користувачами) було обрано фреймворк Flask, написаний мовою програмування

Python. Такий вибір не був випадковим, а базувався на кількох важливих причинах, які варто розглянути детальніше.

Flask – це легкий і зручний інструмент, який дозволяє розробникам не перевантажувати програму зайвими модулями, а використовувати лише те, що дійсно потрібно. Саме це і стало однією з головних переваг: завдяки своїй простоті Flask надає розробнику повну свободу у тому, як організувати проєкт, не нав'язуючи готових шаблонів. Це особливо зручно, коли проєкт має власну логіку, яка не вписується у типові «коробкові» рішення.

Python, на якому побудовано Flask, також був важливим чинником у виборі. Це мова, яка вирізняється зрозумілим синтаксисом, гарною читабельністю та великою спільнотою. Вона активно використовується не лише у веброзробці, а й у сфері аналітики, штучного інтелекту, автоматизації тощо. Це означає, що у майбутньому до проєкту буде легко додати нові функції, наприклад, аналітичні модулі або автоматичні нагадування, не змінюючи основи системи.

На етапі вибору також розглядалися альтернативи — зокрема, фреймворк Django (також побудований на Python) та середовище Node.js, яке використовує мову JavaScript.

Django надає багато функцій «із коробки», тобто одразу має вбудовані інструменти для обліку користувачів, адміністративної панелі, перевірки даних тощо. Це зручно для великих проєктів, однак іноді така надмірна структура лише ускладнює роботу, якщо потрібно створити щось просте й гнучке. У випадку нашого проєкту Django видався надто «важким» та складним для адаптації.

Node.js, своєю чергою, дозволяє будувати вебдодатки на мові JavaScript. Це може бути корисно, коли клієнтська й серверна частина виконані на одній мові, однак у нашому випадку значно зручніше було працювати саме з Python, який краще підходить для логіки, обчислень і роботи з даними.

Ще однією перевагою Flask стала його спільнота: багато корисних прикладів, документації, порад та відкритих бібліотек дають змогу швидко знаходити відповіді на запитання й уникати поширених помилок. Крім того,

Flask добре масштабується: якщо згодом з’явиться потреба розширити функціональність або залучити більше користувачів, проєкт не доведеться переписувати з нуля.

Таким чином, вибір Flask як основи для серверної частини вебдодатку був логічним і зваженим рішенням. Його гнучкість, простота і хороша сумісність з іншими інструментами робить його ідеальним для реалізації проєктів, де важливо мати контроль над архітектурою і зосередитись саме на унікальній логіці додатку, а не на налаштуванні складних інструментів.

2.3 Архітектура веб-додатку блогу

Під час створення вебдодатків важливо ретельно продумати архітектуру, адже саме вона визначає, наскільки легко буде розвивати, масштабувати та підтримувати систему в майбутньому. Особливо це стосується таких проєктів, як блог-платформи, де користувачі регулярно взаємодіють з інтерфейсом, створюють контент, залишають коментарі та працюють з персональними кабінетами.

У рамках проєкту було прийнято рішення реалізовувати систему за клієнт-серверною моделлю, коли клієнтська частина (інтерфейс, з яким взаємодіє користувач) звертається до серверної частини, яка обробляє запити, працює з базою даних і повертає результати. Такий підхід дозволяє чітко розділити відповідальність між зовнішнім виглядом додатку та його внутрішньою логікою.

Розглянемо два основні варіанти архітектури, які були проаналізовані перед початком реалізації:

1. Монолітний підхід означає, що вся логіка системи — автентифікація, управління дописами, коментарі, обробка зображень тощо — об'єднана в одному цілісному застосунку. Такий формат особливо зручний для невеликих або середніх проєктів, а також на початкових етапах розробки, коли важливо швидко отримати робочу версію. Його переваги — простий запуск, швидка розробка MVP та мінімальні витрати на інфраструктуру. Водночас монолітна архітектура має і свої обмеження: складно масштабувати окремі функції, оновлення чи

виправлення часто вимагає зупинки всієї системи, а з часом код може перетворитися на заплутану структуру, що ускладнює подальшу підтримку.

2. Мікросервісний підхід передбачає поділ застосунку на окремі, незалежні сервіси, кожен з яких відповідає за свою чітко визначену функцію — наприклад, один модуль відповідає за авторизацію, інший — за збереження медіа або керування коментарями. Така структура дозволяє гнучко масштабувати лише ті компоненти, що піддаються навантаженню, оновлювати окремі частини системи без зупинки всього проєкту та полегшує підтримку завдяки розмежуванню відповідальності. Водночас вона вимагає більше технічних зусиль для налаштування: потрібно забезпечити взаємодію між сервісами, обробляти можливі помилки, слідкувати за станом кожної частини системи, а тестування та налагодження стають складнішими, особливо на старті.

З урахуванням масштабу й цілей проєкту, для реалізації блог-платформи було обрано клієнт-серверну архітектуру. Такий підхід забезпечує оптимальний баланс між простотою реалізації та можливістю розвитку в майбутньому. Усі ключові функції реалізовані в межах єдиного застосунку, що значно пришвидшує процес розробки, особливо на перших етапах. При цьому клієнт і сервер чітко розділені, що дозволяє легко адаптувати інтерфейс до різних пристроїв або замінити його у майбутньому без зміни основної логіки системи.

РОЗДІЛ 3
СТВОРЕННЯ WEB-ДОДАТКУ

3.1 Створення бази даних

У процесі розробки вебсайту було створено реляційну базу даних, що забезпечує зберігання та ефективну обробку інформації про користувачів, публікації, коментарі та інші елементи контенту. База даних побудована таким чином, щоб забезпечити цілісність даних, а також швидкий доступ до інформації, що необхідна для коректного функціонування сайту.

Таблиця «Mainmenu» відповідає за зберігання основних пунктів меню, що використовуються для навігації по сайту. Вона містить три поля: id, яке є унікальним ідентифікатором кожного запису в таблиці, title, що визначає назву пункту меню, та url, яке вказує на відповідний шлях або адресу для кожного пункту. Завдяки цій таблиці користувачі можуть легко орієнтуватися в структурі сайту та переходити до потрібних розділів.

```
create table if not exists Mainmenu (
    id integer primary key autoincrement,
    title text not null,
    url text not null
);
```

Рисунок 3.1 – SQL-код для створення таблиці «Mainmenu»

Таблиця «Users» зберігає дані про користувачів вебсайту. Вона включає кілька полів: user_id, який є унікальним ідентифікатором кожного користувача, username, що містить ім'я користувача, яке відображатиметься на сайті, та email, що зберігає електронну адресу користувача. Також в таблиці є поле psw, яке зберігає зашифрований пароль для аутентифікації, а поле avatar дозволяє зберігати зображення аватарки користувача (якщо таке є). Поле date_registered вказує дату реєстрації користувача на сайті.

```
create table if not exists Users (
    user_id integer primary key autoincrement,
    username varchar(255) not null,
    email varchar(255) not null,
    psw varchar(255) not null,
    avatar blob default null,
    date_registered date
);
```

Рисунок 3.2 – SQL-код для створення таблиці «Users»

Таблиця «Posts» містить публікації, які створюються користувачами на сайті. Вона включає такі поля, як post_id — унікальний ідентифікатор публікації, user_id, яке є зовнішнім ключем до таблиці **Users** і визначає, який користувач створив публікацію. Поле title зберігає заголовок посту, а post_content — текстовий контент публікації. Поле tags містить теги, що описують зміст публікації, а post_img — зображення, яке може бути додано до посту. Поле date_posted вказує дату публікації.

```
create table if not exists Posts (
    post_id integer primary key autoincrement,
    user_id integer not null,
    title varchar(255) not null,
    post_content text not null,
    tags varchar(255) not null,
    post_img blob default null,
    date_posted date,
    foreign key (user_id) references Users(user_id)
);
```

Рисунок 3.3 – SQL-код для створення таблиці «Posts»

Таблиця «Comments» зберігає коментарі, які користувачі можуть залишати під публікаціями. Вона містить кілька полів: comment_id, яке є унікальним ідентифікатором коментаря, user_id, що вказує на користувача, який залишив коментар (це зовнішній ключ до таблиці **Users**), а також post_id, яке вказує на публікацію, до якої був залишений коментар (зовнішній ключ до таблиці **Posts**). Поле comment_content зберігає текст коментаря, а date_commented — дату написання коментаря.

```
create table if not exists Comments (
    comment_id integer primary key autoincrement,
    user_id integer,
    post_id integer,
    comment_content text,
    date_commented date,
    foreign key (user_id) references Users(user_id),
    foreign key (post_id) references Posts(post_id)
);
```

Рисунок 3.4 – SQL-код для створення таблиці «Comments»

Таблиця «Tags» містить унікальні теги, що можуть бути прикріплені до публікацій для їх класифікації. Вона складається з двох полів: tag_id, яке є унікальним ідентифікатором кожного тегу, та tag_text, що містить текст тегу.

```
create table if not exists Tags (
    tag_id integer primary key autoincrement,
    tag_text text
);
```

Рисунок 3.5 – SQL-код для створення таблиці «Comments»

Для забезпечення логічної цілісності даних між таблицями використовуються зовнішні ключі. Зокрема, таблиця «Posts» має зв'язок з таблицею «Users» через поле user_id, що дозволяє визначити, який користувач створив конкретну публікацію. Таблиця «Comments» має зв'язки з таблицями «Users» та «Posts» через поля user_id та post_id відповідно, що дозволяє визначити, хто залишив коментар і до якої публікації він відноситься.

Використання зовнішніх ключів дозволяє підтримувати цілісність даних в базі. Це означає, що кожен запис у таблицях буде пов'язаний з іншими відповідними записами, наприклад, кожен коментар обов'язково буде прив'язаний до конкретного користувача і публікації. Така структура забезпечує надійність та точність даних, а також запобігає виникненню помилок, пов'язаних з некоректними зв'язками між елементами сайту.

Таким чином, створення бази даних забезпечує стабільну та ефективну роботу сайту, дозволяючи зберігати, обробляти і взаємодіяти з контентом на платформі.

3.2 Розробка back-end частини

Back-end частина веб-додатка відповідає за обробку запитів від клієнта, виконання бізнес-логіки, зберігання даних і взаємодію з базами даних. Для розробки цієї частини веб-додатка можна використовувати різні мови програмування, фреймворки та бази даних. У роботі були використані такі технології як: Python, Flask та SQLite.

Python — це високорівнева мова програмування, що має простий і читабельний синтаксис. Вона широко використовується для розробки веб-додатків, завдяки своїй гнучкості, великій кількості бібліотек і активній спільноті. Використання Python для бекенду має такі переваги:

- Простота: синтаксис Python простий у вивченні та використанні, що прискорює процес розробки.
- Багатофункціональність: Python має велику екосистему бібліотек для різних задач, включаючи веб-розробку, аналіз даних, машинне навчання тощо.
- Портативність: Python є крос-платформенною мовою, що дозволяє розгортати додатки на різних операційних системах.

