МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет водного господарства та природокористування
Навчально-науковий інститут автоматики, кібернетики та обчислювальної техніки
Кафедра комп'ютерних технологій та економічної кібернетики

Допущено до захисту:

Завідувач кафедри

_____ д.е.н.,проф. П. М.Грицюк

«_____» _____ 2023р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня «бакалавр»
за освітньо-професійною програмою «Інформаційні системи і технології»
спеціальності 126 «Інформаційні системи та технології»

на тему: «**Сайт особистого блогу з інтеграцією штучного інтелекту**»

Виконав:

здобувач вищої освіти 4 курсу, групи ІСТ41

Мартинюк Андрій Богданович

Керівник:

канд. техн. наук, доцент Гладка О. М.

Рецензент:

канд. техн. наук, доцент Барановський С. В.

Рівне – 2024

ЗМІСТ

ВСТУП

У сучасному світі, де цифрові технології стрімко розвиваються та стають невід'ємною частиною нашого повсякденного життя, особливо актуальним стає створення ефективних та безпечних веб-платформ для блогів. Ці платформи не тільки сприяють обміну інформацією, але й стимулюють культурний обмін та освітній процес, роблячи їх значущими для соціального розвитку та особистісного зростання.

Метою даної дипломної роботи є розробка та тестування блогової платформи, інтегрованої з системами штучного інтелекту для автоматизації створення контенту. Це дозволить користувачам не тільки публікувати власні матеріали, але й ефективно використовувати передові технології для генерації текстового контенту, оптимізації управління контентом та покращення взаємодії з аудиторією.

Об'єктом дослідження виступає процес розробки веб-платформ для блогів, який включає в себе як програмну реалізацію так і тестування компонентів системи. Предметом дослідження є алгоритми штучного інтелекту для генерації контенту, механізми авторизації та забезпечення безпеки, а також методики тестування цих компонентів.

Для досягнення поставленої мети у роботі використовуються методи програмної інженерії, тестування програмного забезпечення, аналізу систем та розробки веб-додатків. Особлива увага приділяється методам аналітичного моделювання, що дозволяє оптимізувати процеси в межах розробленої системи, а також методам машинного навчання для реалізації модулів штучного інтелекту.

Практичне значення роботи полягає у створенні гнучкої та масштабованої платформи, що може бути використана різними категоріями користувачів – від окремих блогерів до великих медіа комплексів. Завдяки інтеграції штучного

інтелекту платформа значно спрощує процес створення контенту, забезпечуючи високу якість та релевантність матеріалів.

Очікувані результати включають створення повнофункціональної веб-платформи з інтуїтивно зрозумілим інтерфейсом, високим рівнем безпеки та стабільною роботою під час високих навантажень. Результати тестування підтвердять ефективність розроблених алгоритмів та готовність системи до експлуатації, що дозволить їй успішно конкурувати на ринку цифрових медіа-платформ.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття блогу

Блог (скорочення від «weblog») [1] — це інформаційний веб-сайт, що складається з окремих, часто неформальних текстових записів (дописів) у стилі щоденника. Публікації зазвичай відображаються у зворотному хронологічному порядку, щоб найновіша публікація з'являлася першою, у верхній частині веб-сторінки. У 2000-х роках блоги часто були роботою однієї людини, іноді невеликої групи, і часто висвітлювали одну тему чи тему. У 2010-х роках з'явилися «багатоавторські блоги» (MAB), у яких публікувалися статті кількох авторів, а іноді їх редагували професійно. MAB з газет, інших засобів масової інформації, університетів, аналітичних центрів, правозахисних груп і подібних установ спричиняють зростаючу кількість трафіку блогів. Розвиток Twitter та інших систем «мікроблогів» допомагає інтегрувати MAB та блоги одного автора в ЗМІ. Блог також може використовуватися як дієслово, що означає підтримувати або додавати вміст до блогу.

Поява та зростання блогів наприкінці 1990-х збіглося з появою інструментів веб-публікації, які полегшили публікацію вмісту нетехнічними користувачами, які не мали великого досвіду роботи з HTML або комп'ютерним програмуванням. Раніше знання таких технологій, як HTML і протокол передачі файлів, були необхідними для публікації вмісту в Інтернеті, тому перші користувачі Інтернету, як правило, були хакерами та комп'ютерними ентузіастами. Станом на 2010-ті роки більшість із них є інтерактивними веб-сайтами Web 2.0, що дозволяє відвідувачам залишати онлайн-коментарі, і саме ця інтерактивність відрізняє їх від інших статичних веб-сайтів.[2] У цьому сенсі ведення блогу можна розглядати як форму соціальної мережі. Дійсно, блогери не тільки створюють контент для публікації у своїх блогах, але й часто будують

соціальні стосунки зі своїми читачами та іншими блогерами.[3] Власники чи автори блогів часто модерують і фільтрують онлайн-коментарі, щоб видалити ворожі висловлювання чи інший образливий вміст. Є також блоги з великою аудиторією, які не дозволяють коментарі.

Багато блогів містять коментарі на певну тему чи тему, починаючи від філософії, релігії та мистецтва до науки, політики та спорту. Інші функціонують як більш особисті онлайн-щоденники або онлайн-реклама бренду певної особи чи компанії. Типовий блог поєднує текст, цифрові зображення та посилання на інші блоги, веб-сторінки та інші засоби масової інформації, пов'язані з його темою. Більшість блогів переважно текстові, хоча деякі зосереджуються на мистецтві (мистецькі блоги), фотографіях (фотоблоги), відео (відеоблоги або «влоги»), музиці (блоги MP3) та аудіо (подкасти). В освіті блоги можна використовувати як навчальні ресурси; їх називають edublogs. Мікроблоги — це ще один вид блогів, який містить дуже короткі повідомлення.

Терміни «блог» і «ведення блогів» зараз рідко використовуються для створення вмісту та обміну ним у соціальних мережах, особливо якщо вміст є довгостроковим і користувач створює та ділиться вмістом на регулярній основі. Отже, можна вести блог у Facebook або блог в Instagram.

За оцінками 2022 року було понад 600 мільйонів загальнодоступних блогів із понад 1,9 мільярда веб-сайтів.[4]

До того, як блоги стали популярними, цифрові спільноти набули багатьох форм, включаючи Usenet, комерційні онлайн-сервіси, такі як GEnie, Byte Information Exchange (BIX) і ранній CompuServe, списки електронної пошти [10] і системи дощок оголошень (BBS). У 1990-х роках програмне забезпечення Інтернет-форумів створювало поточні розмови з «потоками». Потоки — це тематичні зв'язки між повідомленнями на віртуальній «корковій дошці».

Бернерс-Лі також створив те, що Британська енциклопедія вважає «першим «блогом»» у 1992 році для обговорення прогресу, досягнутого у

створенні Всесвітньої павутини та програмного забезпечення, яке для цього використовувалося. [11]

З 14 червня 1993 року корпорація Mosaic Communications підтримувала свій список нових веб-сайтів «Що нового» [12], який оновлювався щодня та архівувався щомісяця. Сторінка була доступна за допомогою спеціальної кнопки «Що нового» у веб-браузері Mosaic.

У листопаді 1993 року Ранджит Бхатнаґар почав писати про цікаві сайти, сторінки та дискусійні групи, які він знайшов в Інтернеті, а також деяку особисту інформацію на своєму веб-сайті Moonmilk, упорядковуючи їх у хронологічному порядку в спеціальному розділі під назвою Ranjit's HTTP Playground.[13] Інші перші піонери блогів, такі як Джастін Холл, вважають його джерелом натхнення.[14]

Найперший екземпляр комерційного блогу був на першому веб-сайті для споживачів, створеному в 1995 році компанією Ty, Inc., який містив блог у розділі під назвою «Онлайн-щоденник». Записи підтримувалися рекомендованими Beanie Babies, за які щомісяця голосували відвідувачі веб-сайту.[15]

Сучасний блог розвинувся з онлайн-щоденника, де люди вели поточний звіт про події в особистому житті. Більшість таких письменників називали себе щоденниками, журналістами або журналістами. Джастін Холл, який почав вести особистий блог у 1994 році, будучи студентом коледжу Суортмор, загальновизнаний як один із перших блогерів [16], як і Джеррі Пурнелл [17]. Новини сценаріїв Дейва Вінера також вважаються одним із найстаріших і довготривалих веб-журналів.[18][19] Австралійський журнал Netguide підтримував Daily Net News [20] на своєму веб-сайті з 1996 року. Daily Net News запускав посилання та щоденні огляди нових веб-сайтів, переважно в Австралії.

Іншим раннім блогом був Wearable Wireless Webcam, спільний онлайн-щоденник особистого життя людини, що поєднує текст, цифрове відео та

цифрові зображення, що передаються в прямому ефірі з носимого комп'ютера та пристрою EyeTap на веб-сайт у 1994 році. Ця практика напівавтоматичного ведення блогу з живе відео разом із текстом називали sousveillance, і такі журнали також використовувалися як докази в юридичних справах. Деякі ранні блогери, як-от The Misanthropic Bitch, який почав роботу в 1997 році, фактично називали свою онлайн-присутність журналом, перш ніж термін блог увійшов у загальне вживання.

Першою дослідницькою статтею про ведення блогів була стаття Торілла Мортенсена та Джилл Уокер Реттберг «Думки щодо ведення блогів» [21], у якій аналізувалося, як блоги використовувалися для сприяння дослідницьким спільнотам, обміну ідеями та наукою, і як цей новий спосіб спілкування руйнує традиційні силові структури.

Перші блоги були просто оновленими вручну компонентами звичайних веб-сайтів. У 1995 році «Онлайн-щоденник» на веб-сайті Ty, Inc. був створений і оновлений вручну до того, як були доступні будь-які програми для ведення блогів. Дописи відображалися у зворотному хронологічному порядку шляхом ручного оновлення текстового HTML-коду за допомогою програмного забезпечення FTP у реальному часі кілька разів на день. Для користувачів це створювало вигляд щоденника, який містив кілька нових записів на день. На початку кожного нового дня нові щоденникові записи вручну кодувалися в новий HTML-файл, а на початку кожного місяця щоденникові записи архівувалися в окрему папку, яка містила окрему HTML-сторінку для кожного дня місяця. Потім меню, які містили посилання на останній запис щоденника, оновлювалися вручну на всьому сайті. Цей текстовий метод організації тисяч файлів послужив трампліном для визначення майбутніх стилів ведення блогів, які були зафіксовані програмним забезпеченням для ведення блогів, розробленим роками пізніше.[15]

Еволюція електронних і програмних інструментів для полегшення створення та підтримки веб-статей, розміщених у зворотному хронологічному порядку, зробила процес публікації можливим для значно більшої кількості людей, які мають меншу технічну підготовку. Зрештою, це призвело до окремого класу онлайн-публікацій, які створюють блоги, які ми знаємо сьогодні. Наприклад, використання певного програмного забезпечення на основі браузера тепер є типовим аспектом «ведення блогу». Блоги можна розміщувати в спеціалізованих службах хостингу блогів, на звичайних службах веб-хостингу або запускати за допомогою програмного забезпечення для блогів.

1.2 Мовні моделі

Велика мовна модель (LLM) — це обчислювальна модель, яка відома своєю здатністю створювати мову загального призначення та виконувати інші завдання обробки природної мови, наприклад класифікацію. Базуючись на мовних моделях, магістри набувають цих здібностей, вивчаючи статистичні зв'язки з величезних обсягів тексту під час обчислювально інтенсивного процесу самостійного та напівконтрольованого навчання.[1] LLM можна використовувати для генерації тексту, форми генеративного штучного інтелекту, беручи вхідний текст і багаторазово прогнозуючи наступну лексему чи слово.[2]

LLM – це штучні нейронні мережі, які використовують трансформаторну архітектуру, винайдену в 2017 році. Найбільші та найпотужніші LLM станом на червень 2024 року побудовані на базі трансформаторної архітектури лише декодера, що забезпечує ефективну обробку та генерацію великомасштабного тексту. даних.

Історично склалося так, що до 2020 року тонке налаштування було основним методом адаптації моделі для конкретних завдань. Проте більші моделі, такі як GPT-3, продемонстрували здатність досягати подібних результатів за допомогою оперативного проектування, яке передбачає створення конкретних вхідних підказок для керування відповідями моделі.[3] Ці моделі отримують знання про синтаксис, семантику та онтології[4], властиві людським мовним корпусам, але вони також успадковують неточності та упередження, присутні в даних, на яких вони навчаються.[5]

Деякі відомі LLM — це моделі серії GPT OpenAI (наприклад, GPT-3.5 і GPT-4, що використовуються в ChatGPT і Microsoft Copilot), Gemini від Google (останній з яких зараз використовується в однойменному чат-боті), сімейство LLaMA від Meta. моделей, моделі Claude від Anthropic і моделі Mistral AI.

До 2017 року існувало кілька мовних моделей, які були великими порівняно з наявними на той час можливостями. У 1990-х роках моделі вирівнювання IBM стали піонерами статистичного моделювання мови. У 2000-х роках, коли використання Інтернету стало поширеним, деякі дослідники створили мовні набори даних Інтернет-масштабу («веб як корпус» [6]), на основі яких вони навчали статистичні мовні моделі [7] [8]. У 2009 році в більшості завдань обробки мови статистичні мовні моделі домінували над моделями символьної мови, оскільки вони можуть з користю використовувати великі набори даних.[9]

Після того як нейронні мережі стали домінуючими в обробці зображень приблизно в 2012 році, їх також застосували для мовного моделювання. У 2016 році Google перевів свій сервіс перекладу на Neural Machine Translation. Як і до Трансформерів, це робили мережі seq2seq deep LSTM.

На конференції NeurIPS у 2017 році дослідники Google представили архітектуру трансформатора у своїй знаковій статті «Увага — це все, що вам потрібно». Мета цієї статті полягала в тому, щоб покращити технологію Seq2seq 2014 року [10] і базувалася в основному на механізмі уваги, розробленому Bahdanau et al. у 2014 році.[11] Наступного 2018 року було представлено BERT, яке швидко стало «повсюдним».[12] Хоча оригінальний трансформатор має блоки кодера та декодера, BERT є моделлю лише кодера.

Хоча лише декодер GPT-1 був представлений у 2018 році, саме GPT-2 у 2019 році привернув широку увагу, оскільки OpenAI спочатку вважав його надто потужним, щоб оприлюднити його публічно через побоювання зловмисного використання.[13] GPT-3 у 2020 році пішов далі і станом на 2024 рік доступний лише через API без пропозиції щодо завантаження моделі для локального виконання. Але саме браузерний ChatGPT 2022 року, орієнтований на споживачів, захопив уяву широких верств населення та викликав певний

ажіотаж у ЗМІ та онлайн-галас.[14] GPT-4 2023 року хвалили за його підвищену точність і як «святий Грааль» за його мультимодальні можливості.[15] OpenAI не виявив високорівневу архітектуру та кількість параметрів GPT-4.

