Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інтелектуальних інформаційних технологій та автоматизації
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Інформаційна технологія організації блог-систем»

Виконав: студент 2-го курсу, групи 1КН-21м
спеціальності 122 «Комп'ютерні науки»
(шифр і назва напряму підготовки, спеціальності)
Жданов В.В.
(прізвище та ініціали)

Керівник: д.т.н., професор каф. КН
Яровий А.А.
(прізвище та ініціали)
«15» 12 2022 р.

Опонент: д.т.н., професор каф. АІІТ
Квєтний Р.Н.
(прізвище та ініціали)
«15» 12 2022 р.

Допущено до захисту
Завідувач кафедри КН
д.т.н., проф. Яровий А.А.
(прізвище та ініціали)
«16» 12 2022 р.

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра Комп'ютерних Наук
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма – «Системи штучного інтелекту»

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
Д.т.н., проф. Яровий А.А.
„14" 09 2022 року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Жданов Володимир Вікторович
(прізвище, ім'я, по батькові)

1. Тема роботи Інформаційна технологія організації блог-систем

керівник роботи д.т.н., професор кафедри КН Яровий А.А.
затверджені наказом вищого навчального закладу від "14" 09. 2022 року №203.
2. Строк подання студентом роботи 18 листопада 2022 року

3. Вихідні дані до роботи: мова програмування - об'єктно-орієнтована, браузерне середовище роботи додатку, кількість критеріїв класифікації – 7 од., мова текстів, що класифікуються – англійська, відповідність стандарту ODBS, мінімальна кількість символів у пості – 200 од., мінімальна кількість символів у коментарі – 10 од., мінімальна кількість користувачів – 100 чол.
4. Зміст текстової частини: вступ, аналіз сучасного стану розвитку систем організації блог-систем, обґрунтування вибору інструментів технічної реалізації, проектування інформаційної технології, програмна реалізація інформаційної технології, аналіз результатів тестування, розробка інструкції користувача, економічна частина, висновки, список використаних джерел, додатки
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): алгоритм інформаційної технології роботи інформаційної технології організації блог-систем, загальна структурна схема інформаційної технології організації блог-систем, структурна схема модуля автентифікації інформаційної технології організації блог-систем, діаграма класів інформаційної технології організації блог-систем, робочі вікна інформаційної технології організації блог-систем, результати тестування навантаження інформаційної технології організації блог-систем.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	вико прий
1-3	Яровий А.А., д.т.н., проф. каф. КН	14.09.2022	14.10.20
4	Нікіфорова Л. О., к. е. н., доц. каф. ЕПВМ	14.09.2022	12.12.2

7. Дата видачі завдання 14.09. 2022 року

КАЛЕНДАРНИЙ ПЛАН

№ етапу	Назва етапу	Термін виконання		Очікувані рез
		початок	кінець	
1	Аналіз методів та засобів організації блог-систем	14.09.2022	01.10.2022	Аналітичний о літературних д задачі дослідж розділ 1
2	Обґрунтування методу розв'язання задачі, обґрунтування вибору інструментів технічної реалізації	02.10.2022	16.10.2022	Моделі, розді
3	Проектування інтелектуальної системи. Програмна реалізація інтелектуальної системи. Аналіз результатів тестування	17.10.2022	07.11.2022	розділ 2, 3
4	Підготовка економічної частини	09.11.2022	21.11.2022	розділ 4
5	Апробація та/або впровадження результатів дослідження	23.11.2022	01.12.2022	тези доповіде впровадження
6	Оформлення пояснювальної записки, графічного матеріалу та презентації	02.12.2022	14.12.2022	Пояснювальна записка, графі матеріал, през

Студент _____ (підпис) Жданов В.В.

Керівник роботи _____ (підпис) Яровий А.А.

АНОТАЦІЯ

УДК 004.9

Жданов В.В. Інформаційна технологія організації блог-систем. Магістерська кваліфікаційна робота зі спеціальності 122 – Комп’ютерні науки, освітня програма – Системи штучного інтелекту. Вінниця: ВНТУ, 2022. 116 с.

На укр. мові. Бібліогр.: 37 назв; рис.: 29; табл. 17.

Дана магістерська кваліфікаційна робота присвячена розробці програмного забезпечення для організації блог-систем. Були розглянуті та проаналізовані існуючі програмні рішення і їх функціональні можливості, та обрано компромісний підхід. Було проаналізовано різні підходи до вирішення задачі організації блог-систем та обрано розробку користувальницького веб-сайту. Було проаналізовано різні підходи до процесу автентифікації та обрано JWT. Було спроектовано програму для організації блог-систем, написану мовою програмування TypeScript з використанням бібліотеки Preact для клієнтської частини, Node.js і Nest.js для серверної частини та СУБД MongoDB для управління базами даних.