В якості фреймворку для розробки веб-додатку я обрала Flask. Flask — це мікрофреймворк для розробки веб-додатків на Python. Він надає мінімальну структуру, дозволяючи розробникам будувати бекенд з необхідними компонентами. Основні характеристики Flask включають:

- Легкість та гнучкість: Flask надає базовий набір функцій для створення веб-додатків, але дозволяє розширювати їх за потреби. Це робить його ідеальним для невеликих і середніх проектів.
- Модульність: Flask дозволяє легко інтегрувати додаткові розширення для підтримки баз даних, аутентифікації, обробки форм тощо. Це сприяє розширенню функціональності без надмірного ускладнення.
- Ком'юніті та ресурси: Flask має велику спільноту розробників і широкий спектр навчальних ресурсів, що допомагає швидко освоїти фреймворк та вирішувати проблеми.

Обраною базою даних стала SQLite, так як SQLite — це вбудована реляційна база даних, яка використовується для зберігання даних у веб-додатках. На відміну від традиційних реляційних баз даних, SQLite не потребує окремого сервера для запуску, що робить його зручним для невеликих проектів і прототипів. Переваги SQLite включають:

- Легкість та простота: SQLite не вимагає складної конфігурації або окремого процесу запуску. База даних зберігається у вигляді одного файлу, що спрощує розгортання.
- Підтримка SQL: SQLite підтримує основні команди SQL, що дозволяє використовувати стандартні запити для зберігання та отримання даних.
- Портативність: оскільки SQLite зберігає дані в одному файлі, його легко переносити між різними системами та інтегрувати в інші проекти.

Використання Python як основної мови програмування, Flask як веб-фреймворку, та SQLite як бази даних створює потужну та гнучку платформу для розробки бекенду. Python забезпечує простоту та багатофункціональність, Flask дозволяє будувати веб-додатки з мінімальними обмеженнями, а SQLite забезпечує зберігання даних без необхідності складної інфраструктури.

Таке поєднання технологій є ідеальним для невеликих і середніх проектів, а також для прототипування та швидкої розробки. Воно дозволяє створювати бекенд, який легко підтримувати, розширювати та масштабувати, забезпечуючи при цьому високу продуктивність і надійність.

На початку розробки проєкту був створений файл app.py, це головний файл в бекенді додатку, тут код демонструє структуру Flask-додатка, який використовує кілька різних бібліотек і технологій, таких як Flask, Flask-Login для аутентифікації, SQLite як базу даних, а також різні інші модулі для обробки веб-запитів, завантаження файлів, роботи з шаблонами тощо.

Ось основні частини коду, які варто виділити:

1. Конфігурація додатка: додаток Flask налаштований для використання SQLite як бази даних та має інші конфігураційні параметри, такі як секретний ключ, максимальний розмір контенту та інші налаштування.

```
app = Flask(__name__)
app.config.from_object(__name__)

app.config.update(dict(DATABASE=os.path.join(app.root_path, 'flblog.db')))
```

Рисунок 3.6 – Лістинг програми: конфігурація додатка

2. Управління користувачами: Flask-Login використовується для аутентифікації та управління користувачами. Код включає функції для завантаження користувачів, а також маршрути для входу, виходу та реєстрації.

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

@login_manager.user_loader
def load_user(user_id):
    return UserLogin().fromDB(user_id, dbase)
```

Рисунок 3.7 – Лістинг програми: управління користувачами

3. Робота з базою даних: SQLite використовується як база даних, і Flask-SQLAlchemy допомагає з підключенням та взаємодією з базою даних. Код містить функції для підключення до бази даних, створення бази даних та закриття з'єднання.

```
def connect_db():
    conn = sqlite3.connect(app.config['DATABASE'])
    conn.row_factory = sqlite3.Row
    return conn

def create_db():
    db = connect_db()
    with app.open_resource('sql_db.sql', mode='r') as f:
        db.cursor().execute(f.read())
    db.commit()
    db.close()
```

Рисунок 3.8 – Лістинг програми: робота з БД

4. Маршрути та обробники: код містить різні маршрути для різних дій, таких як перегляд головної сторінки, сортування за тегами, пошук, вхід, реєстрація,

додавання постів, коментарів та інші дії. Кожен маршрут пов'язаний з певною функцією, яка обробляє запити та повертає відповідь (весь код можна переглянути в додатку Б).

5. Функції допоміжних процесів: ці функції використовуються для підтримки різних аспектів веб-додатка, таких як обробка дат, взаємодія з базою даних, закриття з'єднання тощо.

```
dbase = None
@app.before_request
def before_request():
    global dbase
    db = get_db()
    dbase = FDataBase(db)

@app.teardown_appcontext
def close_db(error):
    if hasattr(g, 'link_db'):
        g.link_db.close()
```

Рисунок 3.9 – Лістинг програми: допоміжні функції

6. Взаємодія з клієнтською стороною: код містить обробники запитів, які працюють з фронтендом, обробляючи дані, завантажуючи файли, керуючи сесіями, авторизацією та іншим.

```
@app.route('/profile')
@login_required
def profile():
    my_posts = dbase.getMyPosts(current_user.get_id())
    return render_template('profile.html', menu=dbase.getMenu(), my_posts=my_posts)
```

Рисунок 3.10 – Лістинг програми: функція взаємодії з клієнтом

Також був створений файл UserLogin.py в якому клас, UserLogin, використовується для управління інформацією про користувачів у Flask-додатку, який використовує Flask-Login для керування сесіями користувачів та автентифікацією. Клас містить кілька методів, що дозволяють працювати з даними користувачів, отримувати їхню інформацію, а також здійснювати деякі перевірки (код знаходиться в додатку Б).

І ще один файл FDataBase.py в якому знаходиться клас FDataBase призначений для роботи з базою даних SQLite. Він забезпечує методи для отримання, додавання, оновлення та видалення даних з різних таблиць, а також обробляє взаємодію між бекендом і базою даних(код знаходиться в додатку Б).

Для користувачів були розроблені такі функції, які забезпечують зручне створення та перегляд контенту, взаємодію між учасниками платформи, а також сучасні способи подачі інформації. Усі елементи сайту орієнтовані на простоту використання, функціональність і доступність.

Основні можливості сайту:

1. Реєстрація та вхід у систему: кожен охочий може створити свій обліковий запис. Після авторизації користувач отримує доступ до повного функціоналу — зокрема, створення постів і коментування.

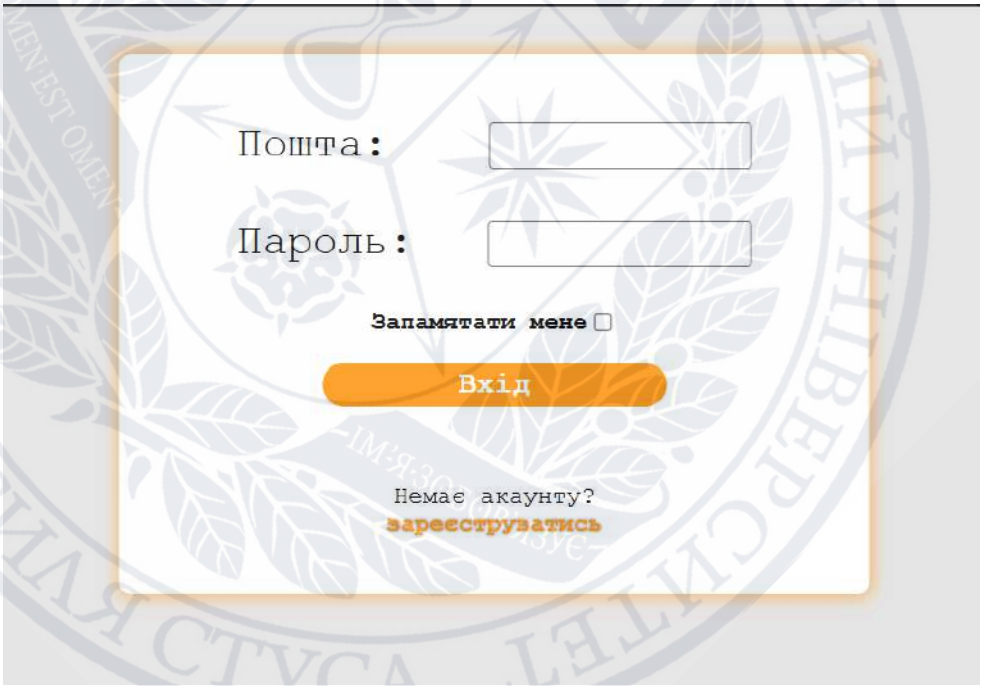

Рисунок 3.11 – Форма входу для користувачів

2. Створення публікацій: зареєстровані користувачі можуть створювати власні пости, додаючи текстовий опис, теги та фотографію (за бажанням). Також передбачена можливість публікувати пости анонімно — у такому випадку ім’я користувача не відображається.

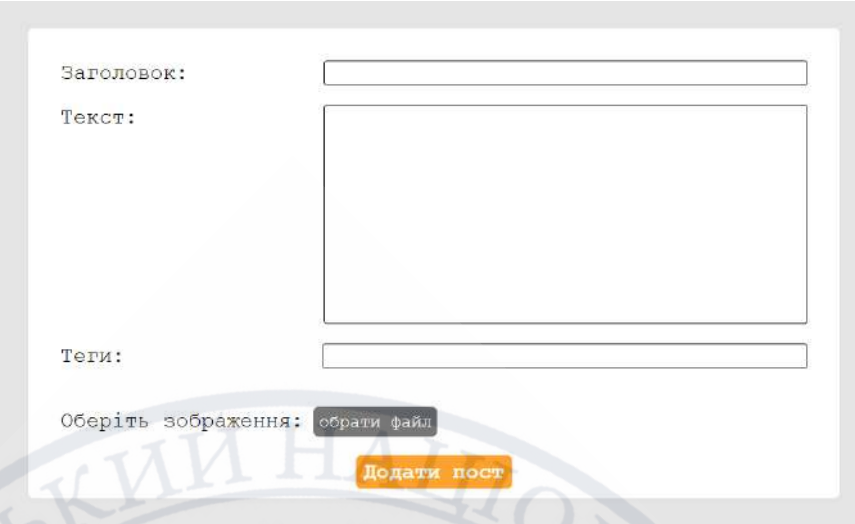

Рисунок 3.12 – Форма додавання допису

3. Перегляд та коментування: відвідувачі сайту можуть переглядати публікації, а зареєстровані користувачі — залишати до них коментарі, що сприяє взаємодії між людьми на платформі.

Рисунок 3.13 – Перегляд допису

4. Оцінка часу на прочитання: щоб користувач міг краще оцінити обсяг інформації, біля кожного поста відображається приблизний час, необхідний для його прочитання. Це дозволяє краще планувати час на ознайомлення з контентом.

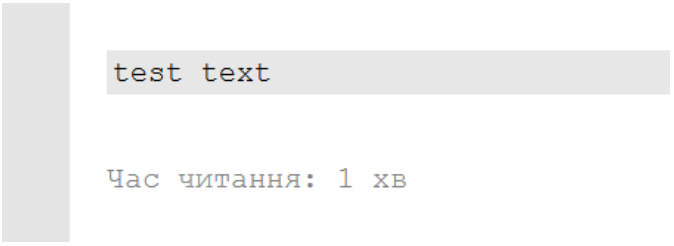

Рисунок 3.14 – Оцінка часу на прочитання

5. Голосове озвучення тексту: для зручності користувачів реалізована функція озвучення поста голосовим помічником. Це може бути особливо корисним у ситуаціях, коли немає можливості читати з екрану, або для людей з вадами зору.

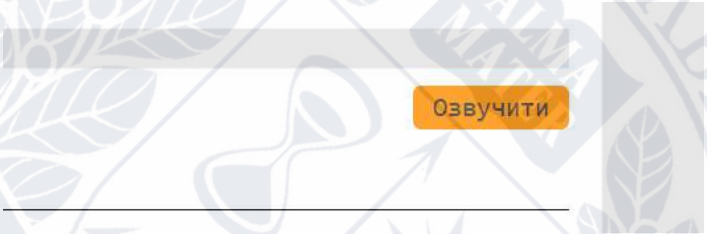

Рисунок 3.15 – Можливість озвучування тексту

6. Фільтрація постів: реалізовані зручні інструменти фільтрації контенту: користувач може переглядати пости за тегами (тобто за тематиками) або обирати найновіші публікації.

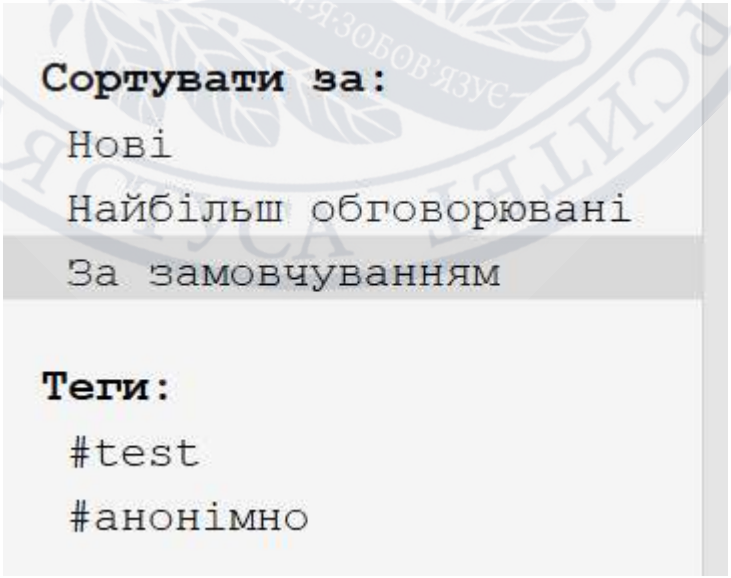

Рисунок 3.16 – Фільтрація постів

Загалом, сайт створений як простий у використанні простір для обміну думками, досвідом або просто цікавими матеріалами. Уся функціональність орієнтована на зручність користувача та сучасні підходи до подання й споживання інформації.

3.3 Розробка front-end частини

Клієнтська частина вебзастосунку реалізована з використанням класичних вебтехнологій — HTML, CSS та шаблонізатора Jinja2, що дозволяє створювати динамічні вебсторінки без залучення складних JavaScript-фреймворків. Такий підхід забезпечив простоту, швидкість розробки, а також легкість у подальшому супроводі коду.

Розмітка сторінок формувалась за допомогою Jinja2, що дозволяє передавати дані з серверної частини безпосередньо у шаблони. CSS використовувався для створення адаптивного та зручного інтерфейсу, що коректно відображається як на комп’ютерах, так і на мобільних пристроях.

JavaScript у застосунку використовується точково — виключно для реалізації функції озвучування тексту публікацій голосовим помічником.

Головна сторінка є першою точкою входу для користувача після авторизації. Вона відображає список публікацій, опублікованих іншими користувачами. Для зручності реалізована можливість фільтрації постів за тегами, а також сортування за новизною. Також на головній сторінці кожному посту відображається:

- короткий опис;
- автор (або позначка «Анонім», якщо публікація була створена анонімно);
- теги;