Конкуруючі мовні моделі здебільшого намагалися зрівнятися з серією GPT, принаймні з точки зору кількості параметрів.[16]

Починаючи з 2022 року, моделі з доступним джерелом набирають популярності, особливо спочатку з BLOOM і LLaMA, хоча обидві мають обмеження щодо сфери використання. Моделі Mistral AI Mistral 7B і Mixtral 8x7b мають більш дозволену ліцензію Apache. Станом на червень 2024 року налаштований варіант The Instruction моделі Llama 3 із 70 мільярдами параметрів є найпотужнішим відкритим LLM згідно з таблицею лідерів LMSYS Chatbot Arena, він потужніший за GPT-3.5, але не такий потужний, як GPT-4.[17]]

Станом на 2024 рік усі найбільші та найпотужніші моделі базуються на архітектурі Transformer. Деякі нещодавні реалізації базуються на інших архітектурах, таких як рекурентні варіанти нейронної мережі та Mamba (модель простору станів).[18][19][20]

Імовірнісна токенізація

Оскільки алгоритми машинного навчання обробляють числа, а не текст, текст потрібно перетворити на числа. На першому кроці визначається словник, потім цілі індекси довільно, але унікально призначаються кожному запису словника, і, нарешті, вбудовування пов’язується з цілочисельним індексом. Алгоритми включають кодування пар байтів і WordPiece.

Імовірнісна токенізація також стискає набори даних. Оскільки LLM зазвичай вимагають, щоб вхідні дані були масивом без зубців, коротші тексти повинні бути «доповнені», доки вони не збігаються з довжиною найдовшого. Скільки в середньому токенів потрібно на одне слово, залежить від мови набору даних.[21][22]

BPE

Використовуючи модифікацію кодування пари байтів, на першому кроці всі унікальні символи (включаючи пробіли та знаки пунктуації) розглядаються як початковий набір n-грам (тобто початковий набір уні-грам). Послідовно найбільш часта пара суміжних символів об'єднується в біграмму, і всі екземпляри пари замінюються нею. Усі входження суміжних пар (раніше об’єднаних) n-грам, які найчастіше зустрічаються разом, потім знову об’єднуються в ще більшу n-граму, доки не буде отримано словник заданого розміру (у випадку GPT-3 розмір становить 50257) [23] Словник токенів складається з цілих чисел від нуля до розміру словника токенів. Нові слова завжди можна інтерпретувати як комбінації лексем і уніграм початкового набору.[24]

Словник лексем, заснований на частотах, отриманих переважно з англійських корпусів, використовує якомога менше лексем для середнього англійського слова. Однак середнє слово іншою мовою, закодоване таким оптимізованим для англійської мови токенізатором, розбивається на субоптимальну кількість токенів. Токенізатор GPT-2 може використовувати до 15 разів більше токенів на слово для деяких мов, наприклад для шанської мови з М’янми. Навіть більш поширені мови, такі як португальська та німецька, мають «надбавку на 50%» порівняно з англійською.[25]

1.3 Постановка задачі

Постановка задачі для розробки блогового додатку із штучним інтелектом полягає в створенні веб-платформи, яка дозволить користувачам ефективно управляти контентом, включаючи створення, редагування та видалення постів. Окрім базових функцій управління контентом, додаток включатиме інтеграцію інструменту штучного інтелекту, зокрема LLama AI, для генерації текстів за запитом користувачів. Це дозволить користувачам легше створювати оригінальний контент, базуючись на заданих описах та темах.

Система повинна забезпечувати високий рівень безпеки та приватності, зокрема через авторизацію і аутентифікацію користувачів. Також важливо врахувати зручність користувачів, забезпечивши інтуїтивно зрозумілий графічний інтерфейс.

Ключові аспекти постановки задачі включають:

1. Розробка веб-інтерфейсу для управління блогом.
2. Інтеграція інструментів штучного інтелекту для автоматизації створення контенту.
3. Реалізація системи авторизації для забезпечення доступу до адміністративних функцій.
4. Забезпечення високого рівня безпеки користувацьких даних та інформації, що зберігається у системі.
5. Розробка зручних та ефективних механізмів для введення, редагування та видалення постів.
6. Тестування та оптимізація системи для забезпечення стабільності роботи та швидкості відгуку на запити користувачів.

Завершення проекту вимагатиме тестування всіх компонентів системи для забезпечення їх надійності та відповідності встановленим вимогам.

РОЗДІЛ 2
ПРОЕКТУВАННЯ СИСТЕМИ

2.1 Архітектура системи

Архітектура MVT (Model-View-Template) є однією з популярних архітектурних концепцій у веб-розробці. Вона схожа на архітектуру MVC (Model-View-Controller), але має свої особливості. У MVT розділення обов'язків між компонентами здійснюється таким чином, що кожен компонент відповідає за свої чітко визначені функції, що сприяє покращенню читабельності та підтримці коду.

Модель (Model) відповідає за доступ до даних і бізнес-логіку програми. У MVT модель представляє об'єкти даних та логіку, що визначає взаємодію з цими об'єктами. Модель може включати класи або функції, які працюють з базою даних або іншими джерелами даних. Вона не повинна залежати від інших компонентів.

Представлення (View) відповідає за відображення даних моделі користувачу та обробку введення користувача. Він може генерувати HTML-код для відображення сторінок, обробляти запити користувачів і взаємодіяти з моделлю для отримання або збереження даних. Представлення не має містити бізнес-логіку і повинен концентруватися на тому, як дані відображаються користувачеві.

Шаблон (Template) відповідає за представлення даних моделі у вигляді, зручному для відображення користувачу. Шаблони використовуються для генерації HTML-коду на основі даних, що надходять з представлення. Вони дозволяють вставляти дані в статичний HTML та управляти логікою відображення, такою як цикли та умовні оператори.

Однією з ключових переваг MVT є відокремлення бізнес-логіки, представлення та логіки відображення. Це сприяє підтримці та розширенню

коду, а також полегшує тестування. Крім того, використання шаблонів дозволяє легко змінювати вигляд веб-сторінок без втручання в бізнес-логіку.

Проте, архітектура MVT може бути складнішою для розуміння, особливо для початківців у веб-розробці, порівняно з іншими архітектурними підходами, такими як MVC. Крім того, у деяких випадках може виникнути дублювання коду, оскільки деякі функції можуть бути присутніми як у виді, так і в шаблоні.

У підсумку, архітектура MVT є потужним інструментом для розробки веб-додатків, який дозволяє чітко визначити обов'язки кожного компонента та сприяє підтримці та розширенню коду. Її використання може полегшити розробку та підтримку веб-додатків з чіткою логікою та розділеними відповідностями.

На рисунку 2.1 представлено архітектуру системи.

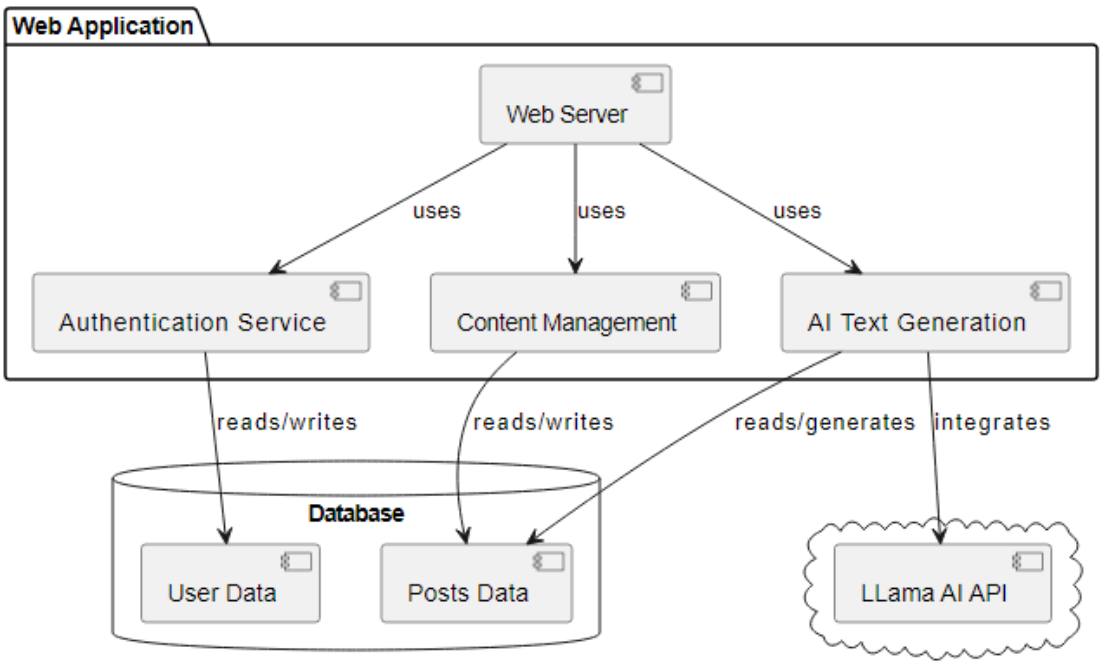

Рисунок 2.1 — Архітектура системи

Модель водоспаду, також відома як модель життєвого циклу програмного забезпечення, є однією з найбільш відомих та широко використовуваних

моделей у сфері програмної інженерії. Ця модель визначає послідовність етапів розробки програмного забезпечення від постановки завдання до впровадження та підтримки.

1. Постановка завдання (Inception): на цьому етапі визначаються вимоги до програмного продукту. Це включає в себе аналіз потреб користувачів, визначення функціональності та встановлення цілей проекту.

2. Аналіз (Requirements Analysis): на цьому етапі детально аналізуються вимоги, визначені на попередньому етапі. Формулюються технічні вимоги, планується архітектура системи та розробляється стратегія реалізації.

3. Проектування (Design): на цьому етапі розробляється детальний план реалізації системи. Визначається архітектура системи, розробляються діаграми, вибираються технології та інструменти для реалізації.

4. Реалізація (Construction): на цьому етапі програмне забезпечення фактично розробляється та тестується. Реалізуються всі компоненти та функціональність системи.

5. Випробування (Testing): після завершення реалізації система піддається випробуванням для перевірки на відповідність вимогам та виявлення можливих помилок.

6. Впровадження (Deployment): після успішного завершення випробувань програмне забезпечення впроваджується в експлуатацію. Цей етап включає підготовку до впровадження та навчання користувачів.

7. Підтримка (Maintenance): на цьому етапі забезпечується подальша підтримка та розвиток програмного забезпечення. Виконуються патчі, виправлення помилок та вдосконалення функціональності.

Модель водоспаду є досить простою та легко зрозумілою для багатьох розробників, оскільки вона передбачає послідовний характер роботи. Проте, вона може бути менш гнучкою у порівнянні з іншими методологіями, такими як

Agile, які надають більше можливостей для адаптації до змін у вимогах та швидшого реагування на них.

2.2 Проектування інтерфейсу системи

Проектування функціоналу системи з використанням діаграм варіантів використання є важливою частиною процесу розробки програмного забезпечення. Діаграми варіантів використання (Use Case Diagrams) дозволяють моделювати функціональність системи шляхом визначення взаємодій між користувачами та системою. Нижче розглянемо, як це відбувається на рівні концептуального проектування.

Визначення акторів: перший крок у проектуванні функціоналу - це визначення акторів. Актори - це користувачі або зовнішні системи, які взаємодіють з системою. На діаграмі вони представлені у вигляді сторін або блоків, що вказує на їх роль у системі.

Визначення варіантів використання: далі потрібно визначити варіанти використання системи. Це можуть бути конкретні дії або функції, які можуть бути виконані акторами в системі. Кожен варіант використання описується умовами та діями, які відбуваються під час виконання.

Побудова діаграми варіантів використання: після визначення акторів та варіантів використання можна побудувати діаграму варіантів використання. На цій діаграмі актори представлені у вигляді блоків або сторін, а варіанти використання - у вигляді стрілок, що вказують на взаємодію між акторами та системою.

Використання діаграм варіантів використання у проекті: діаграми варіантів використання можуть бути використані у проекті для розуміння вимог до системи, визначення функціональності та взаємодії з користувачами. Вони допомагають уникнути непорозумінь між розробниками та замовниками, а також забезпечують основу для подальшого проектування та реалізації системи.

У підсумку, проектування функціоналу системи з використанням діаграм варіантів використання є важливою частиною процесу розробки програмного забезпечення. Ці діаграми дозволяють визначити, як система буде

використовуватися користувачами та іншими системами, що дозволяє забезпечити якість та ефективність розробленого продукту.

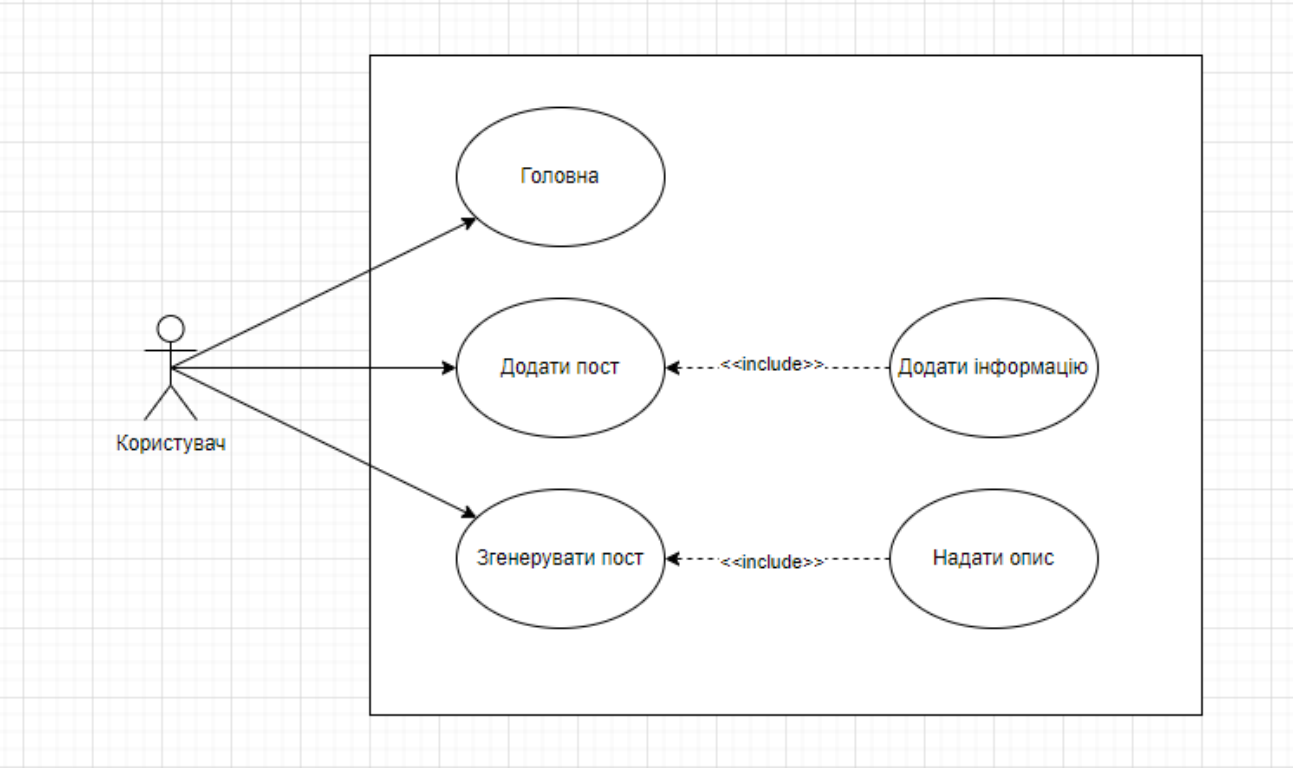

Рисунок 2.2 — Діаграма варіантів використання

Специфікація варіантів використання

Перегляд головної сторінки:

- Опис: користувач може переглянути головну сторінку, щоб бачити всі доступні пости.
- Передумова: користувач має доступ до інтернету.
- Основний сценарій: користувач відкриває головну сторінку, система показує список усіх постів.
- Альтернативний сценарій: якщо пости відсутні, система показує повідомлення про відсутність контенту.