Графічна частина складається з шести плакатів.

У економічному розділі розраховано суму витрат на розробку та виготовлення нового технічного рішення, яка складає 372767 гривень, спрогнозовано орієнтовану величину витрат по кожній з статей витрат, розраховано чистий прибуток, термін окупності витрат для виробника 0,76 роки та економічний ефект для споживача при використанні даної розробки.

Ключові слова: блог-система, веб-клієнт, JWT, клієнт-серверна архітектура.

ABSTRACT

Zhdanov V.V. Information technology of blog systems organization. Master's thesis in the specialty 122 - Computer sciences, educational program – Artificial intelligence systems. Vinnytsia: VNTU, 2022. 116 p.

In Ukrainian language. Bibliographer: 37 titles; fig .: 29; table 17.

This master's thesis is devoted to the development of software for blog systems organization. The existing software solutions for blog systems organization, the hybrid method of lifecycle process was chosen. Different approaches of blog systems organization were analyzed, and the compromisable solution, was substantiated. Different method of authentication were analysed and JWT was chosen. A program for blog systems organization for the presence of offensive statements, written in the TypeScript programming language using the Preact library for the client part, Node.js and Nest.js for the server part and MongoDB for managing the database.

The graphic part consists of six posters.

The economic section calculates the amount of costs for the development and manufacture of a new technical solution, which is 372767 hryvnia, predicted the estimated cost of each of the cost items, calculated net profit, payback period for the manufacturer 0,76 years and the economic effect for consumers using this development.

Keywords: blog-system, web-client, JWT, client-server architecture

ЗМІСТ

ВСТУП

Актуальність теми дослідження. Стрімкий розвиток технічного прогресу, глобальне поширення інформаційних технологій, діджиталізація впливають на зміни у всіх галузях суспільства. Сьогодні традиційні засоби масової інформації, такі як газети, журнали, радіо, поступово втрачають популярність серед аудиторії. Лише телебачення й інтернет залишаються найбільш затребуваними засобами масової інформації, чия популярність збільшується з кожним роком. Кожен користувач соціальних мереж має змогу створювати й транслювати свої матеріали. Завдяки цьому рівень швидкості подачі інформації значно прискорився, але якість інформації, на жаль, здебільшого залишається дуже низькою. Найпоширенішим жанром новітньої інтернет-журналістики можна назвати блог. Сьогодні у світі існує близько 1 млрд. блогів. Більш точну цифру не зможе назвати жоден фахівець. Адже щодня у всесвітній мережі інтернет виникає приблизно 100 000 нових блогів та 70000 видаляється з мережі. Блоги стали невід'ємною частиною спілкування людей, обміну досвідом та інформацією. Також існують блоги відомих тварин, міст або речей. Треба зазначити, що вони також є складовою частиною інтернет-видань. Тепер замість колонок фахівців можна зустріти блоги фахівців, що діляться своїм досвідом, думками з читачами засобу масової інформації.

Ріст числа блогів призвів до виникнення блог-платформ. Блог-платформа – це сервіс, що надає користувачеві готовий до роботи блоґовий рушій і дозволяє вести блоґ без необхідності самостійно займатися його обслуговуванням і програмуванням. Зворотна сторона такої зручності - неможливість повноцінного налаштування блоґу, за винятком шаблонів дизайну. Крім того, контент користувача з правової точки зору знаходиться під контролем власника платформи і належить користувачу лише номінально. Як наслідок, останній обмежений у свободі самовираження, що зазвичай прямо прописано в правилах надання сервісу і часто контролюється

«конфліктною командою» (англ. AT, Abuse Team) власника блог-платформи [1].

Зв'язок роботи з науковими програмами, планами, темами. Магістерська робота виконана відповідно до напрямку наукових досліджень кафедри комп'ютерних наук Вінницького національного технічного університету спеціальності 122 «Комп'ютерні науки» та плану наукової та навчально-методичної роботи кафедри.

Мета та завдання дослідження. Метою магістерської кваліфікаційної роботи є розширення функціональних можливостей блог-систем.

Для вирішення поставленої мети необхідно вирішити наступні **задачі**:

1. Провести аналіз сучасного стану розвитку організації блог-систем.
2. Обґрунтувати вибір інструментів технічної реалізації інформаційної технології організації блог-систем.
3. Виконати проектування інформаційної технології організації блог-систем.
4. Виконати програмну реалізацію інформаційної технології організації блог-систем.
5. Провести тестування інформаційної технології організації блог-систем та аналіз результатів тестування.