```
<div class="all-posts">
    {% if posts %}
        <ul>
            {% for post in posts %}
            <div class="each-post">
                <li>
                    <div class="head">
                        <img class="ava-img" src="{{ url_for('post_userava', user_id=post.user_id) }}" alt="">
                        <div class="head-text">
                            <span>{{ post.username }}</span>
                            <br>
                            <span> {{ post.date }}</span>
                        </div>
                    </div>
                    <a href="{{ url_for('show_post' , post_id=post.post_id) }}">
                        <img class="post-img" src="{{ url_for('post_img', post_id=post.post_id) }}" alt="">
                    </a>
                    <br>
                    <span><a class="title" href="{{ url_for('show_post' , post_id=post.post_id) }}">{{ post.title }}</a></span>
                    <br>
                    <br>
                    <span class="cont" >{{ post.content[:50] }}...</span>
                    <div class="comment">
                        <span>
                            <svg width="24px" height="24px" viewBox="0 0 32 32" version="1.1" xmlns="http://www.w3.org/2000/svg"
                            {{ post.comment_count }}
                        </span>
                        {% for tag in post.tags.split(' ') %}
                                <span><a class="each-tag" href="{{ url_for('sort_by_tags', tag=tag) }}">#{{ tag }}  </a></span>
                        {% endfor %}
                    </div>
```

Рисунок 3.17 – Лістинг програми: відображення дописів

Для користувачів реалізовані прості й зрозумілі форми авторизації та реєстрації. У формі реєстрації передбачені поля для введення логіна, пароля та підтвердження пароля. Усі введені дані перевіряються на коректність, а у разі помилки користувачу відображається відповідне повідомлення.

```
<form action="{{url_for('register')}}" method="post">
    {{ form.hidden_tag() }}
    {% for field in form if field.name not in ['csrf_token', 'submit'] -%}
    <div class="each-field-regis">
        <span>{{ field.label() }}</span>
        {% if field.errors %}
        <span>{{ field(class="enter-field invalid") }}</span>
        <span class="invalid-feedback">···
        </span>
        {% else %}
            <span>{{ field(class="enter-field") }}</span>
        {% endif %}
    </div>
    {% endfor %}
    <p>{{ form.submit(class='input-submit regist-btn') }}

</form>
```

Рисунок 3.18 – Лістинг програми: форма реєстрації

Сторінка створення публікації містить форму, за допомогою якої користувач може створити новий допис. Передбачено введення:

- заголовку;
- текстового опису (основний контент поста);
- одного або кількох тегів;
- можливості додати зображення;

Після публікації користувач автоматично перенаправляється на головну сторінку, де бачить свій пост у загальному списку.

```
<form action="{{url_for('add_post')}}" method="post" enctype="multipart/form-data">
    <div class="add-post-div">
        <span>Заголовок: <input class="title-input" type="text" name="name"></span> <br>
        <span>Текст: <textarea class="text-input" type="text" name="text"></textarea></span> <br>
        <span>Теги: <input class="tags-input" type="text" name="tags"></span> <br>
        <label class="add-post-sel-file" for="filename">обрати файл</label>
        <span>Оберіть зображення: <input type="file" id="filename" name="file_post" accept="image/*"></span>
        <span><input class="btn-add-post" type="submit" value="Додати пост"></span>
    </div>
</form>
```

Рисунок 3.19 – Лістинг програми: форма створення публікації

На сторінці окремої публікації користувач бачить повну версію поста разом із додатковою інформацією: ім’ям автора (або позначкою “Анонім”), списком тегів, точною датою публікації та блоком коментарів, у якому можна залишити власний відгук. Для зручності сприйняття контенту реалізована кнопка озвучування, ця функція особливо корисна для користувачів із порушеннями зору або в ситуаціях, коли немає змоги читати.

```
    <span>Від {{ res.username }} </span>
    <span class="title-to-post">{{ res.title }} </span>
    <img class="img-to-post" src="{{ url_for('post_img', post_id=res.post_id) }}" alt="">
    <span class="content-to-post">{{ res.post_content }} </span>

    <button class="to-voice-button" onclick="speakWithResponsiveVoice()">Озвучити</button>
    <span style="color: gray;">Час читання: {{ read_time }} хв</span>


    <span class="comment-span">Коментарі</span>
    {% for c in coms %}
    <span style="margin-bottom: 5px;">{{ c.user }}: {{ c.comment_content }} </span>
    {% endfor %}
    <form action="{{url_for('add_comments', post_id=post_id)}}" method="post">
        <input class="enter-field" style="margin-top: 5px;" type="text" name="comment" place
        <button type="submit" class="post-comment">
            <svg width="24px" height="24px"  viewBox="0 -4.15 57.875 57.875" xmlns="http://
        </button>
    </form>
</div>

<script>
    function speakWithResponsiveVoice() {
        const text = document.querySelector('.content-to-post').textContent.trim();
        if (text) {
            responsiveVoice.speak(text, "Ukrainian Female", {rate: 1});
        }
    }
</script>
```

Рисунок 3.20 – Лістинг програми: окрема сторінка допису

У процесі розробки клієнтської частини до всіх елементів вебзастосунку було застосовано стилі за допомогою CSS (Cascading Style Sheets), що забезпечило привабливе й зручне оформлення інтерфейсу. Створено єдину кольорову палітру, підібрано відповідні шрифти, визначено відступи, вирівнювання та інші параметри для підтримки цілісного стилю на всіх сторінках. Для адаптації сайту під різні пристрої й розміри екранів використано медіа-запити, що дозволяє коректно відображати контент як на комп’ютерах, так і на мобільних пристроях. Окрему увагу приділено стилізації елементів взаємодії з користувачем — таких як кнопки, форми введення, меню та фільтри — щоб забезпечити інтуїтивну навігацію й комфортне користування сайтом.