Додавання посту:

- Опис: користувач може додати новий пост до блогу.
- Передумова: користувач має бути залогінений.

- Основний сценарій: користувач надає заголовок та текст посту, система зберігає пост.
- Альтернативний сценарій: якщо надані дані є некоректними, система показує помилку.

Генерація посту:

- Опис: користувач може згенерувати текст посту за допомогою інтегрованого ШІ.
- Передумова: користувач має бути залогінений і має доступ до функції генерації.
- Основний сценарій: користувач вводить опис, на основі якого ШІ генерує текст посту.
- Альтернативний сценарій: якщо опис відсутній або занадто короткий, система вимагає більш детальний опис.

Додавання інформації (розширення додавання посту):

- Опис: користувач може додати додаткову інформацію до посту, таку як зображення чи теги.
- Передумова: пост має бути створений або в процесі створення.
- Основний сценарій: користувач вибирає або завантажує додаткові матеріали для посту.
- Альтернативний сценарій: якщо завантаження не вдається, система повідомляє про помилку.

Надання опису (розширення генерації посту):

- Опис: користувач може надати детальний опис для генерації контенту.
- Передумова: користувач вирішив використати ШІ для генерації тексту.
- Основний сценарій: користувач вводить детальний опис теми, яку ШІ повинен обробити для створення тексту.
- Альтернативний сценарій: якщо опис занадто вагомий або нечіткий, система вимагає уточнення.

3.3 Проектування внутрішньої будови

Під час розробки програмного забезпечення, зазвичай, зображення внутрішньої структури системи подається у формі діаграми груп та курсів, яка демонструє склад груп та модулів проекту та їх взаємодію.

У сфері програмної інженерії, діаграма груп курсів, згідно з уніфікованою мовою моделювання (UML), є статичною структурною діаграмою, що надає огляд структури системи, включаючи системні класи, їх атрибути, операції (або методи) та зв'язки між об'єктами.

Діаграми класів виступають як ключові будівельні блоки об'єктно-орієнтованого моделювання. Вони використовуються для загального концептуального моделювання структури програми, а також для більш детального моделювання з метою перетворення моделі в програмний код. Також ці діаграми можуть служити інструментом моделювання даних. Кожен клас на діаграмі класів представляє основні елементи програми та класи, які будуть реалізовані.

На графічному зображенні класу зазвичай можна виділити три окремі частини:

• Верхній блок, який містить назву класу, зазвичай надруковану жирним шрифтом та вирівняну за центром. Перша літера назви має бути великою.

• Середній блок, де розташовані атрибути класу, які вирівнюються ліворуч, а перша літера їх назви повинна бути мала.

• Нижній блок, який містить операції, які можуть бути виконані класом. Операції також вирівнюються ліворуч, і перша літера їх назви має бути мала.

При проектуванні системи, визначення багатьох класів та їх групування в схему класів допомагають зрозуміти статичні відносини між ними. Для більш детального моделювання, категорія концептуального проектування зазвичай поділяється на декілька підкатегорій.

Асоціація описує взаємозв'язок між елементами моделі. Якщо зміна одного елемента може призвести до зміни іншого, то між ними існує асоціація. Це відношення є одностороннім і представлене пунктирною лінією зі стрілкою, що вказує на напрямок від відправлювача до отримувача.

Для подальшого опису системи, ці діаграми груп можуть бути доповнені діаграмами станів або становими автоматами UML.

Асоціація включає в себе різні типи: двосторонню, односторонню, агрегаційну (включаючи комбіновану агрегацію) та рефлексивну. Серед них найпоширеніші двосторонні та односторонні асоціації.

Наприклад, взаємозв'язок між класом "польот" і класом "літак" можна описати як двосторонню асоціацію. Ця асоціація відображає статичні відносини, що існують між об'єктами обох класів.

Агрегація представляє собою варіант відношення "має", і вона є більш специфічною за асоціацію. Агрегація виражає зв'язок, в якому один клас містить частину іншого класу. Наприклад, як показано на діаграмі, клас "професор" має асоціацію з класом "викладання". У якості типу асоціації, агрегація може мати назву та модифікації, подібні до асоціації. Але важливо відзначити, що агрегація обмежена бінарними відносинами, тобто це завжди асоціація між двома класами.

Особливістю агрегації є те, що класи, що містяться в агрегації, не мають сильної залежності від життєвого циклу контейнера. Навіть після знищення контейнера, елементи, що містяться в ньому, все ще існують.

Графічно агрегацію в UML зображують у вигляді порожнистого ромба, що з'єднується однією лінією з класом-господарем. Агрегація може розглядатися як семантично вищий рівень об'єкта, що взаємодіє з іншими об'єктами, хоча фізично складається з кількох менших об'єктів.

Наприклад, розглянемо взаємозв'язок між "бібліотекою" та "студентами". У цьому випадку, "студенти" можуть існувати незалежно від "бібліотеки", і

відношення між ними може бути описане як агрегація, оскільки "студенти" є частиною "бібліотеки". Таке відношення можна позначити як сукупність.

Узагальнення визначає відношення "є спеціалізацією" між класами. Воно показує, що один клас (підтип) є більш конкретним представленням іншого класу (супертипу). Наприклад, "люди" є підтипом "ссавців", і "ссавці" є супертипом "тварин". Графічно відношення узагальнення в UML позначаються порожнім трикутником, який з'єднується зі супертипом.

Відношення узагальнення часто отримують назву спадковістю або відносинами "є". У контексті цих відносин, суперклас (також відомий як базовий клас) можна називати "батьківським класом", "суперкласом", "базовим класом" або "базовим типом". Спеціалізовані класи, що успадковують від суперкласу, можуть називатися "дочірніми" підкласами, "похідними класами", "похідними типами", "успадкованими класами" або "успадкованими типами". Важливо зазначити, що ці терміни вживаються у контексті програмування та моделювання, і вони не мають жодного спільного з біологічними відносинами між батьком і дитиною.

Звідси ми можемо висловити відношення "А є типом В" такими прикладами, як "дуб є одним із видів дерев", "автомобіль є підкласом транспорту".

В моделюванні UML відношення реалізації вказують на те, що один елемент моделі (клієнт) втілює або реалізує функціональність, яка була задана іншим елементом моделі (постачальником). Це відношення графічно позначається порожнім трикутником, який з'єднаний інтерфейсом або деревом реалізаторів. Крім того, на діаграмі компонентів використовується графічна позначка "м'ячна розетка" для позначення реалізації. Важливо відзначити, що відношення реалізації зазвичай зображуються лише на діаграмах класів або компонентів, оскільки вони вказують на зв'язок між класами, інтерфейсами, компонентами і пакетами.

Залежність є слабкішою формою взаємодії, яка вказує на те, що один клас залежить від іншого і може використовувати його в певний момент часу. Відмінністю від асоціації є те, що залежність не передбачає наявність атрибутів одного класу, які були б екземплярами іншого класу. Відношення реалізації і асоціації графічно представлені як лінії, що з'єднують класи, з додатковими символами на кінцях ліній для вказівки додаткових деталей відношення.

Класи сутності в моделюванні представляють інформацію, яку система обробляє, а також поведінку, пов'язану з цією інформацією. Графічне представлення класів сутностей зазвичай має форму кола з короткою лінією, прикріпленою до нижньої частини кола, або їх можна намалювати як звичайні класи зі стереотипом "сутність" над назвою класу.

На рисунку 2.3 зображено діаграму класів, яка відображає внутрішню будову проекту.

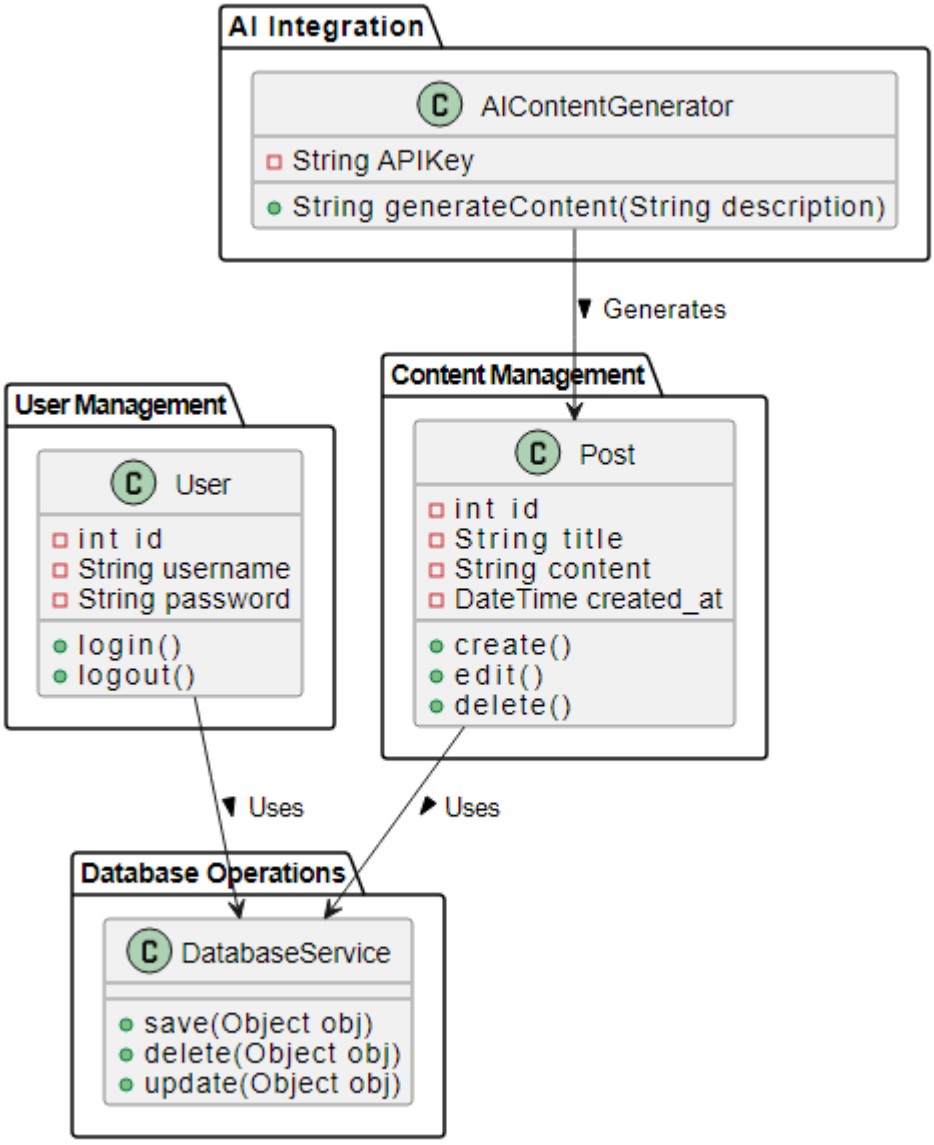

Рисунок 2.3 — Діаграма класів системи

Діаграма включає чотири основні компоненти, кожен з яких відповідає за певний аспект функціональності системи.

Компонент "User Management"

Клас User:

Атрибути:

int id: Унікальний ідентифікатор користувача.

String username: Ім'я користувача, яке використовується для входу в систему.

String password: Пароль користувача для аутентифікації.

Методи:

login(): Метод для входу користувача в систему.

logout(): Метод для виходу користувача з системи.

Компонент "Content Management"

Клас Post:

Атрибути:

int id: Унікальний ідентифікатор посту.

String title: Заголовок посту.

String content: Вміст посту.

DateTime created_at: Час створення посту.

Методи:

create(): Створення нового посту.

edit(): Редагування існуючого посту.

delete(): Видалення посту з системи.

Компонент "AI Integration"

Клас AIContentGenerator:

Атрибути:

String APIKey: Ключ для доступу до API штучного інтелекту.

Методи:

generateContent(String description): Метод, що генерує контент за допомогою штучного інтелекту, базуючись на наданому описі.

Компонент "Database Operations"

Клас DatabaseService:

Методи:

save(Object obj): Зберігання об'єкта в базі даних.

delete(Object obj): Видалення об'єкта з бази даних.

update(Object obj): Оновлення об'єкта в базі даних.

Взаємодії між компонентами

Класи User та Post використовують сервіси DatabaseService для зберігання та управління даними, що відображено стрілками з підписом "Uses".

Клас AIContentGenerator генерує контент для класу Post, що символізується стрілкою з підписом "Generates".

РОЗДІЛ 3
РОЗРОБКА І ТЕСТУВАННЯ СИСТЕМИ

3.1 Вибір інструментів

3.1.1 Вибір мови програмування

Python є однією з найпопулярніших мов програмування у світі завдяки своїй простоті, читабельності та багатофункціональності. Вона широко використовується у різних сферах, включаючи веб-розробку, наукові обчислення, обробку даних, машинне навчання та автоматизацію. Завдяки своїм численним перевагам, Python став вибором номер один для багатьох розробників та компаній по всьому світу.

Однією з головних переваг Python є його простота та читабельність. Синтаксис мови спроєктований таким чином, щоб бути зрозумілим та інтуїтивно зрозумілим навіть для новачків. Використання відступів для позначення блоків коду робить його структурованим та легким для читання. Це дозволяє швидко вивчати мову та розпочати розробку програм навіть без глибокого технічного досвіду. Завдяки цій особливості Python часто рекомендують як першу мову програмування для початківців.

Python також має розвинену екосистему бібліотек та фреймворків, що значно розширює його можливості та робить його придатним для вирішення широкого спектра задач. Наприклад, бібліотеки NumPy та Pandas широко використовуються для наукових обчислень та обробки даних, а бібліотеки TensorFlow та PyTorch – для машинного навчання та глибинного навчання. Фреймворки Django та Flask є популярними інструментами для веб-розробки, що дозволяють швидко створювати потужні та масштабовані веб-додатки.

Ще однією важливою перевагою Python є його крос-платформенність. Код, написаний на Python, може виконуватися на різних операційних системах, включаючи Windows, macOS та Linux, без внесення змін. Це забезпечує високу

портативність та гнучкість, дозволяючи розробникам створювати додатки, які можуть працювати на різних платформах. Крім того, Python підтримує інтеграцію з іншими мовами програмування, такими як C, C++ та Java, що дозволяє використовувати його в різних середовищах та проектах.

Python також відомий своєю активною та підтримуючою спільнотою. Величезна кількість розробників по всьому світу активно використовують та розвивають цю мову, що забезпечує наявність великої кількості ресурсів, таких як документація, курси, форуми та бібліотеки. Спільнота Python постійно працює над вдосконаленням мови, додаючи нові можливості та оптимізації, що дозволяє Python залишатися сучасним та конкурентоспроможним інструментом.