Об'єктом дослідження є процес організації блог-систем.

Предметом дослідження є програмні засоби організації блог-систем.

Методи дослідження - методи автентифікації, методи обробки тексту, методи масштабування та інтеграції програмного забезпечення, методи та підходи до розробки систем ведення блогу, методи та підходи до розробки веб-орієнтованих програмних додатків, методи об'єктно-орієнтованого програмування.

Наукова новизна одержаних результатів полягає в наступному: розроблено інформаційну технологію організації блог-систем, що відрізняється від існуючих гнучким підходом до розробки блог-систем на

основі композитної архітектури, що забезпечило розширення функціональних можливостей блог-систем.

Практичне значення одержаних результатів полягає у наступному:

1. Розроблено алгоритм роботи інформаційної технології організації блог-систем, що описує клієнт серверну обробку даних з використанням процесу автентифікації

2. Здійснено програмну реалізацію інформаційної технології організації блог-систем з підвищеною швидкодією клієнтської частини.

Достовірність теоретичних положень магістерської кваліфікаційної роботи підтверджується строгістю постановки задач, коректним застосуванням математичних методів під час доведення наукових положень, строгим виведенням аналітичних співвідношень, порівнянням результатів з відомими.

Особистий внесок магістранта. Усі результати, наведені у магістерській кваліфікаційній роботі, отримані самостійно.

Публікації. За результатами магістерської кваліфікаційної роботи отримане авторське свідоцтво про реєстрацію авторського права на твір [2].

Апробація. Основні результати досліджень апробовано на III Міжнародній науково-практичній конференції "SCIENCE AND TECHNOLOGY: PROBLEMS, PROSPECTS AND INNOVATIONS" (14-16.12.2022 року, м. Осака, Японія).

1 АНАЛІЗ СУЧАСНОГО СТАНУ РОЗВИТКУ ОРГАНІЗАЦІЇ БЛОГ-СИСТЕМ

1.1 Огляд проблем в області організації блог-систем

Надзвичайна популярність блогів зумовлена двома головними обставинами: по-перше, публікувати інформацію в Інтернеті за допомогою блогів досить легко — фактично, створення нового посту зводиться до набирання його тексту у відповідному полі та відправки його на сервер шляхом натисканням кнопки «Публікувати» (англ. Publish) або подібної. Після цього пост зберігається на сервері, який автоматично формує веб-сторінки, різні посилання, додає стиль форматування тощо (такі серверні програми називаються системами управління змістом — англ. Content Management Systems або CMS). Тому користуватися блогом не важче, ніж простим текстовим редактором. Друга причина — це моментальна доступність в Інтернеті опублікованої інформації; існують декілька безкоштовних блогових платформ (наприклад Wordpress або Livejournal), де будь-хто може зареєструватися та вести свій блог. Блоги періодично скануються Інтернет-пошуковиками, такими як Google, Yahoo!, та ін [3].

Саме з цих характеристик — мобільності та доступності блогів — випливає їх визначна роль у творенні інформаційного суспільства, реалізації свободи слова, боротьбі за права людини у світі, та й просто у комунікації та обговоренні будь-яких ідей поміж людьми. Визнанням великої соціальної ролі блогосфери у боротьбі за права та свободи людей стала нещодавня публікація так званого «Довідника блогерів та кібер-дисидентів» (Handbook for Bloggers and Cyber-dissidents) асоціацією «Репортери без кордонів» (Reporters without Borders), який є вільним для використання та поширення і служить прекрасним вступом як до основ ведення блогу і різних його аспектів, так і до застосування блогів як зброї для боротьби з тоталітарними урядами та

цензурою. Одним з багатьох прикладів соціальної ролі блогів можна вказати відомий Interdictor's блог на LiveJournal, який був присвячений урагану «Катріна» і вівся хлопцем, який залишився у Новому Орлеані і кожні кілька годин сповіщав про розгортання трагічних подій, що у той час сприяло підбуренню суспільної думки проти занадто повільної реакції Білого Дому [3].

Більшість блогів приватного характеру становлять інтерес лише для невеликої групи людей, які можуть бути знайомі з автором або цікавляться його думками та коментарями щодо подій у світі, фахової діяльності, родинного життя, мистецтва тощо. Тільки деякі з них стають відомими блогерами, нові пости яких можуть отримувати десятки або сотні коментарів [4].