```css
.header-name {
    font-weight: 600;
    font-size: 21px;
    top: 20px;
    left: 20px;
    position: absolute;
    color: white;
}

.left-sidebar {
    background: whitesmoke;
    width: 360px;
    border-right: 1px solid #d5d5d5;
    font-size: 18px;
    min-height: calc(100vh - 60px);;
}

.search-form {
    width: fit-content;
    position: absolute;
    display: flex;
    justify-content: center;
    align-items: center;
    margin: 12px 0 0 165px;
}
```

Рисисунок 3.21 – Лістинг програми: приклад CSS коду в проекті

Отже, фронтенд-частина вебзастосунку була розроблена з урахуванням зручності, доступності та візуальної привабливості для користувача. Кожна сторінка має чітку структуру та логіку взаємодії, що сприяє легкому орієнтуванню у функціоналі. Завдяки використанню HTML та CSS вдалося створити сучасний адаптивний інтерфейс, який коректно відображається на різних пристроях. Додавання голосового озвучення тексту за допомогою JavaScript додатково підвищило доступність ресурсу. Усі ці рішення спрямовані на покращення досвіду користувача та ефективне представлення контенту.

3.4 Перспективи на вдосконалення веб-додатку

Розробка веб-додатку для ведення блогу – це лише перший крок у довготривалому процесі його розвитку та вдосконаленні. Враховуючи сучасні тенденції та технологічні інновації, є кілька ключових напрямків, які можуть значно підвищити конкурентоспроможність та популярність веб-додатку.

1. Покращення інтерфейсу користувача (UI) та користувацького досвіду (UX):

- Пристосовчий дизайн: забезпечення оптимального відображення контенту на різних пристроях, включаючи мобільні телефони, планшети та настільні комп’ютери. Це включає в себе адаптивний дизайн, швидке завантаження сторінок та інтуїтивно зрозумілу навігацію.
- Персоналізація: додавання функцій, які дозволяють користувачам налаштовувати інтерфейс під свої потреби. Це може включати зміну теми, шрифтів, кольорів та макетів.
- Покращення навігації: спрощення доступу до різних розділів блогу, зручна структура категорій та тегів, функціональний пошук по сайту.

2. Розширення функціональності:

- Підтримка мультимедіа: інтеграція можливостей для додавання різних видів контенту, таких як відео, аудіо, слайд-шоу та інтерактивні елементи. Це зробить контент більш привабливим та різноманітним.
- Інтеграція з соціальними мережами: автоматичний постинг нових записів у соціальні мережі, можливість коментування через акаунти в соціальних мережах та спрощення процесу реєстрації через соціальні платформи.
- Функціональність електронної комерції: додавання можливостей для монетизації блогу через продаж товарів, послуг або підписок.

3. Підвищення продуктивності та безпеки:

- Оптимізація швидкості завантаження: використання кешування, оптимізація зображень, мінімізація запитів до серверу та використання Content Delivery Networks (CDN) для підвищення швидкості завантаження сторінок.
- Покращення безпеки: реалізація захисту від атак, таких як SQL-ін’єкції, XSS та CSRF. Використання SSL для захисту даних користувачів, регулярне оновлення програмного забезпечення та проведення аудиту безпеки.
- Надійність та резервне копіювання: забезпечення стабільної роботи додатку, регулярне резервне копіювання даних та швидке відновлення після можливих збоїв.

4. Використання аналітики та штучного інтелекту:

- Аналітика користувацької поведінки: використання інструментів аналітики для збору даних про поведінку користувачів, що допоможе виявити слабкі місця у функціоналі та покращити користувацький досвід.
- Рекомендаційні системи: впровадження алгоритмів машинного навчання для надання персоналізованих рекомендацій щодо контенту, що може підвищити залученість користувачів.
- Автоматизація модерації контенту: використання штучного інтелекту для автоматизації модерації коментарів та виявлення неналежного контенту.

ВИСНОВКИ

У процесі виконання дипломної роботи на тему "Розробка вебдодатку для блог-платформи" було виконано ряд досліджень і реалізовано ключові аспекти створення ефективного веб-додатку. Враховуючи завдання, що були поставлені на початку роботи, досягнуто наступних висновків:

1. Аналіз функціональних вимог та потреб користувачів: Під час роботи було з'ясовано, якими функціями повинна володіти сучасна блог-платформа, а також важливість різних аспектів для користувачів. Це включає зручність створення та редагування постів, можливість коментування та поширення контенту, а також інтерактивність і безпеку. Завдяки цьому вдалося сформувати чітке уявлення про необхідні функції та побудувати їх у веб-додатку.

2. Вибір технологій: У результаті вивчення сучасних фреймворків і бібліотек для створення веб-додатків було здійснено порівняння різних технологій. Для цього проекту було обрано стек Python (Flask), SQLite для бази даних, Jinja2 для рендерингу шаблонів на фронтенді. Ці технології забезпечують гнучкість, швидкість розробки, ефективну взаємодію між компонентами і масштабованість.

3. Проектування інтерфейсу: Був розроблений інтерфейс, орієнтуючись на зручність і простоту використання. Зважаючи на вимоги користувачів, інтерфейс має бути інтуїтивно зрозумілим, адаптивним і забезпечувати безпроблемний доступ до основних функцій платформи. Було використано сучасний дизайн, який дозволяє користувачам зручно створювати, редагувати пости, а також взаємодіяти з іншими користувачами через коментарі та поширення контенту.

4. Реалізація основних функцій: У процесі розробки було реалізовано ключові функції блог-платформи: створення, редагування постів, коментування, завантаження зображень, сортування і пошук контенту. Ці функції відповідають вимогам користувачів і забезпечують зручність використання платформи. Для безпеки користувачів була реалізована система автентифікації та авторизації.

5. Проектування бази даних: Структура бази даних була спроектована таким чином, щоб забезпечити зручну роботу з постами, тегами, категоріями,

користувачами та їх взаємодіями. Це дозволило організувати дані зручно і ефективно для подальшої роботи та масштабування системи.

6. Забезпечення безпеки: Були впроваджені важливі заходи для забезпечення безпеки веб-додатку. Це включає використання захищених протоколів зв'язку, захист конфіденційної інформації та надійну систему автентифікації та авторизації, що забезпечує захист особистих даних користувачів і контенту.

У підсумку, завдання, поставлені на початку дипломної роботи, були успішно виконані. Створений веб-додаток відповідає вимогам до сучасної блог-платформи: він має всі необхідні функції для зручної взаємодії користувачів, захист даних та зручний інтерфейс. У майбутньому рекомендується продовжувати вдосконалення додатку, впроваджуючи нові технології та функціональність для покращення користувацького досвіду та забезпечення ще більш високого рівня безпеки і зручності. Реалізація можливостей, зазначених у перспективах покращення, допоможе зробити продукт ще більш привабливим і функціональним для сучасної аудиторії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Васильєв О.М. Електронна книга Програмування мовою Python. Тернопіль, 2019. 504 с.
2. В. Манако, Д.Манако, О.Данилова, О.Войченко. Основи будування сайтів. Київ: Шкільний світ, 2006. 120 с.
3. Eric Matthes. *Python Crash Course*. 3rd Edition. San Francisco: No Starch Press, 2023. 552 с.
4. Wezom. Як створити веб-додаток // 2025. URL: https://wezom.com.ua/ua/blog/kak-sozdat-veb-prilozhenie
5. AVADA MEDIA. WEB-додатки і Системи // 2025. URL: https://avada-media.ua/ua/services/web-systems/
6. CSS.in.ua. Український веб-довідник // 2025. URL: https://css.in.ua/
7. DAN.IT. Розробка з боку Front end – що це таке та чим відрізняється від Back end? // 2025. URL: https://dan-it.com.ua/blog/razrabotka-so-storony-front-end-chto-jeto-takoe-i-chem-otlichaetsja-ot-back-end/
8. Grinberg M. *Flask Web Development: Developing Web Applications with Python*, 1st Edition. Sebastopol: O'Reilly Media, 2014. 258 с.
9. Grinberg M. *Flask Web Development: Developing Web Applications with Python*, 2nd Edition. Sebastopol: O'Reilly Media, 2018. 284 с.
10. W3Schools. HTML Introduction // 2025. URL: https://www.w3schools.com/html/html_intro.asp
11. W3Schools UA. CSS Медіа запити// 2025. URL: https://w3schoolsua.github.io/css/css3_mediaqueries_ex.html#gsc.tab=0
12. W3Schools. CSS Reference // 2025. URL: https://www.w3schools.com/cssref/index.php
13. Flask. Flask documentation // 2025. URL: https://flask.palletsprojects.com/en/3.0.x/
14. Python Software Foundation. Python 3.11.3 documentation // 2025. URL: https://docs.python.org/3/

15. Acode. Learn How to Create a Website Using Python // 2025. URL: https://acode.com.ua/object-oriented-programming-python/
16. SQLite. SQLite documentation // 2025. URL: https://www.sqlite.org/docs.html
17. W3Schools. Python Tutorial // 2025. URL: https://www.w3schools.com/python/default.asp
18. Jinja. Jinja2 documentation // 2025. URL: https://jinja.palletsprojects.com/en/3.1.x/
19. Olatunde Sanni. How to build a web application using Flask and deploy it to the cloud // 2021. URL: https://www.freecodecamp.org/news/how-to-build-a-web-application-using-flask-and-deploy-it-to-the-cloud-3551c985e492/