Порівнюючи Python з іншими мовами програмування, такими як Java, C++ або JavaScript, можна виділити кілька ключових переваг. По-перше, Python забезпечує швидший процес розробки завдяки своїй простоті та високорівневій природі. Розробники можуть писати менше коду для досягнення тих самих результатів, що знижує час на розробку та тестування. По-друге, Python має велику кількість готових бібліотек та модулів, що дозволяє швидко знаходити та використовувати рішення для різних задач без необхідності писати все з нуля.

У порівнянні з Java, Python відрізняється меншою суворістю типізації та більш динамічним підходом до управління пам'яттю. Це робить Python більш гнучким та зручним для швидкої розробки та прототипування. Однак, Java може забезпечити вищу продуктивність та надійність у великих та складних проектах завдяки своїй статичній типізації та суворим правилам компіляції.

Python також виграє у порівнянні з C++ у питанні простоти та читабельності коду. Хоча C++ може забезпечити вищу продуктивність та більший контроль над ресурсами системи, це також призводить до більшої складності та ризику помилок. Python, з іншого боку, забезпечує високу продуктивність розробки та легкість підтримки, що робить його ідеальним

вибором для багатьох проектів, де швидкість розробки та зручність мають пріоритет.

JavaScript, який є основною мовою для веб-розробки, також має свої переваги та недоліки у порівнянні з Python. Хоча JavaScript добре підходить для створення інтерактивних веб-інтерфейсів, Python забезпечує більш потужні інструменти для обробки даних, наукових обчислень та машинного навчання. Це робить Python кращим вибором для задач, що вимагають складної обробки та аналізу даних.

Ще однією важливою особливістю Python є його підтримка об'єктно-орієнтованого програмування (ООП). Python дозволяє створювати класові структури, наслідування та інкапсуляцію, що робить його потужним інструментом для розробки складних додатків. ООП у Python є інтуїтивно зрозумілим та легко освоюваним, що дозволяє швидко створювати модульний та масштабований код.

Python також підтримує функціональне програмування, що дозволяє використовувати такі концепції, як лямбда-функції, ітератори та генератори. Це забезпечує додаткову гнучкість та можливість використовувати різні парадигми програмування в залежності від конкретних вимог проекту. Така мультипарадигмовість робить Python універсальним інструментом, придатним для широкого спектру задач.

У сфері машинного навчання та обробки даних Python є безперечним лідером завдяки своїм потужним бібліотекам та фреймворкам. Бібліотеки, такі як scikit-learn, TensorFlow, Keras та PyTorch, забезпечують інструменти для побудови та тренування моделей машинного навчання, обробки великих обсягів даних та аналізу результатів. Це робить Python ідеальним вибором для дослідників та розробників, які працюють у цій галузі.

Крім того, Python широко використовується у сфері автоматизації завдяки своїм зручним та потужним інструментам. Бібліотеки, такі як Selenium та

Beautiful Soup, дозволяють автоматизувати взаємодію з веб-сайтами, збирання даних та тестування веб-додатків. Це значно спрощує процес автоматизації та підвищує продуктивність розробки.

Підтримка Python у наукових обчисленнях та аналізі даних є ще однією важливою перевагою. Бібліотеки, такі як NumPy, SciPy та Pandas, забезпечують потужні інструменти для роботи з масивами даних, математичними обчисленнями та статистичним аналізом. Це робить Python важливим інструментом для дослідників, вчених та аналітиків, які працюють з великими обсягами даних.

Підсумовуючи, Python є потужною, гнучкою та універсальною мовою програмування, яка забезпечує високу продуктивність, легкість освоєння та широкі можливості для розробників. Її простота, багатофункціональність та активна спільнота роблять її ідеальним вибором для широкого спектру задач, від веб-розробки до наукових обчислень та машинного навчання. Незалежно від рівня досвіду чи типу проекту, Python надає всі необхідні інструменти для успішної реалізації будь-якої ідеї.

3.1.2 Вибір середовища розробки

PyCharm є одним з найпопулярніших інтегрованих середовищ розробки (IDE) для мови програмування Python, розроблене компанією JetBrains. PyCharm забезпечує розробникам потужний набір інструментів для написання, тестування та налагодження коду, що робить його вибором номер один для багатьох Python-розробників по всьому світу.

Однією з ключових особливостей PyCharm є його інтуїтивно зрозумілий і зручний інтерфейс. Розробники можуть легко налаштовувати середовище під свої потреби, використовуючи різні теми, гарячі клавіші та інші параметри. Це забезпечує високу продуктивність роботи та комфорт при написанні коду.

PyCharm підтримує багато мов інтерфейсу, що робить його доступним для розробників з різних країн.

PyCharm пропонує потужний редактор коду з функцією автозаповнення, яка допомагає швидко писати код та зменшити кількість помилок. Інтелектуальні підказки та завершення коду значно спрощують процес розробки, особливо для новачків. Крім того, PyCharm забезпечує підтримку рефакторингу коду, що дозволяє легко змінювати структуру проекту та підтримувати чистоту коду.

Однією з найважливіших особливостей PyCharm є його потужні інструменти для налагодження та тестування. Інтегрований відлагоджувач дозволяє розробникам запускати код по кроках, встановлювати точки зупинки та аналізувати значення змінних у реальному часі. Це значно спрощує процес виявлення та виправлення помилок, підвищуючи якість коду. Крім того, PyCharm підтримує інтеграцію з популярними фреймворками для тестування, такими як pytest та unittest, що дозволяє легко створювати та запускати тести для вашого проекту.

PyCharm також підтримує роботу з віртуальними середовищами, що є важливою складовою сучасного розроблення на Python. Це дозволяє розробникам ізолювати залежності проекту та уникати конфліктів між різними бібліотеками та версіями пакетів. Вбудована підтримка інструментів для управління пакетами, таких як pip та conda, дозволяє легко встановлювати та оновлювати залежності безпосередньо з середовища розробки.

Ще однією важливою особливістю PyCharm є його інтеграція з системами контролю версій, такими як Git, Mercurial та SVN. Це дозволяє розробникам ефективно керувати версіями коду, працювати в команді та відстежувати зміни в проектах. Інтеграція з Git забезпечує зручні інструменти для злиття змін, вирішення конфліктів та відкату до попередніх версій, що значно підвищує продуктивність та надійність роботи.

PyCharm також пропонує потужні інструменти для роботи з базами даних. Вбудований інструмент для роботи з SQL дозволяє розробникам підключатися до різних баз даних, виконувати SQL-запити та переглядати результати безпосередньо з середовища розробки. Це значно спрощує процес роботи з базами даних та підвищує ефективність розробки.

Однією з ключових переваг PyCharm є його підтримка різних фреймворків та технологій. Наприклад, PyCharm підтримує популярні веб-фреймворки, такі як Django, Flask та Pyramid, що дозволяє легко створювати та налагоджувати веб-додатки. Крім того, PyCharm підтримує фреймворки для наукових обчислень та машинного навчання, такі як NumPy, Pandas, TensorFlow та PyTorch, що робить його ідеальним інструментом для дослідників та розробників у цих галузях.

Порівнюючи PyCharm з іншими IDE, такими як Visual Studio Code або Spyder, можна виділити кілька ключових переваг. По-перше, PyCharm забезпечує більш глибоку інтеграцію з Python та його екосистемою, що робить його більш потужним інструментом для розробки на цій мові. По-друге, PyCharm пропонує ширший набір функцій для налагодження та тестування, що значно полегшує процес розробки та підвищує якість коду. Крім того, PyCharm підтримує більшу кількість фреймворків та технологій, що робить його більш універсальним інструментом для різних типів проектів.

Ще однією важливою перевагою PyCharm є його активна та підтримуюча спільнота. Величезна кількість розробників по всьому світу активно використовують та розвивають це середовище, що забезпечує наявність великої кількості ресурсів, таких як документація, курси, форуми та плагіни. Спільнота PyCharm постійно працює над вдосконаленням IDE, додаючи нові можливості та оптимізації, що дозволяє PyCharm залишатися сучасним та конкурентоспроможним інструментом.

PyCharm також пропонує розширені можливості для налаштування середовища розробки. Розробники можуть встановлювати різні плагіни, які додають нові функції та інтеграції, що робить PyCharm ще більш гнучким та потужним інструментом. Вбудована підтримка для роботи з контейнерами Docker та оркестрацією Kubernetes дозволяє легко створювати, тестувати та розгортати додатки в контейнеризованих середовищах, що є важливим аспектом сучасного розроблення.

Для командної роботи PyCharm пропонує розширені можливості для кодування та співпраці. Інтеграція з інструментами для огляду коду, такими як Upsource, дозволяє розробникам ефективно переглядати та обговорювати зміни в коді, що підвищує якість та продуктивність команди. Крім того, PyCharm підтримує інтеграцію з різними системами управління проектами, такими як JIRA та YouTrack, що спрощує процес управління завданнями та відстеження прогресу.

Однією з важливих особливостей PyCharm є його підтримка різних середовищ розробки, включаючи віддалені сервери та хмарні платформи. Це дозволяє розробникам працювати з кодом, що знаходиться на віддалених серверах, використовуючи всі можливості PyCharm для написання, налагодження та тестування коду. Така гнучкість забезпечує ефективну роботу в різних умовах та середовищах.

Загалом, PyCharm є потужним та зручним інтегрованим середовищем розробки для Python, яке забезпечує високий рівень продуктивності, надійності та зручності використання. Завдяки своїм розширеним можливостям, інтеграції з різними фреймворками та технологіями, а також активній спільноті, PyCharm залишається одним з найкращих виборів для розробників Python у всьому світі. Незалежно від рівня досвіду чи типу проекту, PyCharm надає всі необхідні інструменти для успішної реалізації будь-якої ідеї.

3.1.3 Вибір СКБД

SQLite є популярною вбудованою системою управління базами даних (СУБД), яка використовується в різних додатках та пристроях. Відома своєю легкістю, продуктивністю та повною відповідністю стандартам SQL, SQLite забезпечує надійне зберігання даних без необхідності встановлення окремого серверного програмного забезпечення. Її головними перевагами є простота використання, відсутність необхідності в налаштуванні та висока продуктивність при роботі з невеликими та середніми об'ємами даних.

SQLite була розроблена у 2000 році Річардом Гіпом з метою створення легкої та самодостатньої СУБД, яка могла б використовуватись вбудовано в програми. Відтоді вона стала однією з найпоширеніших баз даних у світі, використовуючись у мільйонах додатків, включаючи веб-браузери, мобільні додатки, операційні системи та вбудовані системи.

Однією з ключових особливостей SQLite є її вбудованість. На відміну від традиційних СУБД, таких як MySQL або PostgreSQL, які потребують окремого серверного середовища для роботи, SQLite є бібліотекою, яка компілюється та вбудовується безпосередньо у додаток. Це значно спрощує розгортання та використання бази даних, оскільки немає потреби в адмініструванні сервера, налаштуванні мережевих з'єднань та інших супутніх задачах.

Ще однією важливою перевагою SQLite є її портативність. Дані зберігаються у звичайному файлі на диску, що дозволяє легко переносити базу даних між різними системами та пристроями. Це робить SQLite ідеальним вибором для мобільних додатків, де необхідно забезпечити локальне зберігання даних без підключення до мережі. Крім того, файли баз даних SQLite можуть бути легко збережені, передані та інтегровані у резервні копії, що забезпечує додаткову гнучкість та надійність.

Продуктивність SQLite також є одним з її сильних аспектів. Завдяки оптимізації для роботи з файлами та ефективному використанню ресурсів

системи, SQLite забезпечує високу швидкість виконання запитів навіть на пристроях з обмеженими ресурсами. Хоча вона не призначена для роботи з великими об'ємами даних або високонавантажених додатків, її продуктивність є більш ніж достатньою для більшості типових сценаріїв використання, включаючи зберігання налаштувань додатків, історії користувачів, тимчасових даних та інше.

Серед інших переваг SQLite можна виділити її відповідність стандартам SQL. Вона підтримує більшість стандартних SQL команд та синтаксису, що дозволяє розробникам використовувати знайомі інструменти та підходи для роботи з базою даних. Це також полегшує міграцію даних та кодів між SQLite та іншими СУБД, якщо виникає така необхідність. Завдяки цьому SQLite є чудовим інструментом для розробки, тестування та прототипування додатків, які згодом можуть бути перенесені на більш потужні серверні СУБД.

Порівнюючи SQLite з іншими СУБД, такими як MySQL, PostgreSQL або Oracle, можна виділити кілька ключових переваг. По-перше, SQLite не потребує налаштування сервера та адміністрування, що значно спрощує розгортання та знижує витрати на підтримку. По-друге, вона забезпечує високу продуктивність та ефективність роботи з даними завдяки оптимізації для вбудованих додатків та роботи з файлами. По-третє, її портативність та можливість зберігання даних у звичайному файлі на диску робить її ідеальним вибором для мобільних додатків та вбудованих систем.

Однак варто зазначити, що SQLite має свої обмеження. Вона не призначена для роботи з великими об'ємами даних або високонавантажених додатків, де необхідна підтримка одночасного доступу великої кількості користувачів та складних транзакцій. У таких випадках доцільно використовувати більш потужні серверні СУБД, такі як PostgreSQL або MySQL, які забезпечують кращу масштабованість та продуктивність у таких сценаріях.

SQLite також має потужні можливості для роботи з транзакціями, що забезпечує надійність та цілісність даних. Вона підтримує ACID (Atomicity, Consistency, Isolation, Durability) властивості, що гарантує коректне виконання транзакцій навіть у випадку збоїв або аварійних завершень роботи системи. Це забезпечує високу надійність та стабільність роботи додатків, які використовують SQLite для зберігання важливих даних.

Ще однією важливою особливістю SQLite є її розширюваність. Вона підтримує створення користувацьких функцій, агрегацій та тригерів, що дозволяє розробникам розширювати функціональність бази даних відповідно до своїх потреб. Це забезпечує додаткову гнучкість та можливість адаптації SQLite до специфічних вимог додатка.

Однією з цікавих можливостей SQLite є підтримка віртуальних таблиць через інтерфейс SQL/MED (Management of External Data). Це дозволяє розробникам створювати віртуальні таблиці, які взаємодіють з зовнішніми джерелами даних, такими як файли, веб-сервіси або інші бази даних. Це відкриває нові можливості для інтеграції та обробки даних з різних джерел у межах одного додатка.

SQLite також широко використовується у вбудованих системах та Інтернеті речей (IoT) завдяки своїй легкості та ефективності. Вона забезпечує надійне зберігання даних на пристроях з обмеженими ресурсами, таких як сенсори, контролери та інші вбудовані пристрої. Це робить SQLite ідеальним вибором для розробки додатків для IoT, де необхідно забезпечити ефективне зберігання та обробку даних у реальному часі.

Постійний розвиток та підтримка з боку спільноти розробників є ще однією важливою перевагою SQLite. Вона має велику та активну спільноту, яка постійно працює над вдосконаленням бази даних, виправленням помилок та додаванням нових можливостей. Це забезпечує високу якість та надійність

продукту, а також доступ до великої кількості ресурсів, таких як документація, приклади коду та інструменти для розробки.