20. DigitalOcean. How to Use Flask-SQLAlchemy // 2025. URL: https://www.digitalocean.com/community/tutorials/how-to-use-flask-sqlalchemy-to-interact-with-databases-in-a-flask-application

ДОДАТКИ

ДОДАТОК А

Огляд сайту

Головна сторінка блогового веб-додатка виконує роль центрального місця, де користувачі можуть переглядати нові пости, сортувати їх, шукати контент за ключовими словами, переглядати популярні теги, а також входити в систему або виходити з неї. Давайте детальніше розглянемо кожен з цих компонентів.

Основним елементом головної сторінки є стрічка постів. Тут відображаються всі публікації, створені користувачами, у хронологічному порядку або відповідно до іншого критерію сортування. Кожен пост містить:

- Заголовок: відображає основну тему або назву посту.
- Короткий опис: дає користувачам уявлення про зміст посту.
- Автор: показує, хто створив пост.
- Дата публікації: вказує на те, коли пост був опублікований.
- Коментарі: показує кількість коментарів до кожного посту.

Рисунок А.1 – Головна сторінка

На головній сторінці користувачі можуть сортувати пости за кількома критеріями, що дозволяє їм легко знайти контент, який відповідає їхнім інтересам. Сортування може бути виконане за:

- Новизною: від найновіших до найстаріших.
- Кількістю коментарів: від найбільш обговорюваних до найменш.
- За замовчуванням: без особливого сортування, у хронологічному порядку.

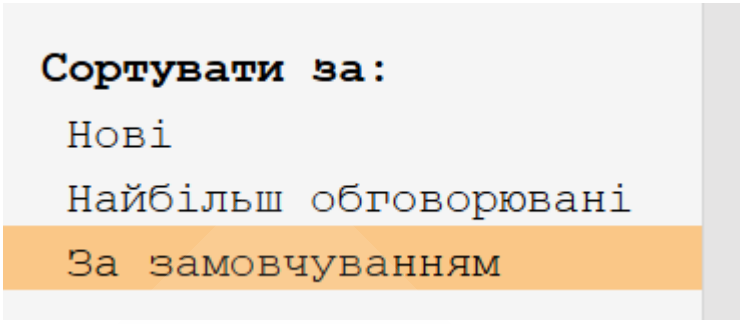

Рисунок А.2 – Сортування

На головній сторінці також є секція з популярними тегами. Користувачі можуть клацнути на тег, щоб побачити всі пости, які містять цей тег. Це допомагає знаходити контент за певними темами або категоріями.

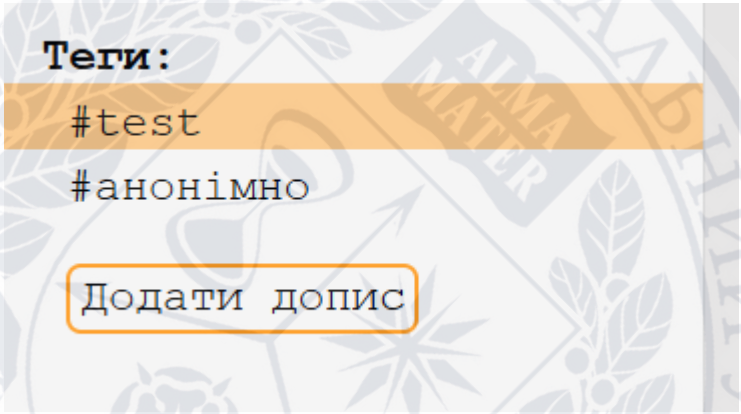

Рисунок А.3 – Теги

Для більш зручного пошуку контенту на головній сторінці є пошукова стрічка. Користувачі можуть вводити ключові слова або фрази, щоб знайти пости, які відповідають їхнім інтересам. Пошуковий механізм відображає результати, що відповідають введеним параметрам.

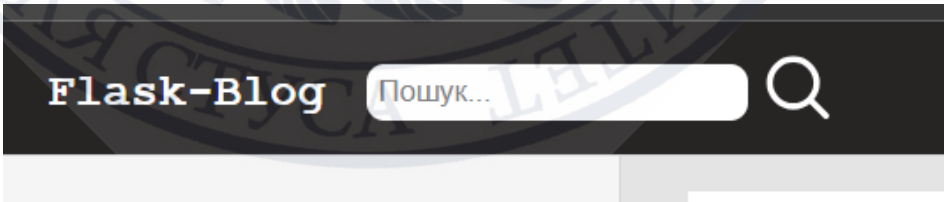

Рисунок А.4 – Пошукова стрічка

На головній сторінці є кнопка для входу в систему (login), що дозволяє користувачам увійти, використовуючи свої облікові дані.

Рисунок А.5 – Вхід в систему

Сторінка одного поста у блоговому веб-додатку — це місце, де користувачі можуть переглядати деталі певного поста, читати його повний вміст, переглядати коментарі та додавати власні.

Сторінка поста починається з заголовка, який чітко вказує на тему або зміст поста. Поруч із заголовком зазначається ім'я автора, щоб користувачі могли знати, хто написав пост.

Центральна частина сторінки, де відображається повний вміст поста. Він включає:

- **Текст**: основний текст поста.
- **Зображення**: якщо пост містить зображення, вони відображаються у відповідних місцях.

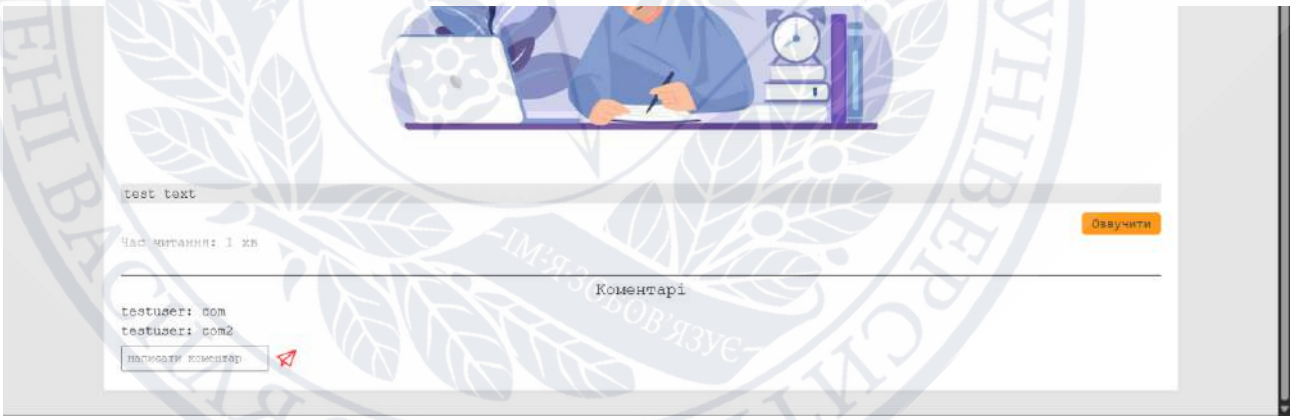

Рисунок А.6 – Сторінка одного поста

Під основним контентом поста розташована секція з коментарями. Користувачі можуть бачити всі коментарі, залишені іншими користувачами, а також додавати власні коментарі. Секція коментарів може містити такі елементи:

- Список коментарів: відображає всі коментарі до цього поста, із зазначенням імені автора та дати коментаря.

- Форма для додавання коментаря: дозволяє залогіненим користувачам додавати свої коментарі. Це може бути просте текстове поле з кнопкою для відправлення.

Рисунок А.7 – Секція коментарів

Сторінки реєстрації та логінізації у веб-додатку призначені для керування обліковими записами користувачів. Вони забезпечують можливість створення нового облікового запису (реєстрація) та входу в систему (логінізація). Ці сторінки є ключовими для додатка, який потребує аутентифікації та управління користувачами.

Сторінка реєстрації дозволяє користувачам створити новий обліковий запис, надавши необхідну інформацію. Ось основні компоненти цієї сторінки:

- Форма реєстрації:
  - Поле для імені користувача: дозволяє користувачеві вказати бажане ім'я.
  - Поле для електронної пошти: використовується для введення електронної адреси, яка буде асоційована з обліковим записом.
  - Поле для пароля: дозволяє користувачеві встановити пароль для облікового запису.
  - Поле для підтвердження пароля: для підтвердження, що введені паролі збігаються.
- Кнопка "Зареєструватися": після введення всієї необхідної інформації користувач натискає цю кнопку для завершення процесу реєстрації.
- Валідація:

- Перед відправленням даних відбувається перевірка введеної інформації. Це включає перевірку унікальності електронної пошти, збігу паролів, мінімальних вимог до паролів тощо.

- Перенаправлення після успішної реєстрації:
  - Якщо реєстрація пройшла успішно, користувач буде перенаправлений на сторінку входу.

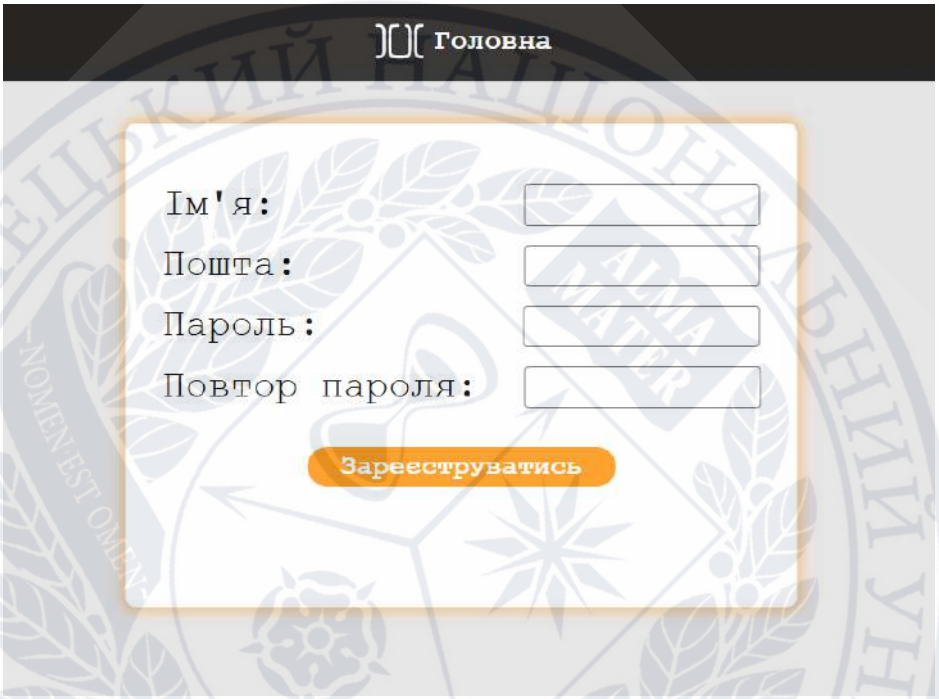

Рисунок А.8 – Сторінка реєстрації

Сторінка логінізації дозволяє зареєстрованим користувачам увійти в систему, використовуючи свою електронну пошту та пароль. Основні компоненти цієї сторінки:

- Форма логінізації:
  - Поле для електронної пошти: для введення електронної адреси, що використовується для входу.
  - Поле для пароля: для введення пароля.
- Кнопка "Увійти": після введення електронної пошти та пароля користувач натискає цю кнопку для входу в систему.
- Функція "Запам'ятати мене":

- Дозволяє користувачеві зберегти сесію для автоматичного входу при наступному відвідуванні.

- Валідація та повідомлення про помилки:
  - Перевірка електронної пошти та пароля на відповідність обліковим даним у базі даних. Якщо введено невірні дані, користувачеві показується повідомлення про помилку.
- Перенаправлення після успішного входу:
  - Якщо логінізація пройшла успішно, користувач перенаправляється на іншу сторінку, наприклад, на профіль або домашню сторінку.

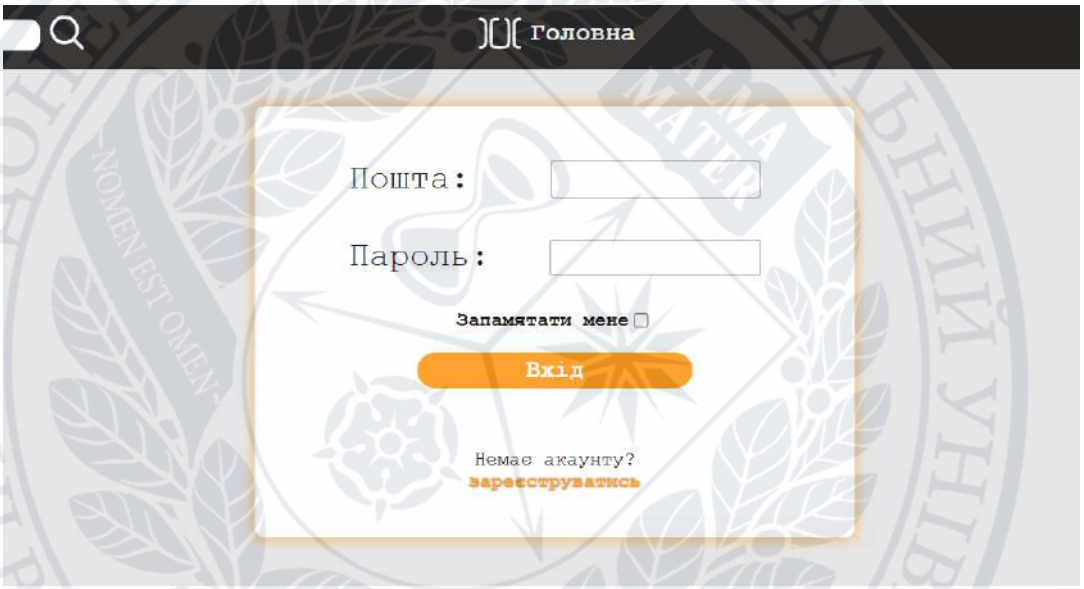

Рисунок А.9 – Сторінка логінізації

Сторінка профілю у веб-додатку є особистим простором користувача, де він може переглядати і змінювати свої дані, а також взаємодіяти з іншими елементами, такими як власні пости. На цій сторінці також є можливість виходу з облікового запису.

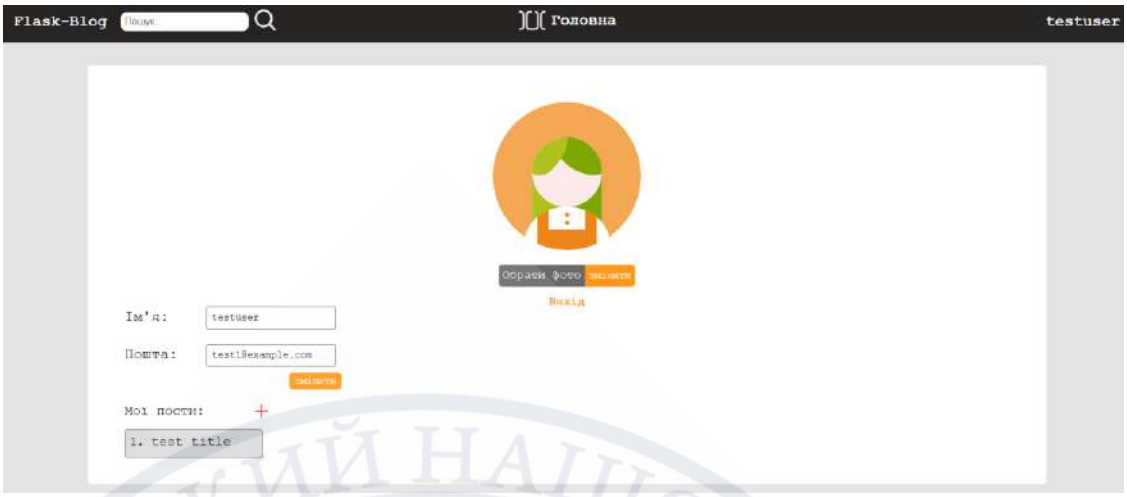

Рисунок А.10 – Сторінка профілю

Основні компоненти сторінки профілю включають:

1. На початку сторінки профілю відображається основна інформація про користувача, включаючи:

- Ім'я користувача.
- Електронна пошта.
- Фото профілю: якщо користувач має аватар, він відображається посередині. У разі відсутності аватара показується стандартне зображення.

2. Сторінка профілю дозволяє користувачу змінити свої дані. Є кнопки або посилання, що відкривають форму для редагування:

- Зміна ім'я: користувач може оновити своє ім'я, ввівши нове значення у відповідному полі.
- Зміна електронної пошти: якщо користувач хоче змінити електронну адресу, він може зробити це, вказавши нову адресу.
- Зміна фото профілю: користувач може завантажити новий аватар, вибравши файл із локального пристрою.

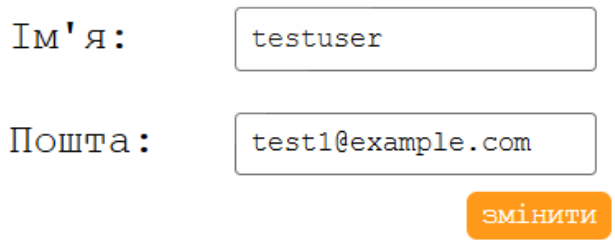

Рисунок А.11 – Зміна імені чи пошти

Рисунок А.12 – Зміна аватару

3. Сторінка профілю також містить список постів, створених користувачем. Кожен пост відображається з коротким описом. Користувач може натиснути на пост, щоб переглянути його деталі або відредагувати, якщо необхідно.

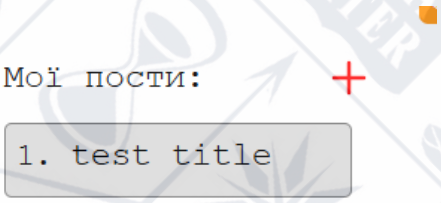

Рисунок А.13 – Перелік постів

4. Також є кнопка "logout" (вийти), яка дозволяє користувачу вийти з облікового запису. Натиснувши цю кнопку, користувач завершить свою сесію і повернеться на сторінку входу.

Рисунок А.14 – Вихід

Сторінка редагування поста у блоговому веб-додатку дозволяє автору поста внести зміни в існуючий контент. Це корисний інструмент для оновлення інформації, виправлення помилок або додавання нового контенту.

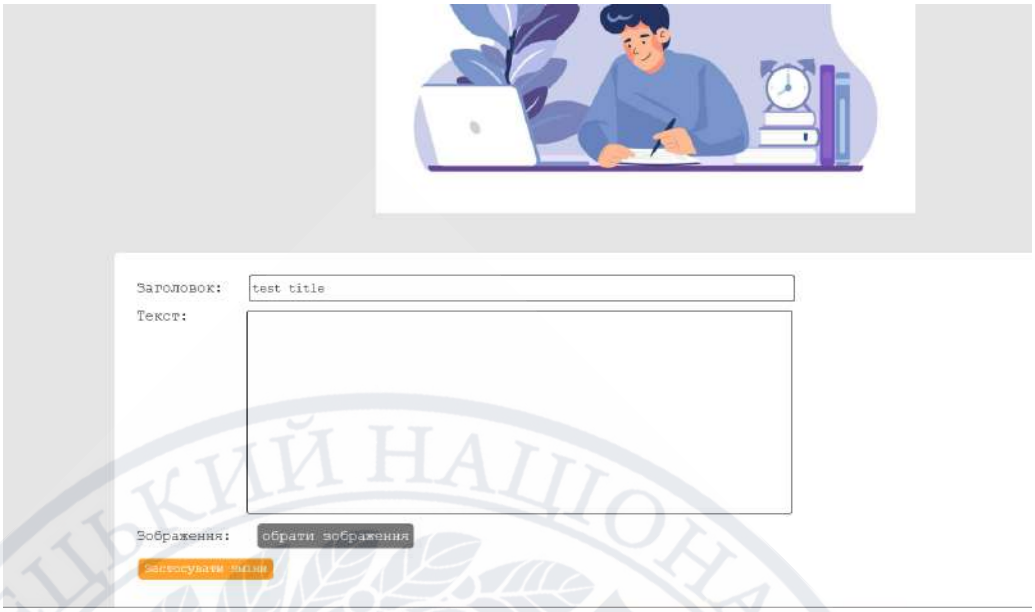

Рисунок А.15 – Форма для редагування поста

ДОДАТОК Б

Код програми

Основні компоненти коду з головного файлу:

- Ініціалізація Flask-додатка: Визначаються конфігураційні параметри, такі як база даних, секретний ключ і налаштування режиму налагодження. Flask-Login використовується для керування сесіями користувачів.
- Підключення до бази даних: Визначаються функції для підключення до SQLite-бази даних, створення бази та закриття з'єднання. Клас FDataBase використовується для взаємодії з базою даних.
- Маршрути для роботи з постами:
  - Головна сторінка зі стрічкою постів, можливістю сортування, тегами та пошуковою стрічкою.
  - Сторінка для створення нових постів, редагування та додавання коментарів.
  - Сторінка для перегляду окремих постів і їх коментарів.
- Маршрути для роботи з користувачами:
  - Сторінка логінізації з формою для входу в систему.
  - Сторінка реєстрації для створення нового облікового запису.
  - Сторінка профілю, де користувач може змінити свої дані, додати аватар, переглянути свої пости та вийти з системи.
- Маршрути для зміни особистої інформації: Користувачі можуть змінити своє ім'я, електронну пошту, аватар та інші особисті дані.
- Додаткові функції:
  - Завантаження та обробка зображень (аватарів і зображень постів).
  - Валідація даних, сповіщення про успіх або помилки.
  - Механізми авторизації для захисту певних маршрутів.
- Запуск додатка: Додаток запускається у режимі налагодження на порту 8080.

```
from flask import Flask, render_template, request, redirect, url_for, g, flash,
abort, make_response
from flask_sqlalchemy import SQLAlchemy
```

```
from FDataBase import FDataBase
import os
import re
import sqlite3
from flask_login import LoginManager, login_user, login_required, logout_user,
current_user
from UserLogin import UserLogin
from datetime import datetime
from werkzeug.security import generate_password_hash, check_password_hash
from forms import LoginForm, RegisterForm
import math


DATABASE = '/tmp/flblog.db'
DEBUG = True
SECRET_KEY = 'c289e5ace89efcecbb6be9f17eaa4045ad3591b8'
MAX_CONTENT_LENGTH = 1024*1024


app = Flask(__name__)
app.config.from_object(__name__)

app.config.update(dict(DATABASE=os.path.join(app.root_path, 'flblog.db')))


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

@login_manager.user_loader
def load_user(user_id):
    return UserLogin().fromDB(user_id, dbase)

def connect_db():
    conn = sqlite3.connect(app.config['DATABASE'])
    conn.row_factory = sqlite3.Row
    return conn

def create_db():
    db = connect_db()
    with app.open_resource('sql_db.sql', mode='r') as f:
        db.cursor().execute(f.read())
    db.commit()
    db.close()

def get_db():
    if not hasattr(g, 'link_db'):
        g.link_db = connect_db()
    return g.link_db

dbase = None
@app.before_request
```

```
def before_request():
    global dbase
    db = get_db()
    dbase = FDataBase(db)

@app.teardown_appcontext
def close_db(error):
    if hasattr(g, 'link_db'):
        g.link_db.close()



def get_date(sort):
    get_post = dbase.getPostsAnonce(sort)
    print(get_post)
    posts = []
    for p in get_post:
        elem = dict(p)
        date_object = datetime.strptime(elem['date'], "%Y-%m-%d")
        formatted_date = date_object.strftime("%a %b %d %Y")
        elem['date'] = formatted_date
        posts.append(elem)
    return posts


@app.route('/')
def index():
    return render_template('index.html', posts = get_date({'sort' : 'default'}),
tags = dbase.getTags(),
                           title = 'Main', select_sort = 'default',
                           username = current_user.getName() if
current_user.is_authenticated else None)