Загалом, SQLite є потужною та гнучкою системою управління базами даних, яка забезпечує високу продуктивність, надійність та зручність використання. Її легкість, портативність та відповідність стандартам SQL роблять її ідеальним вибором для широкого спектру додатків, включаючи мобільні додатки, вбудовані системи, веб-додатки та багато інших. Незалежно від рівня складності чи обсягу даних, SQLite надає всі необхідні інструменти для ефективного зберігання та обробки даних, забезпечуючи високу якість та надійність роботи додатків.

3.1.4 Вибір веб-фреймворку

Flask є мікрофреймворком для веб-розробки на Python, який був створений Арміном Ронахером (Armin Ronacher) у 2010 році. Flask базується на Werkzeug WSGI інструментальному наборі та Jinja2 шаблонізаторі. Завдяки своїй легкості, гнучкості та простоті, Flask став одним з найпопулярніших фреймворків для веб-розробки на Python, особливо серед розробників, які цінують мінімалізм і контроль над своїм кодом.

Однією з ключових особливостей Flask є його мікроархітектура, що означає, що він поставляється з мінімальним набором базових функцій, необхідних для веб-розробки. Це робить Flask дуже легким та швидким у налаштуванні і дозволяє розробникам додавати тільки ті компоненти, які дійсно потрібні для їхнього проекту. Такий підхід забезпечує високу гнучкість та дає можливість створювати як прості, так і дуже складні додатки.

Flask надає розробникам повний контроль над маршрутизацією URL. Використовуючи декоратори, можна легко визначати маршрути для різних URL, що дозволяє створювати чисту та зрозумілу структуру веб-додатка. Це

спрощує процес розробки та підтримки додатка, оскільки всі маршрути чітко визначені та легко доступні.

Ще однією важливою перевагою Flask є вбудована підтримка шаблонів через Jinja2. Jinja2 – це потужний шаблонізатор, який дозволяє розробникам створювати динамічні HTML-сторінки з використанням Python-коду. Використовуючи Jinja2, можна легко вбудовувати змінні, керуючі структури та інші елементи безпосередньо у HTML, що забезпечує високу гнучкість та зручність створення користувацьких інтерфейсів.

Flask також підтримує розширення, які дозволяють додавати додаткові функціональні можливості до базового фреймворка. Існує велика кількість розширень для Flask, які покривають різні аспекти веб-розробки, такі як автентифікація (Flask-Login), обробка форм (Flask-WTF), інтеграція з базами даних (Flask-SQLAlchemy), міграції баз даних (Flask-Migrate) та багато інших. Це дозволяє розробникам легко додавати нові функції до свого додатка, не ускладнюючи базовий код.

Порівнюючи Flask з іншими веб-фреймворками, такими як Django або Pyramid, можна виділити кілька ключових відмінностей. По-перше, Flask надає більш мінімалістичний підхід до веб-розробки, що дозволяє розробникам створювати додатки з нуля та додавати тільки ті компоненти, які їм потрібні. Це забезпечує високу гнучкість та легкість налаштування. По-друге, Flask має меншу криву навчання, що робить його більш доступним для новачків у веб-розробці. По-третє, Flask забезпечує більш гнучку структуру проекту, що дозволяє розробникам організовувати код відповідно до своїх уподобань та потреб.

Однією з важливих особливостей Flask є його висока продуктивність та ефективність. Завдяки своїй легкості та мінімалістичному підходу, Flask забезпечує швидкий час відгуку та низьку затримку, що робить його ідеальним вибором для створення високонавантажених веб-додатків. Крім того, Flask

легко інтегрується з різними системами кешування, такими як Redis або Memcached, що дозволяє ще більше підвищити продуктивність додатка.

Flask також відомий своєю відмінною документацією та активною спільнотою. Документація Flask детально пояснює всі аспекти використання фреймворка, включаючи налаштування, маршрутизацію, обробку запитів, роботу з шаблонами та багато іншого. Це робить Flask доступним та зрозумілим навіть для новачків. Активна спільнота розробників Flask постійно працює над вдосконаленням фреймворка, додаючи нові можливості та виправляючи помилки. Існує також велика кількість ресурсів, таких як форуми, блоги, відеоуроки та курси, які допомагають розробникам вивчати та використовувати Flask.

Ще однією важливою перевагою Flask є його інтеграція з різними інструментами для розробки та деплою. Наприклад, Flask легко інтегрується з системами контролю версій, такими як Git, що дозволяє розробникам ефективно керувати своїми проектами та співпрацювати з іншими. Крім того, Flask підтримує інтеграцію з інструментами для безперервної інтеграції та доставки (CI/CD), такими як Jenkins, Travis CI або GitHub Actions, що спрощує процес автоматизації розгортання додатків.

Flask також підтримує різні способи розгортання додатків, включаючи розгортання на традиційних веб-серверах, таких як Apache або Nginx, а також у хмарних середовищах, таких як AWS, Google Cloud Platform або Heroku. Це забезпечує високу гнучкість та дозволяє розробникам вибирати найкращі методи розгортання відповідно до своїх потреб та бюджету.

Flask є відмінним вибором для створення RESTful API завдяки своїй гнучкості та легкості використання. Використовуючи Flask, розробники можуть легко створювати API для своїх додатків, визначаючи маршрути та обробники для різних HTTP-запитів. Це дозволяє створювати масштабовані та ефективні

серверні частини додатків, які можуть обробляти великі обсяги запитів від клієнтів.

У підсумку, Flask є потужним та гнучким інструментом для веб-розробки на Python, який забезпечує високу продуктивність, простоту використання та широкий набір можливостей. Завдяки своїй мікроархітектурі, Flask дозволяє розробникам створювати додатки будь-якої складності, від простих сайтів до складних веб-сервісів та API. Активна спільнота та відмінна документація роблять Flask доступним та зрозумілим для розробників різного рівня. Незалежно від того, чи ви новачок у веб-розробці, чи досвідчений програміст, Flask надає всі необхідні інструменти для успішної реалізації ваших проектів.

3.1.5 Вибір LLM

LLama (Large Language Model) є потужною мовною моделлю, розробленою для генерації тексту та вирішення різноманітних задач обробки природної мови. В основі LLama лежить сучасна архітектура трансформерів, що забезпечує високу якість та точність генерації тексту. Завдяки своїй здатності розуміти та генерувати людську мову, LLama знаходить застосування у багатьох сферах, включаючи автоматизацію, створення контенту, підтримку клієнтів та інше.

Однією з ключових особливостей LLama є її здатність працювати з великими обсягами даних та генерувати текст, який є природним та зрозумілим для людей. Це досягається завдяки навчанню моделі на великих корпусах текстів, що включають різноманітні джерела, такі як книги, статті, блоги, форуми та інші текстові ресурси. Завдяки цьому LLama може генерувати текст, який враховує контекст та є логічно зв'язним.

LLama може використовуватись для різних задач обробки природної мови, включаючи генерацію тексту, автоматичний переклад, аналіз настроїв, відповідь на питання, створення діалогових систем та багато інших. Висока

точність та якість генерації тексту робить LLama потужним інструментом для автоматизації різних процесів та покращення взаємодії з користувачами.

Однією з важливих переваг LLama є її гнучкість та адаптивність. Модель може бути навчена для вирішення конкретних задач або адаптована до специфічних доменів знань. Це дозволяє створювати спеціалізовані системи, які забезпечують високу точність та ефективність у конкретних галузях. Наприклад, LLama може бути використана для створення діалогових систем, які відповідають на питання користувачів у сфері медицини, юриспруденції, технічної підтримки тощо.

Порівнюючи LLama з іншими мовними моделями, такими як GPT-3 від OpenAI або BERT від Google, можна виділити кілька ключових переваг. По-перше, LLama забезпечує високу якість генерації тексту завдяки сучасній архітектурі трансформерів та великому обсягу навчальних даних. По-друге, модель є дуже гнучкою та може бути адаптована до різних задач та доменів знань. По-третє, LLama забезпечує високу продуктивність та ефективність, що дозволяє використовувати її у реальному часі для вирішення різних задач обробки природної мови.

Однією з важливих особливостей LLama є її здатність до навчання на основі прикладів. Це дозволяє розробникам швидко навчати модель для вирішення конкретних задач, використовуючи невеликий набір прикладів. Такий підхід значно скорочує час та ресурси, необхідні для навчання моделі, та дозволяє швидко адаптувати її до нових задач та умов.

LLama також підтримує інтеграцію з різними платформами та інструментами, що робить її доступною для широкого кола розробників та компаній. Модель може бути інтегрована у веб-додатки, мобільні додатки, системи автоматизації бізнес-процесів та інші програмні рішення. Це забезпечує високу гнучкість та дозволяє використовувати LLama у різних сценаріях та умовах.

Однією з важливих сфер застосування LLama є автоматизація створення контенту. Модель може використовуватись для генерації статей, блогів, новинних повідомлень, маркетингових текстів та іншого контенту. Це дозволяє значно зменшити витрати на створення контенту та підвищити його якість та релевантність. Крім того, LLama може використовуватись для автоматизації перекладу текстів, що забезпечує високу якість та швидкість перекладу.

Ще однією важливою сферою застосування LLama є підтримка клієнтів та автоматизація обробки звернень. Модель може бути використана для створення чат-ботів та систем автоматичної відповіді на питання клієнтів. Це дозволяє значно знизити навантаження на служби підтримки та забезпечити швидку та якісну відповідь на питання клієнтів. Завдяки своїй здатності розуміти контекст та генерувати природний текст, LLama забезпечує високу якість взаємодії з користувачами.

У сфері навчання та освіти LLama також знаходить широке застосування. Модель може бути використана для створення інтелектуальних навчальних систем, які адаптуються до потреб та рівня знань учнів. Це дозволяє забезпечити індивідуальний підхід до навчання та підвищити ефективність освітнього процесу. Крім того, LLama може використовуватись для автоматизації перевірки завдань, генерування навчальних матеріалів та багато іншого.

Підсумовуючи, LLama є потужною та гнучкою мовною моделлю, яка забезпечує високу якість генерації тексту та може бути використана для вирішення різних задач обробки природної мови. Її простота у використанні, гнучкість та адаптивність роблять її ідеальним інструментом для розробників та компаній, які прагнуть автоматизувати свої процеси та покращити взаємодію з користувачами. Завдяки своїм численним перевагам та широкому спектру застосувань, LLama залишається однією з провідних мовних моделей у сфері обробки природної мови.

3.1.6 Вибір інструментів створення інтерфейсу

HTML (HyperText Markup Language) є основною мовою розмітки, що використовується для створення та структурування веб-сторінок. HTML забезпечує основу для відображення тексту, зображень, відео та інших елементів на веб-сторінках, а також дозволяє інтегрувати різні технології та скрипти для додавання інтерактивності та функціональності. Вперше розроблений Тімом Бернерсом-Лі у 1991 році, HTML став стандартом для веб-розробки, який продовжує розвиватися і вдосконалюватися з кожною новою версією.

Однією з головних переваг HTML є його простота та зручність у використанні. Синтаксис HTML є інтуїтивно зрозумілим, що дозволяє швидко навчитися основам створення веб-сторінок навіть без попереднього досвіду у програмуванні. HTML використовує теги для визначення різних елементів сторінки, таких як заголовки, параграфи, посилання, зображення тощо. Кожен тег має свою функцію та може містити атрибути для налаштування зовнішнього вигляду та поведінки елементів.

HTML забезпечує структурованість та семантичність веб-сторінок. Використання семантичних тегів, таких як <header>, <nav>, <section>, <article>, <aside> та <footer>, дозволяє створювати логічно організовані сторінки, що полегшує розуміння їх змісту як для користувачів, так і для пошукових систем. Це сприяє покращенню SEO (Search Engine Optimization) та доступності веб-сторінок для людей з обмеженими можливостями.

HTML тісно інтегрований з іншими веб-технологіями, такими як CSS (Cascading Style Sheets) та JavaScript. CSS використовується для стилізації та оформлення HTML-елементів, що дозволяє створювати привабливі та професійно виглядаючі веб-сторінки. JavaScript додає інтерактивність та

динамічність, дозволяючи створювати складні функціональні можливості, такі як форми з перевіркою даних, анімації, динамічне завантаження контенту та багато іншого. Разом HTML, CSS та JavaScript створюють потужну тріаду технологій, які є основою сучасної веб-розробки.

Однією з важливих особливостей HTML є його крос-браузерна сумісність. Веб-сторінки, створені на HTML, можуть бути відображені у різних веб-браузерах, таких як Google Chrome, Mozilla Firefox, Safari, Microsoft Edge та інші. Хоча можуть виникати деякі відмінності у відображенні через специфічні особливості кожного браузера, HTML забезпечує базову сумісність та коректне відображення контенту на всіх основних платформах.

HTML також підтримує інтеграцію мультимедійного контенту, що дозволяє додавати на веб-сторінки відео, аудіо, графіку та інші медіаелементи. Використання тегів <video> та <audio> дозволяє вбудовувати мультимедійні файли без необхідності використання додаткових плагінів або програмного забезпечення. Це робить веб-сторінки більш інтерактивними та привабливими для користувачів.

Завдяки розвитку стандартів HTML5, можливості HTML значно розширились. HTML5 включає нові теги та атрибути, які дозволяють створювати більш складні та функціональні веб-сторінки. Наприклад, теги <canvas> та <svg> дозволяють створювати графіку та анімацію без використання зовнішніх бібліотек. API для роботи з геолокацією, локальним сховищем даних та іншими функціями дозволяють створювати більш інтерактивні та функціональні веб-додатки.

Порівнюючи HTML з іншими мовами розмітки, такими як XML (eXtensible Markup Language) або XHTML (eXtensible HyperText Markup Language), можна виділити кілька ключових відмінностей. По-перше, HTML призначений спеціально для веб-розробки та забезпечує більш простий та інтуїтивно зрозумілий синтаксис. По-друге, HTML має широку підтримку з

боку веб-браузерів та інструментів для розробки, що робить його більш зручним для використання у веб-проектах. XHTML, з іншого боку, є більш строгим варіантом HTML, який дотримується правил XML, що робить його менш гнучким, але більш структурованим.

Однією з важливих сфер застосування HTML є створення адаптивних веб-сторінок, які можуть автоматично підлаштовуватися під різні розміри екранів та пристроїв. Використання медіа-запитів у поєднанні з CSS дозволяє створювати сторінки, які виглядають та працюють добре як на настільних комп'ютерах, так і на мобільних пристроях. Це особливо важливо в сучасному світі, де все більше користувачів отримують доступ до веб-контенту через смартфони та планшети.

Ще однією важливою особливістю HTML є його підтримка форм, що дозволяє створювати інтерактивні елементи, такі як текстові поля, кнопки, перемикачі, випадаючі списки та інші елементи управління. Форми є важливою складовою більшості веб-додатків, дозволяючи користувачам взаємодіяти з веб-сторінками, вводити та надсилати дані. Використання HTML-форм у поєднанні з JavaScript дозволяє створювати складні та інтерактивні інтерфейси користувача.