@app.route('/sort_by/<string:sort>')
def sort_by(sort):
    return render_template('index.html', posts = get_date({'sort' : sort}), tags =
dbase.getTags(),
                           title = 'Main', select_sort=sort,
                           username = current_user.getName() if
current_user.is_authenticated else None
)


@app.route('/tags/<string:tag>')
def sort_by_tags(tag):
    return render_template('index.html', posts = get_date({'tag' : tag}), tags =
dbase.getTags(),
                           title = 'Main', select_tags=tag, select_sort =
'default',
```

```
                            username = current_user.getName() if
current_user.is_authenticated else None
)

@app.route('/search', methods = ['POST', 'GET'])
def search():
    posts = None
    if request.method == 'POST':
        get_post = dbase.getPostsAnoncSearch(request.form['title_search'])
        posts = []
        for p in get_post:
            elem = dict(p)
            date_object = datetime.strptime(elem['date'], "%Y-%m-%d")
            formatted_date = date_object.strftime("%a %b %d %Y")
            elem['date'] = formatted_date
            posts.append(elem)
    return render_template('index.html', posts = posts, tags = dbase.getTags(),
                           title = 'Main', select_sort = 'default',
                           username = current_user.getName() if
current_user.is_authenticated else None
)



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

    form = LoginForm()
    if form.validate_on_submit():
        user = dbase.getUserByEmail(form.email.data)
        if user and check_password_hash(user['psw'], form.psw.data):
            userlogin = UserLogin().create(user)
            rm = form.remember.data
            login_user(userlogin, remember=rm)
            return redirect(request.args.get('next') or url_for('profile'))

    return render_template('login.html', menu=dbase.getMenu(), form=form,
                           username = current_user.getName() if
current_user.is_authenticated else None
)


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

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

@app.route('/register', methods = ['POST', 'GET'])
def register():
    form = RegisterForm()
    if form.validate_on_submit():
        hash_psw = generate_password_hash(form.psw.data)
        res = dbase.addUser(form.name.data, form.email.data, hash_psw,
datetime.now().date())
        if res:
            return redirect(url_for('login'))
        return render_template('register.html', menu=dbase.getMenu(), form=form,
                               username = current_user.getName() if
current_user.is_authenticated else None
)


@app.route('/profile')
@login_required
def profile():
    my_posts = dbase.getMyPosts(current_user.get_id())
    return render_template('profile.html', menu=dbase.getMenu(), my_posts=my_posts,
                           username = current_user.getName() if
current_user.is_authenticated else None
)




@app.route('/change_info', methods = ['POST', 'GET'])
@login_required
def change_info():
    if request.method == 'POST':
        res = dbase.updateUserInfo(current_user.get_id(), request.form['name'],
request.form['email'])
        if res:
            flash('success')
        else:
            flash('error')
    return redirect(url_for('profile'))

@app.route('/userava')
def userava():
    img = current_user.getAvatar(app)
    if not img:
        return ''
    h = make_response(img)
    h.headers['Content-Type'] = 'image/png'
```

```
    return h

@app.route('/post_img/<int:post_id>')
def post_img(post_id):
    img_data = dbase.getImg(post_id)
    if not img_data:
        return ''
    response = make_response(img_data)
    response.headers['Content-Type'] = 'image/png'
    return response

@app.route('/post_userava/<int:user_id>')
def post_userava(user_id):
    img_data = dbase.getAva(user_id)
    if not img_data:
        return ''
    response = make_response(img_data)
    response.headers['Content-Type'] = 'image/png'
    return response

@app.route('/edit_post<int:post_id>', methods=['POST', 'GET'])
def edit_post(post_id):
    if request.method == 'POST':
        file = request.files['file_post']
        if file:
            if current_user.verifyExt(file.filename):
                try:
                    img = file.read()
                except FileNotFoundError as e:
                    print(e)
            else:
                flash('error, MUST BE PNG')
                return redirect(url_for('edit_post'))
        else:
            img = None
        upd = dbase.updatePostInfo(post_id, request.form['title'],
request.form['content'], img)
        if upd:
            flash('success')
        else:
            flash('error')

    res = dbase.getPost(post_id)
    return render_template('edit_post.html', menu=dbase.getMenu(), post=res,
                           username = current_user.getName() if
current_user.is_authenticated else None
)
```

```python
@app.route('/upload', methods = ['POST', 'GET'])
def upload():
    if request.method == "POST":

        file = request.files['file']
        if file and current_user.verifyExt(file.filename):
            try:
                img = file.read()
                res = dbase.updateUserAvatar(img, current_user.get_id())
                if res:
                    flash('update')
                else:
                    flash('error')

            except FileNotFoundError as e:
                print(e)
        else:
            print('error55')
    return redirect(url_for('profile'))



@app.route('/add_post', methods=['POST', 'GET'])
def add_post():
    if request.method == 'POST':
        file = request.files.get('file_post')
        img = None
        if file and file.filename:
            if current_user.is_authenticated:
                is_valid = current_user.verifyExt(file.filename)
            else:
                is_valid = file.filename.lower().endswith('.png')

            if is_valid:
                try:
                    img = file.read()
                except FileNotFoundError as e:
                    print(e)
            else:
                flash('error, MUST BE PNG')
                return redirect(url_for('add_post'))


        tags = re.findall(r'\w+', request.form['tags'])

        if current_user.is_authenticated:
            user_id = current_user.get_id()
        else:
```

```
            user_id = 11

        res = dbase.addPost(user_id, request.form['name'], request.form['text'],
tags, img, datetime.now().date())

        if not res:
            flash('error')
        else:
            flash('success')

    return render_template('add_post.html', menu=dbase.getMenu(),
                           username = current_user.getName() if
current_user.is_authenticated else None
)



@app.route('/add_comments/<int:post_id>', methods = ['POST', 'GET'])
@login_required
def add_comments(post_id):
    if request.method == 'POST':
        res = dbase.addComment(current_user.get_id(), post_id,
request.form['comment'], datetime.now().date())
        if res:
            flash('success')
        else:
            flash('error')


    return redirect(url_for('show_post', post_id=post_id))



def estimate_read_time(text):
    words = text.split()
    word_count = len(words)
    read_time_minutes = math.ceil(word_count / 200)
    return read_time_minutes



@app.route('/post/<int:post_id>')
@login_required
def show_post(post_id):

    res = dbase.getPost(post_id)
    comments = dbase.getComments(post_id)

    read_time = estimate_read_time(res['post_content'])
```

```
    return render_template('post.html', menu = dbase.getMenu(), post_id=post_id,
                           res=res, coms=comments, read_time = read_time,
                           username = current_user.getName() if
current_user.is_authenticated else None
)


if __name__ == '__main__':
    app.run(debug=True, port=5000)
```

Клас, UserLogin, використовується для управління інформацією про користувачів у Flask-додатку, який використовує Flask-Login для керування сесіями користувачів та автентифікацією. Клас містить кілька методів, що дозволяють працювати з даними користувачів, отримувати їхню інформацію, а також здійснювати деякі перевірки.

- fromDB(self, user_id, db): цей метод отримує дані користувача з бази даних за вказаним ідентифікатором (user_id). Він встановлює ці дані як приватний атрибут класу (self.__user) і повертає поточний екземпляр об'єкта.
- create(self, user): цей метод приймає об'єкт користувача і встановлює його як приватний атрибут, що дозволяє ініціалізувати об'єкт UserLogin з даними користувача.
- get_id(self): цей метод повертає ідентифікатор користувача як рядок, що використовується для ідентифікації користувача в сесіях.
- getName(self): повертає ім'я користувача. Якщо немає даних користувача, повертає "none".
- getEmail(self): повертає електронну адресу користувача. Якщо немає даних користувача, повертає "none".
- getAvatar(self, app): повертає аватар користувача. Якщо у користувача немає встановленого аватара, завантажує стандартний аватар з файлу за замовчуванням.

- verifyExt(self, filename): перевіряє розширення файлу на допустимість (наприклад, дозволені розширення: png, PNG, jpeg). Якщо розширення відповідає вимогам, повертає True, інакше — False.

Загалом, цей клас призначений для забезпечення управління інформацією про користувачів, а також для роботи з автентифікацією та управлінням сесіями у Flask-додатках, що використовують Flask-Login.

```
from flask_login import UserMixin
from flask import url_for

class UserLogin(UserMixin):
    def fromDB(self, user_id, db):
        self.__user = db.getUser(user_id)
        return self

    def create(self, user):
        self.__user = user
        return self

    def get_id(self):
        return str(self.__user['user_id'])

    def getName(self):
        return self.__user['username'] if self.__user else 'none'

    def getEmail(self):
        return self.__user['email'] if self.__user else 'none'

    def getAvatar(self, app):
        img = None
        if not self.__user['avatar']:
            try:
                with app.open_resource(app.root_path + url_for('static',
filename='imgs/avatar-default-svgrepo-com.png'), "rb") as f:
                    img = f.read()
            except FileNotFoundError as e:
                print(e)
        else:
            img = self.__user['avatar']

        return img


    def verifyExt(self, filename):
        ext = filename.rsplit('.', 1)[1]
        if ext == 'png' or ext == 'PNG' or ext == 'jpeg':
            return True
```

```
        return False
```

Ключові функції та методи класу FDataBase:

Ініціалізація

- Клас ініціалізується об'єктом бази даних та курсором для виконання SQL-запитів.

Методи для отримання даних

- getMenu(): отримує дані з таблиці меню.
- getPost(post_id): отримує деталі конкретного поста за його ідентифікатором.
- getPostsAnonce(sort): отримує анонси постів з можливістю сортування.
- getPostsAnoncSearch(title): виконує пошук постів за заголовком.
- getComments(post): отримує коментарі для певного поста.
- getUser(user_id) та getUserByEmail(email): отримує дані користувача за його ID або електронною поштою.
- getImg(id) та getAva(id): отримує зображення поста або аватар користувача.
- getMyPosts(user_id): отримує всі пости, створені певним користувачем.
- getTags(): отримує список тегів.

Методи для додавання даних

- addPost(user_id, title, text, tags, img, date): додає новий пост з певними даними.
- addComment(user_id, post_id, text, date): додає коментар до певного поста.
- addUser(username, email, psw, date): додає нового користувача до бази даних.

Методи для оновлення даних

- updateUserAvatar(avatar, user_id): оновлює аватар користувача.
- updateUserInfo(user_id, name, email): оновлює ім'я та електронну пошту користувача.

- updatePostInfo(post_id, title, content, img): оновлює дані поста, з можливістю оновлення зображення.