HTML також підтримує гіпертекстові посилання, що дозволяє створювати зв'язки між різними сторінками та ресурсами в Інтернеті. Використання тегу <a> дозволяє створювати посилання на інші веб-сторінки, файли, зображення або інші ресурси. Гіпертекстові посилання є основою веб-навігації, забезпечуючи користувачам можливість переходити між різними частинами веб-сайту або Інтернету.

У підсумку, HTML є основою сучасної веб-розробки, забезпечуючи структуру, семантику та можливості інтеграції для створення функціональних та привабливих веб-сторінок. Завдяки своїй простоті, гнучкості та потужності, HTML залишається незамінним інструментом для розробників, які прагнуть створювати високоякісні веб-додатки. Незалежно від рівня досвіду чи типу

проекту, HTML надає всі необхідні інструменти для успішної реалізації будь-якої ідеї в Інтернеті.

CSS (Cascading Style Sheets) є мовою стилів, яка використовується для оформлення та презентації HTML-документів. CSS дозволяє веб-розробникам відокремлювати структуру веб-сторінки (створену за допомогою HTML) від її зовнішнього вигляду, що забезпечує більш гнучкий та ефективний підхід до веб-дизайну. Вперше представлений у 1996 році, CSS став невід'ємною частиною сучасної веб-розробки, дозволяючи створювати привабливі, адаптивні та користувацькі веб-сторінки.

Однією з головних переваг CSS є його здатність забезпечувати чітке та структуроване оформлення веб-сторінок. Використовуючи CSS, розробники можуть визначати стилі для різних HTML-елементів, таких як заголовки, параграфи, посилання, зображення тощо. Стилі можуть включати властивості, що визначають кольори, шрифти, розміри, відступи, поля, вирівнювання та багато інших аспектів зовнішнього вигляду елементів. Це дозволяє створювати послідовний та професійний дизайн, який легко підтримувати та змінювати.

CSS також забезпечує каскадність стилів, що означає, що стилі можуть успадковуватися та перевизначатися. Це дозволяє розробникам визначати загальні стилі для всього документа, а потім уточнювати їх для окремих елементів або груп елементів. Каскадність забезпечує гнучкість та зручність управління стилями, дозволяючи створювати складні та багаторівневі дизайни з мінімальними зусиллями.

Ще однією важливою особливістю CSS є можливість створення адаптивних веб-дизайнів. Використовуючи медіа-запити, розробники можуть визначати стилі, які змінюються залежно від характеристик пристрою, таких як ширина або висота екрана, орієнтація (пейзаж або портрет) та роздільна здатність. Це дозволяє створювати веб-сторінки, які виглядають та працюють добре на різних пристроях, включаючи настільні комп'ютери, планшети та

смартфони. Адаптивний дизайн забезпечує оптимальний користувацький досвід, незалежно від того, на якому пристрої переглядається веб-сторінка.

CSS також підтримує анімації та переходи, що дозволяє додавати динамічність та інтерактивність до веб-сторінок. Використовуючи властивості анімації, розробники можуть створювати складні анімаційні ефекти, які можуть бути застосовані до будь-яких CSS-властивостей. Переходи дозволяють плавно змінювати значення властивостей при взаємодії користувача з веб-сторінкою, що забезпечує більш привабливий та інтуїтивно зрозумілий користувацький досвід.

Порівнюючи CSS з іншими мовами стилів, такими як LESS або SASS (Syntactically Awesome Stylesheets), можна виділити кілька ключових відмінностей. LESS та SASS є препроцесорами CSS, що дозволяють використовувати розширені можливості, такі як змінні, вкладені правила, міксіни та функції, які не підтримуються стандартним CSS. Це забезпечує більш гнучкий та ефективний підхід до написання стилів, особливо для великих та складних проектів. Проте, ці препроцесори вимагають додаткового кроку компіляції, що може ускладнити процес розробки. CSS, з іншого боку, є стандартною мовою стилів, яка підтримується безпосередньо браузерами без потреби у додатковій обробці.

CSS також забезпечує потужні можливості для роботи з макетами за допомогою технологій, таких як Flexbox та CSS Grid. Flexbox дозволяє створювати гнучкі та адаптивні макети з використанням контейнерів та елементів, які автоматично вирівнюються та розподіляються відповідно до доступного простору. CSS Grid надає більш потужний та гнучкий інструмент для створення двовимірних макетів, що дозволяє розробникам точно визначати розташування елементів у сітці. Обидві технології значно спрощують процес створення складних макетів та забезпечують високу гнучкість у розробці дизайну.

Ще однією важливою особливістю CSS є його здатність до повторного використання стилів. Використовуючи класи та ідентифікатори, розробники можуть визначати стилі, які можуть бути застосовані до багатьох елементів на веб-сторінці. Це дозволяє уникнути дублювання коду та забезпечує послідовність стилів по всьому проекту. Крім того, розробники можуть використовувати зовнішні CSS-файли, які можуть бути підключені до кількох HTML-документів, що спрощує управління стилями для великих веб-сайтів.

CSS також підтримує змінні (CSS Variables), що дозволяє визначати значення властивостей, які можуть бути використані повторно в різних частинах стилю. Це забезпечує більшу гнучкість та зручність у управлінні стилями, дозволяючи швидко змінювати значення властивостей у одному місці та автоматично застосовувати ці зміни по всьому проекту. Змінні CSS значно спрощують процес розробки та підтримки стилів, особливо для великих та складних проектів.

Ще однією важливою перевагою CSS є його підтримка візуальних ефектів та стилів, що дозволяє створювати привабливі та естетично приємні веб-сторінки. Використовуючи властивості, такі як тіні (box-shadow та text-shadow), прозорість (opacity), градієнти (linear-gradient та radial-gradient), трансформації (transform), розробники можуть додавати глибину, динамічність та індивідуальність до дизайну своїх веб-сторінок. Це забезпечує більш привабливий та професійний вигляд, який може залучити та утримати увагу користувачів.

У підсумку, CSS є потужною та гнучкою мовою стилів, яка забезпечує всі необхідні інструменти для створення привабливих, адаптивних та користувацьких веб-сторінок. Завдяки своїй простоті, гнучкості та потужності, CSS залишається незамінним інструментом для веб-розробників, які прагнуть створювати високоякісні веб-додатки. Незалежно від рівня досвіду чи типу

проекту, CSS надає всі необхідні інструменти для успішної реалізації будь-якої ідеї в Інтернеті.

3.2 Розробка основних алгоримтів

Процес розробки основних алгоритмів для блогової системи з інтеграцією штучного інтелекту включає створення низки компонентів, кожен з яких відіграє ключову роль у функціонуванні додатку. Важливість цих алгоритмів полягає у забезпеченні ефективності, безпеки та зручності користувачів платформи.

Алгоритм автентифікації користувачів

Алгоритм автентифікації користувачів є фундаментальним для забезпечення безпеки додатку. Він включає перевірку даних користувача, які вводяться при вході в систему. Процес розпочинається з форми входу, де користувачі вводять свої імена користувачів та паролі. Введені дані надсилаються на сервер через захищене з'єднання. На сервері відбувається пошук у базі даних для знаходження відповідності введеним користувачем даним. Якщо збіг знайдено, користувачу надається доступ до свого аккаунту. Цей процес також включає механізми забезпечення безпеки, такі як хешування паролів, що запобігає можливості витоку даних.

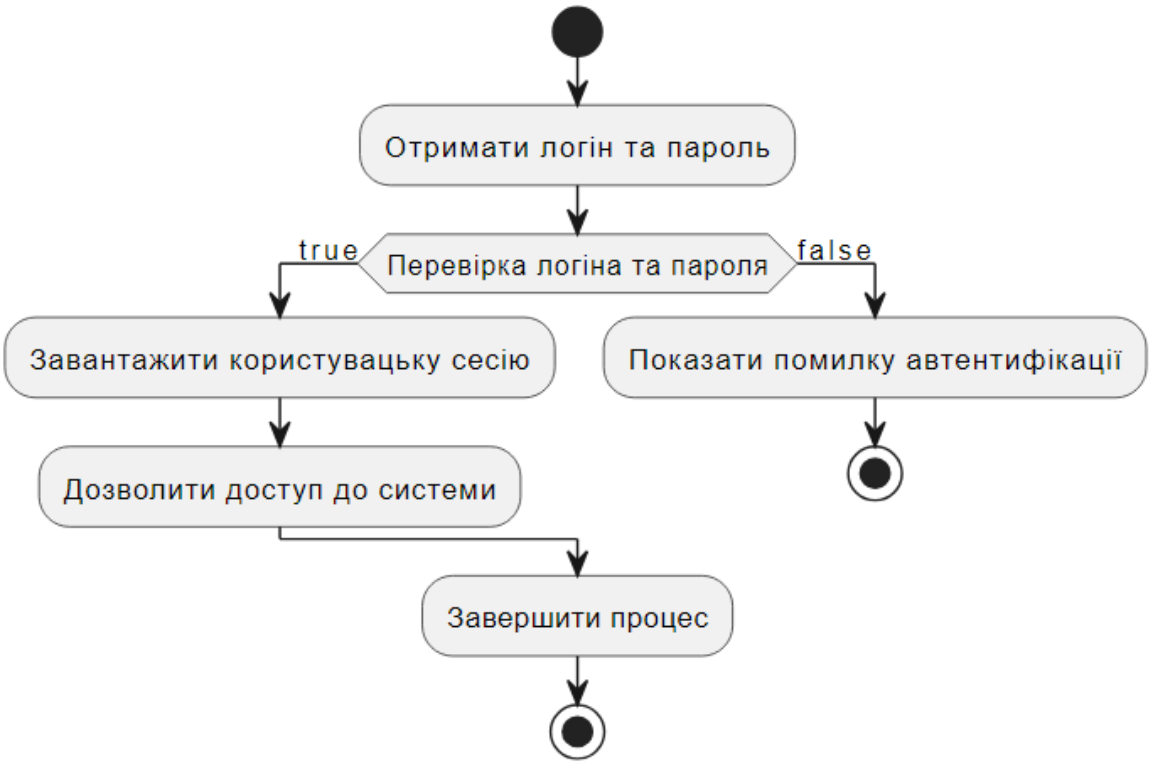

Рисунок 3.1 — Блок-схема алгоритму автентифыкацыъ

Алгоритми управління контентом

Алгоритми управління контентом відіграють ключову роль у дозволянні користувачам створювати, редагувати та видаляти контент. Коли користувач створює новий пост, введені дані обробляються сервером, перевіряються на коректність та зберігаються у базі даних. Подібні процеси використовуються для редагування та видалення постів. Кожна з цих операцій вимагає ретельної валідації вхідних даних для запобігання помилок та забезпечення консистентності даних.

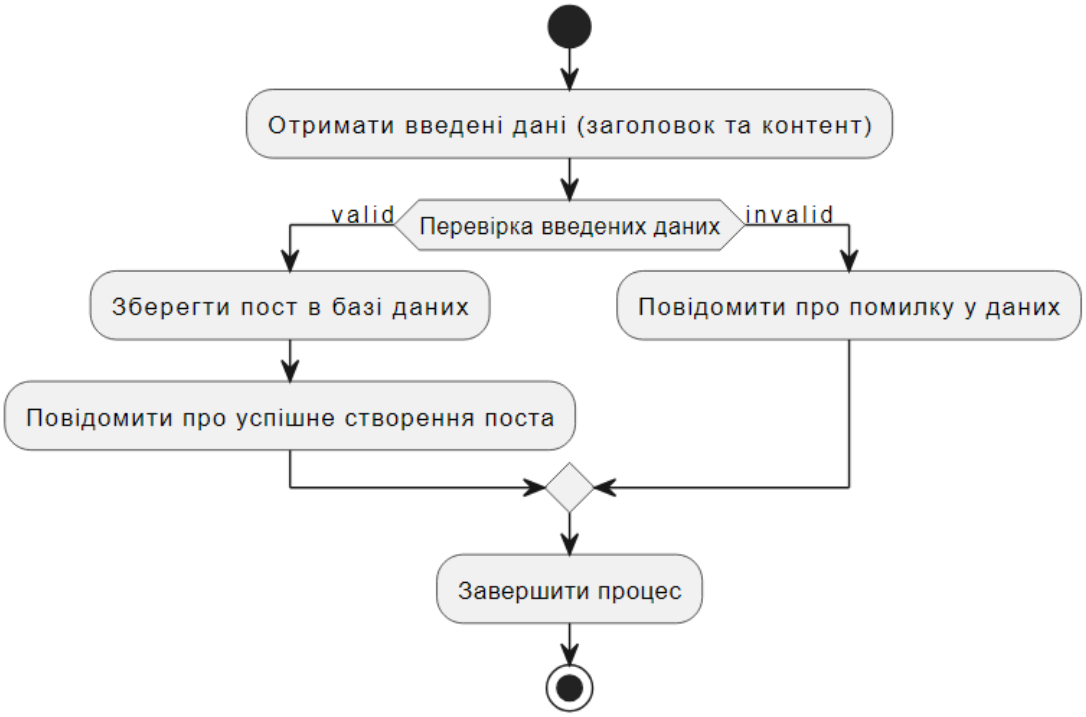

Рисунок 3.2 — Блок-схема алгоритму додавання поста

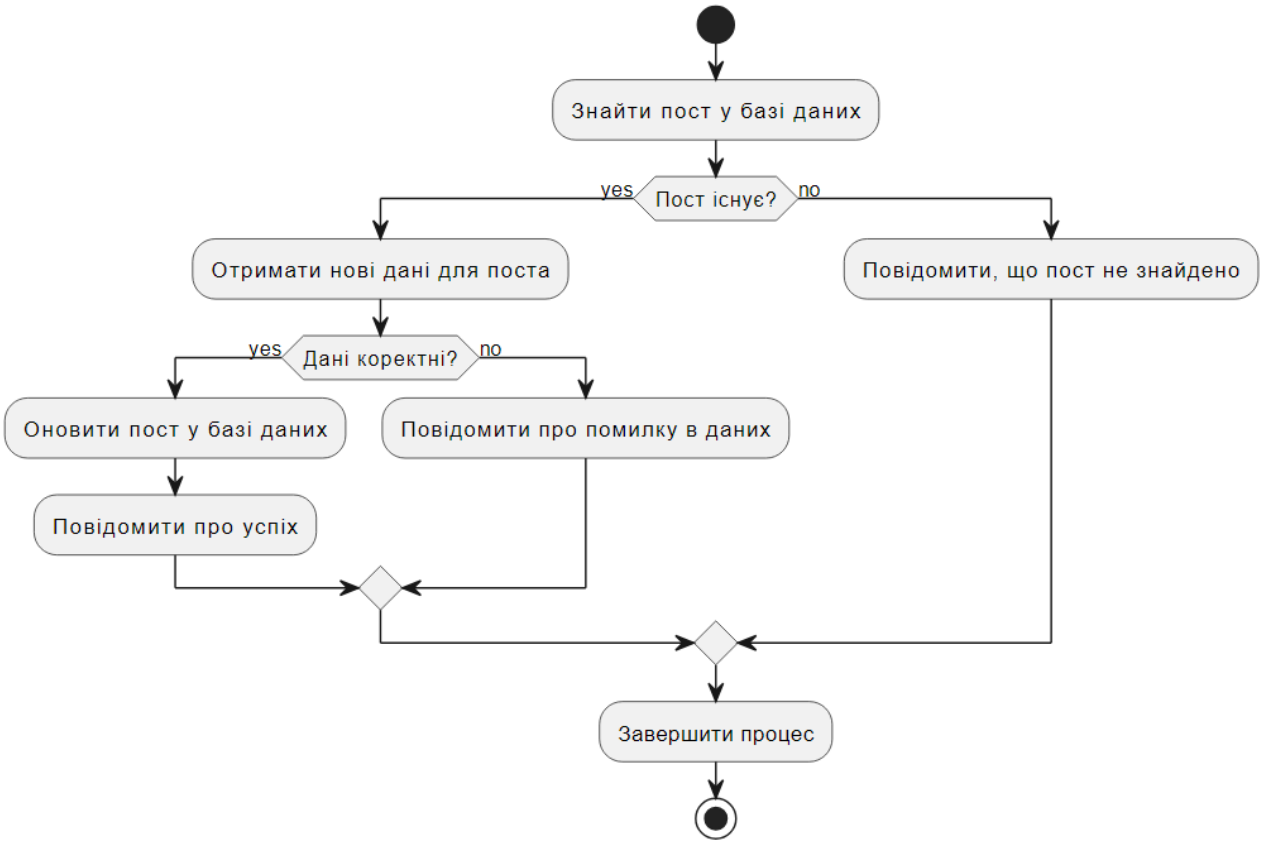

Рисунок 3.3 — Блок-схема алгоритму редагування поста

Інтеграція з штучним інтелектом для генерації контенту

Однією з унікальних особливостей розглядуваного додатку є його здатність інтегрувати штучний інтелект для генерації контенту. Цей процес починається з введення користувачем короткого опису того, що він бажає побачити у своєму пості. Опис надсилається до зовнішнього API штучного інтелекту, який обробляє запит і генерує текст на основі наданої інформації. Згенерований текст повертається користувачу для перегляду і, при необхідності, корекції. Такий підхід значно спрощує процес створення контенту, роблячи платформу більш привабливою для користувачів, які можуть не мати великого досвіду в написанні тексти. Завдяки інтеграції штучного інтелекту, користувачі можуть легко створювати високоякісний контент без необхідності глибокого знання теми або вміння письменно викладати свої думки. Це зменшує бар'єри

для нових користувачів та стимулює їх активніше використовувати платформу для вираження своїх ідей та спілкування з аудиторією.

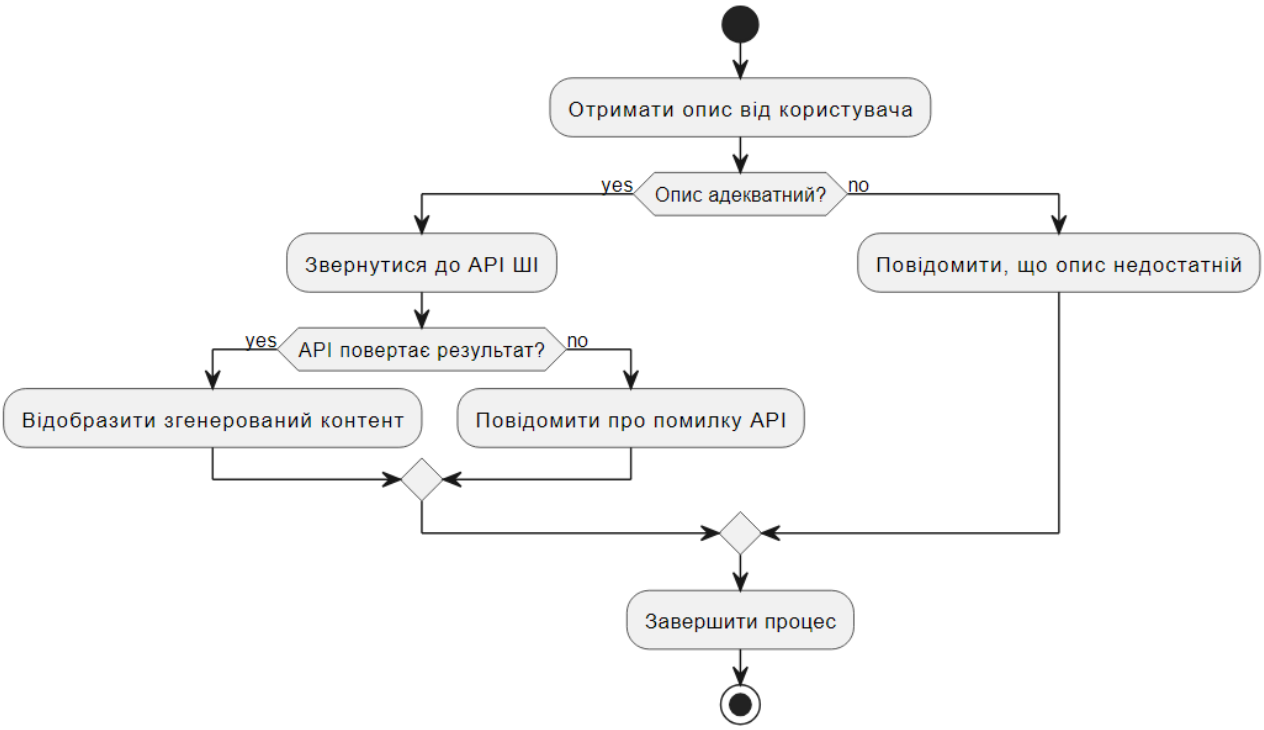

Рисунок 3.4 — Блок-схема алгоритму генерації поста

3.3 Розробка інтерфейсу користувача

На головній сторінці відображається список усіх постів з можливістю входу через кнопку "Login". Перед користувачем з'являється простий інтерфейс, де наразі відсутні пости, і підпис "No posts yet!" вказує на те, що блог поки що порожній.

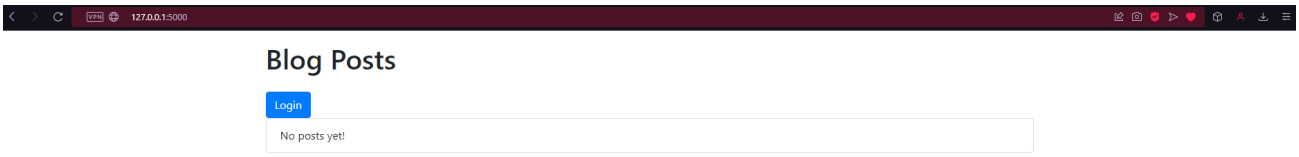

Рисунок 3.5 — Головна сторінка

Сторінка входу містить поля для введення імені користувача та пароля, а також кнопку для підтвердження входу. Це стандартна форма входу, яка вимагає від користувачів ввести свої облікові дані для отримання доступу до додаткових функцій блогу.

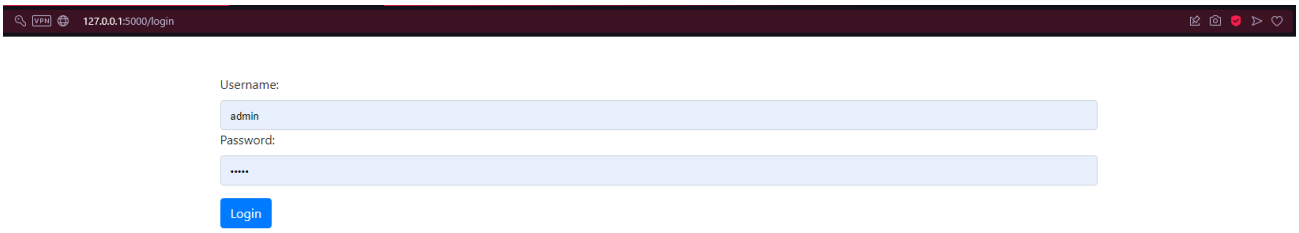

Рисунок 3.6 —Сторінка входу

Після входу в систему, інтерфейс головної сторінки змінюється, додавши кнопки "Add New Post" та "Generate New Article". Це дозволяє користувачам активно участвувати у створенні контенту та використанні можливостей штучного інтелекту для генерації текстів.

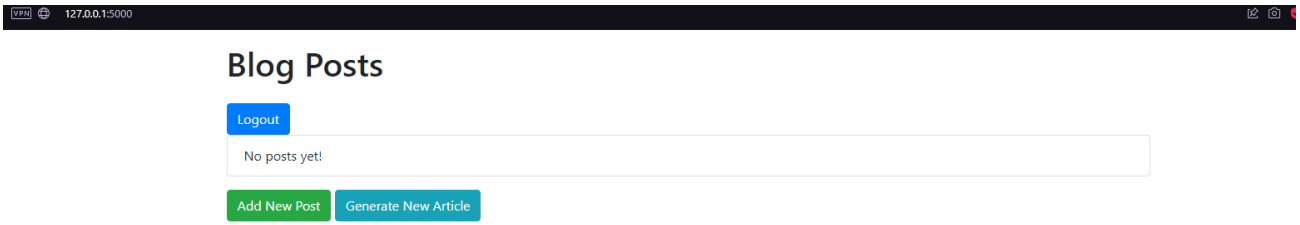

Рисунок 3.7 — Головна сторінка (після входу)

На цій сторінці користувачі можуть додавати нові пости, вводячи заголовок та вміст посту у відповідні поля. Кнопка "Submit" слугує для надсилання введених даних, що дозволяє публікувати пост на блозі.

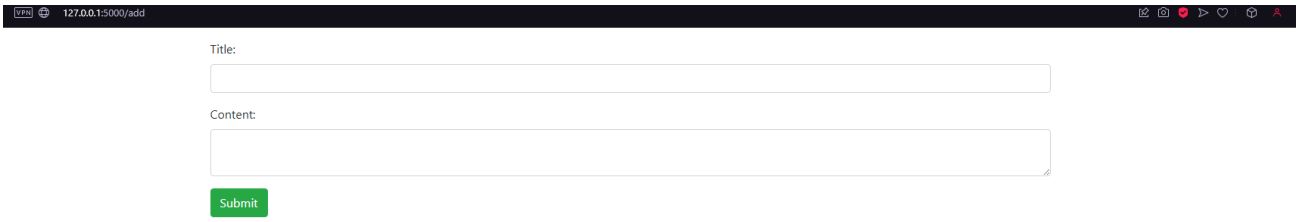

Рисунок 3.8 — Сторінка додавання поста

На сторінці для опису генерації користувачі мають можливість ввести опис, на основі якого штучний інтелект згенерує статтю. Це надає користувачам унікальну можливість створювати контент без потреби безпосереднього його написання.

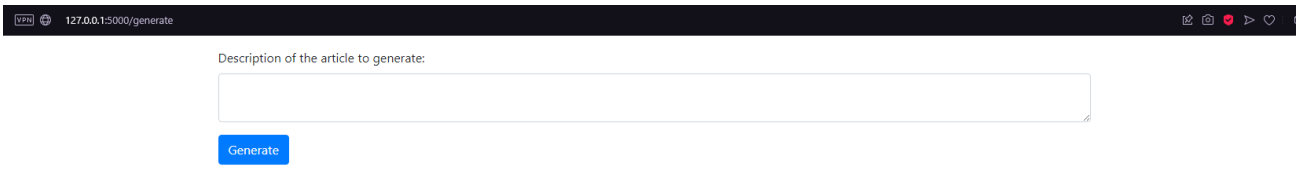

Рисунок 3.9 — Сторінка опису генерації

Після натискання кнопки "Generate" на попередній сторінці, система переходить на сторінку генерації, де користувачі можуть переглянути згенерований контент. Текст відображається у великому текстовому полі, дозволяючи користувачам переглядати та копіювати згенерований контент для подальшого використання.

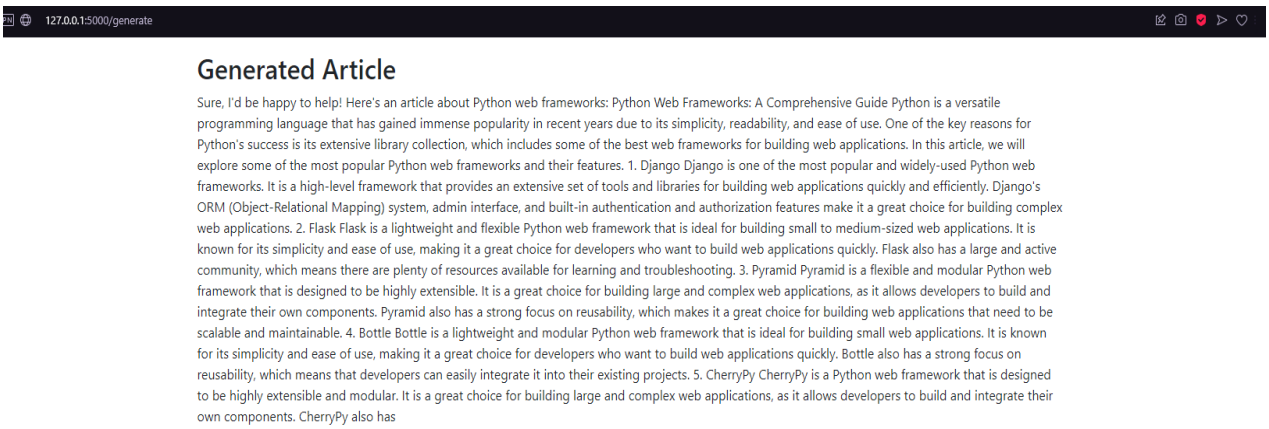

Рисунок 3.10 — Сторінка генерації

Цей інтерфейс системи є інтуїтивно зрозумілим і забезпечує користувачам легкий доступ до всіх необхідних функцій для ефективного блогінгу та взаємодії з штучним інтелектом.

3.4 Тестування системи

Тестування програмного забезпечення — це критично важливий етап розробки ПЗ, який полягає в перевірці та валідації функціональності, продуктивності, безпеки та інших важливих параметрів системи перед її запуском. Цей процес дозволяє забезпечити, що програмне забезпечення відповідає усім заздалегідь визначеним вимогам та очікуванням користувачів, а також знижує ризики потенційних збоїв та помилок у роботі продукту. Тестування допомагає виявити помилки та недоліки на ранніх стадіях розробки, що істотно спрощує процес їх усунення та зменшує загальні витрати на розробку.

Тестування може бути ручним або автоматизованим. Ручне тестування виконується людьми, які вручну перевіряють різні аспекти системи, щоб забезпечити її правильну поведінку. Автоматизоване тестування використовує спеціалізоване програмне забезпечення для створення тестів, які можна виконувати багаторазово без додаткової участі тестувальників.

Типи тестування включають, але не обмежуються:

- Функціональне тестування: перевірка специфічних вимог до поведінки системи.
- Навантажувальне тестування: визначення здатності системи витримувати велике навантаження.
- Тестування безпеки: перевірка на вразливості та потенційні загрози.
- Інтеграційне тестування: перевірка взаємодій між різними компонентами системи.
- Тестування прийняття: забезпечення того, що система відповідає бізнес-вимогам і готова до впровадження.