```
import sqlite3

class FDataBase:
    def __init__(self, db) -> None:
        self.__db = db
        self.__cur = db.cursor()


    def getMenu(self):
        sql = 'select * from mainmenu'
        try:
            self.__cur.execute(sql)
            res = self.__cur.fetchall()
            if res: return res
        except:
            print('error')
        return []



    def addPost(self, user_id, title, text, tags, img, date):
        try:
            binary = sqlite3.Binary(img) if img else None
            self.__cur.execute(
                'INSERT INTO Posts (user_id, title, post_content, tags, post_img,
date_posted) '
                'VALUES (?, ?, ?, ?, ?, ?)',
                (user_id, title, text, ' '.join(tags), binary, date)
            )

            for t in tags:
                self.__cur.execute('SELECT COUNT(*) AS "count" FROM Tags WHERE
tag_text = ?', (t,))
                res = self.__cur.fetchone()
                if res['count'] > 0:
                    print(f'{t} already exists')
                else:
                    self.__cur.execute('INSERT INTO Tags (tag_text) VALUES (?)',
(t,))

            self.__db.commit()
        except sqlite3.Error as e:
            print(e)
            return False

        return True
```

```
    def addComment(self, user_id, post_id, text, date):
        try:
            self.__cur.execute('insert into Comments values(null, ?, ?, ?, ?)',
(user_id, post_id, text, date))
            self.__db.commit()
        except sqlite3.Error as e:
            print(e)
            return False

        return True

    def addUser(self, username, email, psw, date):
        try:
            self.__cur.execute(f'select count() as "count" from Users where email
like "{email}"')
            res = self.__cur.fetchone()
            if res['count'] > 0:
                print('error2')
                return False
            # print('error44')

            self.__cur.execute('insert into Users values (null, ?, ?, ?, null, ?)',
(username, email, psw, date))
            self.__db.commit()
        except sqlite3.Error as e:
            print(e)
            return False

        return True



    def getPost(self, post_id):
        try:
            self.__cur.execute(f'''select Posts.post_id as "post_id",
                                          Posts.user_id as "user_id",
                                          Posts.title as "title",
                                          Posts.post_content as "post_content",
                                          Posts.tags as "tags",
                                          Posts.post_img as "post_img",
                                          Posts.date_posted as "date_posted",
                                          Users.username as "username"
                                   from Posts
                                   INNER JOIN
                                       Users ON Users.user_id = Posts.user_id
```

```
                                where post_id = {post_id} limit 1''')
                res = self.__cur.fetchone()
                if res:
                    return res
            except sqlite3.Error as e:
                print(e)

            return []

        def getPostsAnonce(self, sort):
            try:
                query = '''SELECT
                                Users.username AS "username",
                                Users.user_id AS "user_id",
                                Posts.post_id AS "post_id",
                                Posts.title AS "title",
                                Posts.post_content AS "content",
                                Posts.tags as 'tags',
                                Posts.post_img AS "img",
                                Posts.date_posted AS "date",
                                COUNT(Comments.comment_id) AS "comment_count"
                            FROM
                                Posts
                            INNER JOIN
                                Users ON Users.user_id = Posts.user_id
                            LEFT JOIN
                                Comments ON Comments.post_id = Posts.post_id
                            GROUP BY
                                Users.username, Posts.post_id, Posts.title,
Posts.post_content, Posts.post_img
                            '''
                if next(iter(sort.keys()), None) == 'sort':
                    if sort['sort'] == 'new':
                        self.__cur.execute(query + 'ORDER BY Posts.date_posted DESC;')
                    elif sort['sort'] == 'most_discussed':
                        self.__cur.execute(query + 'ORDER BY COUNT(Comments.comment_id)
DESC;')
                    elif sort['sort'] == "default":
                        self.__cur.execute(query + ';')
                elif next(iter(sort.keys()), None) == 'tag':
                    tag = sort['tag']
                    self.__cur.execute(f'''SELECT
                                Users.username AS "username",
                                Users.user_id AS "user_id",
                                Posts.post_id AS "post_id",
                                Posts.title AS "title",
                                Posts.post_content AS "content",
                                Posts.tags as 'tags',
                                Posts.post_img AS "img",
                                Posts.date_posted AS date,
```

```
                        COUNT(Comments.comment_id) AS "comment_count"
                    FROM
                        Posts
                    INNER JOIN
                        Users ON Users.user_id = Posts.user_id
                    LEFT JOIN
                        Comments ON Comments.post_id = Posts.post_id
                    WHERE
                        Posts.tags LIKE "%{tag}%"
                    GROUP BY
                        Users.username, Posts.post_id, Posts.title,
Posts.post_content, Posts.post_img
                    ''')
            else:
                self.__cur.execute(query + ';')
            res = self.__cur.fetchall()
            if res:
                return res
        except sqlite3.Error as e:
            print(e)

        return []

    def getPostsAnoncSearch(self, title):
        try:
            self.__cur.execute(f'''SELECT
                                Users.username AS "username",
                                Users.user_id AS "user_id",
                                Posts.post_id AS "post_id",
                                Posts.title AS "title",
                                Posts.post_content AS "content",
                                Posts.tags as 'tags',
                                Posts.post_img AS "img",
                                Posts.date_posted AS date,
                                COUNT(Comments.comment_id) AS "comment_count"
                            FROM
                                Posts
                            INNER JOIN
                                Users ON Users.user_id = Posts.user_id
                            LEFT JOIN
                                Comments ON Comments.post_id = Posts.post_id
                            WHERE
                                Posts.title LIKE "%{title}%"
                            GROUP BY
                                Users.username, Posts.post_id, Posts.title,
Posts.post_content, Posts.post_img
                            ''')
            res = self.__cur.fetchall()
            if res:
                return res
```

```
        except sqlite3.Error as e:
            print(e)

        return []

    def getComments(self, post):
        try:
            self.__cur.execute(f'''select Users.username as "user",
Comments.comment_content as "comment_content"
                                  from Comments inner join Users on Users.user_id =
Comments.user_id
                                  where post_id = {post}''')
            res = self.__cur.fetchall()
            if res: return res
        except sqlite3.Error as e:
            print(e)

        return []

    def getUser(self, user_id):
        try:
            self.__cur.execute(f'select * from Users where user_id = {user_id}
limit 1')
            res = self.__cur.fetchone()
            if res: return res

        except sqlite3.Error as e:
            print(e)

        return []

    def getUserByEmail(self, email):
        try:
            self.__cur.execute(f'select * from Users where email = "{email}" limit
1')
            res = self.__cur.fetchone()
            if res: return res

        except sqlite3.Error as e:
            print(e)

        return []

    def getImg(self, id):
        img_data = None
        try:
            self.__cur.execute(f'select post_img from Posts where post_id = {id}
limit 1')
            img = self.__cur.fetchone()
            if img:
```

```
                img_data = img['post_img']
        except sqlite3.Error as e:
            print(e)
        return img_data

    def getAva(self, id):
        img_data = None
        try:
            self.__cur.execute(f'select avatar from Users where user_id = {id}
limit 1')
            img = self.__cur.fetchone()
            if img:
                img_data = img['avatar']
        except sqlite3.Error as e:
            print(e)
        return img_data

    def getMyPosts(self, user_id):
        try:
            self.__cur.execute(f'select * from Posts where user_id = {user_id}')
            res = self.__cur.fetchall()
            if res: return res

        except sqlite3.Error as e:
            print(e)

        return []

    def getTags(self):
        try:
            self.__cur.execute(f'select * from Tags ORDER BY tag_text ASC')
            res = self.__cur.fetchall()
            if res: return res

        except sqlite3.Error as e:
            print(e)

        return []



    def updateUserAvatar(self, avatar, user_id):
        if not avatar:
            return False

        try:
            binary = sqlite3.Binary(avatar)
            self.__cur.execute(f'update Users set avatar = ? where user_id = ?',
(binary, user_id))
```

```
            self.__db.commit()
        except sqlite3.Error as e:
            print(e)
            return False
        return True


    def updateUserInfo(self, user_id, name, email):
        try:
            self.__cur.execute(f'update Users set username = ?, email = ? where
user_id = ?', (name, email, user_id))
            self.__db.commit()
        except sqlite3.Error as e:
            print(e)
            return False
        return True

    def updatePostInfo(self, post_id, title, content, img):
        try:
            if img:
                binary = sqlite3.Binary(img)
                self.__cur.execute(f'update Posts set title = ?, post_content = ?,
post_img = ? where post_id = ?', (title, content, binary, post_id))
                self.__db.commit()
            else:
                self.__cur.execute(f'update Posts set title = ?, post_content = ?
where post_id = ?', (title, content, post_id))
                self.__db.commit()
        except sqlite3.Error as e:
            print(e)
            return False
        return True
```

Один з html файлів:

Код представляє шаблон Jinja2 для сторінки, на якій можна додати новий пост у блоговому веб-додатку. Основні компоненти:

- Флеш-повідомлення: Відображає флеш-повідомлення, якщо такі є, з умовою для повідомлень типу "message".
- Форма для додавання поста: Форма використовує метод POST і дозволяє завантажувати файли (через enctype="multipart/form-data"). Вона містить:
    - Поле для заголовка.

- Поле для основного тексту (текстове поле/textarea).
- Поле для тегів.
- Поле для вибору файлу з міткою та обмеженням для зображень.
- Кнопка "Add" для додавання поста.

Ця сторінка надає інтерфейс для додавання нового поста з усіма необхідними елементами, включаючи флеш-повідомлення, заголовок, текст, теги та можливість додавати зображення.

```
{% extends 'base.html' %}
{% block content %}
{% for msg_type, msg_text in get_flashed_messages(True) %}
    {% if msg_type == 'message' %}
        <span class="msg-flash">{{ msg_text }}</span>
    {% endif %}
{% endfor %}
    <form action="{{url_for('add_post')}}" method="post" enctype="multipart/form-
data">
        <div class="add-post-div">
            <span>Заголовок: <input class="title-input" type="text"
name="name"></span> <br>
            <span>Текст: <textarea class="text-input" type="text"
name="text"></textarea></span> <br>
            <span>Теги: <input class="tags-input" type="text" name="tags"></span>
<br>
            <label class="add-post-sel-file" for="filename">обрати файл</label>
            <span>Оберіть зображення: <input type="file" id="filename"
name="file_post" accept="image/*"></span>
            <span><input class="btn-add-post" type="submit" value="Додати
пост"></span>
        </div>
    </form>
{% endblock %}
```

ДЕКЛАРАЦІЯ

про дотримання академічної доброчесності

Я, Мельник Дар’я Сергівна

Повністю вказується ПІБ та статус (посада для працівників, освітня (освітньо-наукова) програма – для здобувачів вищої освіти)

що нижче підписалась/підписався, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

ЗОБОВ’ЯЗУЮСЬ:

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами,Мельник Даря Сергівна визначеними іншими суб’єктами, та не допускати їх порушення.

ПІДТВЕРДЖУЮ:

що мені відомі положення статті 42 Закону України «Про освіту»;

що у даній роботі не представляла/представляв чиїсь роботи повністю або частково як свої власні. Там, де я скористалася/скористався працею інших, я зробила/зробив відповідні посилання на джерела інформації;

що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

УСВІДОМЛЮЮ:

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів;

що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнута/притягнутий до академічної відповідальності.

_____ _____

(дата) (підпис)