Таблиця 3.1

Тестування системи

Тип тестування	Опис тесту	Метод тестування	Відповідальний	Очікуваний результат	Фактичний результат	Статус
Тестування авторизації	Перевірка системи авторизації за допомогою валідних та невалідних даних.	Ручне	QA Engineer	Авторизація успішна за валідних даних, помилка за невалідних.	Як очікувалося	Пройдено
Тестування додавання посту	Перевірка здатності системи приймати та зберігати нові пости. Перевірка реакції на пусті поля.	Автоматизоване	QA Engineer	Пости додаються коректно, пусті поля відхиляються.	Як очікувалося	Пройдено
Тестування видалення посту	Тестування функціональності видалення постів.	Ручне	QA Engineer	Пости видаляються без помилок, система оновлюється.	Як очікувалося	Пройдено
Тестування	Верифікація	Автоматизо	Developer	Контент	Як	Пройде

генерації контенту	алгоритмів ШІ для створення контенту на основі введеного опису.	ване		генерується точно і відповідає опису.	очікувалося	но
Навантажувальне тестування	Симуляція великої кількості одночасних користувачів.	Автоматизоване	QA Team	Система стабільно працює під високим навантаженням.	Як очікувалося	Пройдено
Тестування безпеки	Перевірка на SQL ін'єкції, XSS атаки та інші веб-вразливості.	Автоматизоване	Security Team	Система захищена від основних веб-вразливостей.	Як очікувалося	Пройдено
Інтеграційне тестування	Перевірка взаємодії між авторизацією, управлінням контентом та модулем ШІ.	Автоматизоване	QA Engineer	Всі модулі ефективно взаємодіють без помилок.	Як очікувалося	Пройдено
Тестування відповідності	Перевірка відповідності системи вимогам та специфікаціям.	Ручне	Compliance Team	Система повністю відповідає всім встановленим вимогам.	Як очікувалося	Пройдено

Тестування UI/UX	Оцінка зручності інтерфейсу користувача та загального досвіду використання системи.	Ручне	UX Designer	Інтерфейс є інтуїтивно зрозумілим та зручним для користувачів.	Як очікувалося	Пройдено

Завдяки детальному тестуванню, було підтверджено, що система блогу з інтеграцією штучного інтелекту функціонує стабільно та ефективно у всіх аспектах її роботи. Всі тести завершилися успішно, що свідчить про високу якість розробленого продукту. Система готова до впровадження та використання в продуктивному середовищі, забезпечуючи користувачам надійний та безпечний досвід користування. Такий підхід до тестування гарантує, що потенційні проблеми виявлені та виправлені до запуску, знижуючи ризики та забезпечуючи високу задоволеність користувачів.

ВИСНОВКИ

Розробка такої системи є актуальною завдяки зростаючій потребі у високоякісних цифрових інструментах для контент-менеджменту, які можуть ефективно задовольнити потреби як індивідуальних блогерів, так і великих медіаорганізацій.

Проект був успішно реалізований з урахуванням всіх встановлених технічних та бізнес-вимог. Блогова платформа обладнана інтуїтивно зрозумілим інтерфейсом, що забезпечує зручність використання як новачками, так і досвідченими користувачами. Інтеграція з штучним інтелектом дозволяє автоматично генерувати контент високої якості, що значно скорочує час на його створення та редагування.

Система пройшла ретельне тестування, що включало функціональне, навантажувальне, безпекове та інтеграційне тестування. Результати тестування підтвердили високу продуктивність, надійність та безпеку системи. Тестування виявило декілька незначних помилок, які були швидко виправлені, що підтвердило ефективність використаних методів розробки та готовність системи до масштабування.

Розроблена платформа має велике практичне значення, оскільки забезпечує користувачам потужний інструмент для ефективного блогінгу та управління контентом. Інтеграція штучного інтелекту збільшує продуктивність користувачів, дозволяючи їм сконцентруватися на стратегічних аспектах створення контенту, в той час як рутинні завдання автоматизуються. Такий підхід може сприяти збільшенню кількості якісного контенту без додаткових витрат часу та ресурсів.

На основі результатів розробки та тестування системи можна очікувати, що її впровадження дозволить значно покращити якість та ефективність управління контентом в блогах. Очікується, що платформа знайде широке

застосування серед блогерів, журналістів та медіакомпаній, які прагнуть максимізувати вплив свого контенту в цифровому просторі.

У висновку, дана робота демонструє успішне впровадження сучасних технологій в розробку веб-платформ, забезпечуючи користувачам ефективні та інноваційні інструменти для ведення блогу. Результати цього проекту мають значний вплив на подальше розвиток цифрових комунікацій та взаємодії з аудиторією.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. "Better Language Models and Their Implications". OpenAI. 2019-02-14. Archived from the original on 2020-12-19. Retrieved 2019-08-25.
2. Bowman, Samuel R. (2023). "Eight Things to Know about Large Language Models". arXiv:2304.00612 [cs.CL].
3. Brown, Tom B.; Mann, Benjamin; Ryder, Nick; Subbiah, Melanie; Kaplan, Jared; Dhariwal, Prafulla; Neelakantan, Arvind; Shyam, Pranav; Sastry, Girish; Askell, Amanda; Agarwal, Sandhini; Herbert-Voss, Ariel; Krueger, Gretchen; Henighan, Tom; Child, Rewon; Ramesh, Aditya; Ziegler, Daniel M.; Wu, Jeffrey; Winter, Clemens; Hesse, Christopher; Chen, Mark; Sigler, Eric; Litwin, Mateusz; Gray, Scott; Chess, Benjamin; Clark, Jack; Berner, Christopher; McCandlish, Sam; Radford, Alec; Sutskever, Ilya; Amodei, Dario (Dec 2020). Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.F.; Lin, H. (eds.). "Language Models are Few-Shot Learners" (PDF). Advances in Neural Information Processing Systems. 33. Curran Associates, Inc.: 1877–1901.
4. Fathallah, Nadeen; Das, Arunav; De Giorgis, Stefano; Poltronieri, Andrea; Haase, Peter; Kovriguina, Liubov (2024-05-26). NeOn-GPT: A Large Language Model-Powered Pipeline for Ontology Learning (PDF). Extended Semantic Web Conference 2024. Hersonissos, Greece.
5. Manning, Christopher D. (2022). "Human Language Understanding & Reasoning". Daedalus. 151 (2): 127–138. doi:10.1162/daed_a_01905. S2CID 248377870.
6. Kilgarriff, Adam; Grefenstette, Gregory (September 2003). "Introduction to the Special Issue on the Web as Corpus". Computational Linguistics. 29 (3): 333–347. doi:10.1162/089120103322711569. ISSN 0891-2017.
7. Banko, Michele; Brill, Eric (2001). "Scaling to very very large corpora for natural language disambiguation". Proceedings of the 39th Annual Meeting on

Association for Computational Linguistics - ACL '01. Morristown, NJ, USA: Association for Computational Linguistics. doi:10.3115/1073012.1073017.

8. Resnik, Philip; Smith, Noah A. (September 2003). "The Web as a Parallel Corpus". Computational Linguistics. 29 (3): 349–380. doi:10.1162/089120103322711578. ISSN 0891-2017.
9. Halevy, Alon; Norvig, Peter; Pereira, Fernando (March 2009). "The Unreasonable Effectiveness of Data". IEEE Intelligent Systems. 24 (2): 8–12. doi:10.1109/MIS.2009.36. ISSN 1541-1672.
10. Vaswani, Ashish; Shazeer, Noam; Parmar, Niki; Uszkoreit, Jakob; Jones, Llion; Gomez, Aidan N; Kaiser, Łukasz; Polosukhin, Illia (2017). "Attention is All you Need" (PDF). Advances in Neural Information Processing Systems. 30. Curran Associates, Inc.
11. Bahdanau, Dzmitry; Cho, Kyunghyun; Bengio, Yoshua (2014). "Neural Machine Translation by Jointly Learning to Align and Translate". arXiv:1409.0473 [cs.CL].
12. Rogers, Anna; Kovaleva, Olga; Rumshisky, Anna (2020). "A Primer in BERTology: What We Know About How BERT Works". Transactions of the Association for Computational Linguistics. 8: 842–866. arXiv:2002.12327. doi:10.1162/tacl_a_00349. S2CID 211532403.
13. Hern, Alex (14 February 2019). "New AI fake text generator may be too dangerous to release, say creators". The Guardian. Retrieved 20 January 2024.
14. "ChatGPT a year on: 3 ways the AI chatbot has completely changed the world in 12 months". Euronews. November 30, 2023. Retrieved January 20, 2024.
15. Heaven, Will (March 14, 2023). "GPT-4 is bigger and better than ChatGPT—but OpenAI won't say why". MIT Technology Review. Retrieved January 20, 2024.
16. "Parameters in notable artificial intelligence systems". ourworldindata.org. November 30, 2023. Retrieved January 20, 2024.

17. "LMSYS Chatbot Arena Leaderboard". huggingface.co. Retrieved June 12, 2024.
18. Peng, Bo; et al. (2023). "RWKV: Reinventing RNNS for the Transformer Era". arXiv:2305.13048 [cs.CL].
19. Merritt, Rick (2022-03-25). "What Is a Transformer Model?". NVIDIA Blog. Retrieved 2023-07-25.
20. Gu, Albert; Dao, Tri (2023-12-01), Mamba: Linear-Time Sequence Modeling with Selective State Spaces, arXiv:2312.00752
21. Yennie Jun (2023-05-03). "All languages are NOT created (tokenized) equal". Language models cost much more in some languages than others. Archived from the original on 2023-08-17. Retrieved 2023-08-17. In other words, to express the same sentiment, some languages require up to 10 times more tokens.
22. Petrov, Aleksandar; Malfa, Emanuele La; Torr, Philip; Bibi, Adel (June 23, 2023). "Language Model Tokenizers Introduce Unfairness Between Languages". NeurIPS. arXiv:2305.15425 – via openreview.net.
23. "OpenAI API". platform.openai.com. Archived from the original on April 23, 2023. Retrieved 2023-04-30.
24. Paaß, Gerhard; Giesselbach, Sven (2022). "Pre-trained Language Models". Foundation Models for Natural Language Processing. Artificial Intelligence: Foundations, Theory, and Algorithms. pp. 19–78. doi:10.1007/978-3-031-23190-2_2. ISBN 9783031231902. Retrieved 3 August 2023.
25. Petrov, Aleksandar; Emanuele La Malfa; Torr, Philip H. S.; Bibi, Adel (2023). "Language Model Tokenizers Introduce Unfairness Between Languages". arXiv:2305.15425 [cs.CL].
26. Dodge, Jesse; Sap, Maarten; Marasović, Ana; Agnew, William; Ilharco, Gabriel; Groeneveld, Dirk; Mitchell, Margaret; Gardner, Matt (2021).

"Documenting Large Webtext Corpora: A Case Study on the Colossal Clean Crawled Corpus". arXiv:2104.08758 [cs.CL].

27. Lee, Katherine; Ippolito, Daphne; Nystrom, Andrew; Zhang, Chiyuan; Eck, Douglas; Callison-Burch, Chris; Carlini, Nicholas (May 2022). "Deduplicating Training Data Makes Language Models Better" (PDF). Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics. 1: Long Papers: 8424–8445. doi:10.18653/v1/2022.acl-long.577.
28. Li, Yuanzhi; Bubeck, Sébastien; Eldan, Ronen; Del Giorno, Allie; Gunasekar, Suriya; Lee, Yin Tat (2023-09-11), Textbooks Are All You Need II: phi-1.5 technical report, arXiv:2309.05463
29. Lin, Zhenghao; Gou, Zhibin; Gong, Yeyun; Liu, Xiao; Shen, Yelong; Xu, Ruochen; Lin, Chen; Yang, Yujiu; Jiao, Jian (2024-04-11). "Rho-1: Not All Tokens Are What You Need". arXiv:2404.07965 [cs.CL].
30. Brown, Tom B.; et al. (2020). "Language Models are Few-Shot Learners". arXiv:2005.14165 [cs.CL].
31. Abdin, Marah; Jacobs, Sam Ade; Awan, Ammar Ahmad; Aneja, Jyoti; Awadallah, Ahmed; Awadalla, Hany; Bach, Nguyen; Bahree, Amit; Bakhtiari, Arash (2024-04-23). "Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone". arXiv:2404.14219 [cs.CL].

ДОДАТОК А
ЛІСТИНГ ПРОГРАМНОГО КОДУ

```
import os

from flask import Flask, render_template, request, redirect, url_for, session
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager, UserMixin, login_user, logout_user, login_required, current_user
import replicate
os.environ["REPLICATE_API_TOKEN"] = "r8_MQQeA5baJW7exVdiQZcYKtTrMTvSkFc2T9Dil"

app = Flask(__name__)
app.config['SECRET_KEY'] = 'your-secret-key'
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///blog.db'
db = SQLAlchemy(app)

login_manager = LoginManager()
login_manager.init_app(app)

class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(100), unique=True)

class Post(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(100))
```

```
    content = db.Column(db.Text)

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

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

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        if username == 'admin' and password == 'admin':
            user = User.query.filter_by(username=username).first()
            if not user:
                user = User(username=username)
                db.session.add(user)
                db.session.commit()
            login_user(user)
            return redirect(url_for('index'))
        return 'Invalid credentials'
    return render_template('login.html')

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

```
def logout():
    logout_user()
    return redirect(url_for('index'))

@app.route('/add', methods=['GET', 'POST'])
@login_required
def add_post():
    if request.method == 'POST':
        title = request.form['title']
        content = request.form['content']
        new_post = Post(title=title, content=content)
        db.session.add(new_post)
        db.session.commit()
        return redirect(url_for('index'))
    return render_template('add_post.html')

@app.route('/delete/<int:post_id>')
@login_required
def delete_post(post_id):
    post = Post.query.get_or_404(post_id)
    db.session.delete(post)
    db.session.commit()
    return redirect(url_for('index'))

@app.route('/generate', methods=['GET', 'POST'])
@login_required
def generate_content():
    if request.method == 'POST':
```

```
            description = request.form['description']
            generated_text = generatePost(description)
            return render_template('generated_post.html', text=generated_text)
        return render_template('generate_form.html')

    def generatePost(prompt_input):
        pre_prompt = ("You are a useful assistant to the person running his blog. Your task is to generate short articles based on description."
                      "Your answer must contain exclusively the text of the article according to the given description. "
                      "Return only article text. Dont ask questions. Dont add additional comments."
                      "You do not respond as 'User' or pretend to be 'User'. You only respond once as 'Assistant'.")
        output = replicate.run(
            'a16z-infra/llama13b-v2-chat:df7690f1994d94e96ad9d568eac121aecf50684a0b0963b25a41cc40061269e5',  # LLM model
            input={"prompt": f"{pre_prompt} {prompt_input} Assistant: ",  # Prompts
                   "temperature": 0.1, "top_p": 0.9, "max_length": 1024, "repetition_penalty": 1})  # Model parameters
        full_response = ""

        for item in output:
            full_response += item

        return full_response
```

```
with app.app_context():
    db.create_all()

if __name__ == '__main__':

    app.run(debug=True)
```