Феномен ведення блогів сильно пов'язаний з еволюцією Інтернету, і деякі можуть подумати, що лише цифрові аборигени – так зване покоління С – здатні вести блоги. Він також створив зв'язки з маркетингом/брендингом та його кількома значними тенденціями, такими як сторітеллінг та автентичність. Очевидно, що ведення блогів сьогодні розглядається як професія, і блоги можуть мати різні ролі, які будуть обговорюватися. По-перше, компанії використовують власні блоги як частину своєї комунікаційної стратегії. По-друге, є приватні блогери, яких можна розглядати як загрозу або можливість для компаній і їх брендів. Компанії можуть вибрати, якого блогера найняти для просування свого товару або послуги. Нинішня блогосфера насичена, і компанії можуть вибирати з широкого кола блогерів. По-третє, ведення блогів поширюється на різні галузі, включаючи засоби масової інформації, де блогерів можуть сприймати як журналістів. Однак важливо знати про вміст, який вони створюють, і ділитися ним у формі дописів у блозі, оскільки вміст перевіряє потужність.

Якщо ми хочемо повернутися в минуле і з'ясувати, які витоки терміна «блог», важливо нагадати собі, де розміщуються блоги. Саме Інтернет дозволив виникнути іншому програмному забезпеченню та платформам. З 90-х років, коли Інтернет став доступним для громадськості, він пройшов кілька

етапів своєї еволюції. На першому етапі люди не могли насправді взаємодіяти в Інтернеті інакше, ніж читання. Ми називаємо цей етап Інтернету Веб 1.0. Тоді експертам знадобилося всього пару років, щоб розробити так званий Web 2.0, який дозволив людям читати, писати і публікувати власний контент. У той час, зокрема в 1997 році, Йорн Баргер представив перший особистий блог в історії. Він назвав свій блог "веб-журналом". Однак насправді не має значення, хто був першим блогером в історії, тому що блоги, швидше за все, будуть розвиватися пліч-о-пліч з технологічним прогресом [5].

Блоги стали швидко популярними, оскільки їх легко створювати та підтримувати. Сьогодні кожен користувач Інтернету може стати творцем контенту. Кожен може написати власний блог і поділитися інформацією. Існує кілька інструментів, таких як WordPress або Tumblr, де люди можуть створювати та налаштовувати свої блоги. Всі ці інструменти зручні для користувачів, і тому кількість блогів і блогерів постійно збільшується. За словами Hsu and Lin, існують також інші причини постійного збільшення кількості блогів, таких як насолода, обмін знаннями та ідентифікація спільноти як соціальний фактор. Здається, що блоги заполоняють Інтернет. Дані 2013 року показують, що у всьому світі існує понад 152 мільйони блогів і що люди щодня читають блоги. Блоги розглядаються як важливе джерело громадської думки, і, таким чином, очевидно, що блоги є одним з найважливіших засобів комунікації. Це пов'язано з тим, що ми, користувачі, значною мірою покладаємося на Інтернет, оскільки він є одним з основних джерел інформації. Просто ми краще поінформовані, ніж будь-коли раніше. Блоги надають інформацію на різні теми (засоби масової інформації, краса, їжа, сім'я, здоров'я тощо), але, згідно з Technorati, блоги можуть бути розділені на п'ять наступних категорій: блоги для розваги; професійні блоги за сумісництвом; професійні блоги на повний робочий день; корпоративні блоги; підприємницькі блоги [5].

Блоги зазвичай пишуться окремими особами або співробітниками компанії. Однак можна подумати, що блоги підходять лише для молодих

людей або навіть лише для цифрових аборигенів, до яких деякі посилаються як покоління С [5].

Покоління С представляє людей, які народилися після 90-х років. Однак вони повинні відрізнятися скоріше технологічним прогресом, ніж роком народження, тому що покоління С наділене цією технологією. Ось чому «С» означає зв'язаний, завжди клацає, спілкується, творчий або орієнтований на контент. Тим не менш, вони все ще належать до двох найбільших категорій користувачів Інтернету, яким від 15 до 25-34 років. За даними Wertime and Fenwick, ці користувачі беруть участь в Інтернеті, читаючи та записуючи вміст. Сьогодні молоді люди дуже залучені в Інтернеті, де вони можуть вільно висловлювати свою думку. Також прогнозується, що покоління С продовжить все більше споживати цифрову інформацію, щоб спростити і поліпшити своє життя. Зрозуміло, що покоління С є «двигуном» цифрової ери. Тим не менш, є також докази того, що люди старше 44 років також мають досвід створення інтернет-контенту. Це означає, що завдяки різним зручним інструментам в Інтернеті блогером може стати практично кожен [5].

Початкова роль блогів полягала в тому, щоб ділитися або обговорювати вміст Інтернету у формі коментарів або щоденника за допомогою тексту, інфографіки, картинки, відео або посилання. Тим не менш, поява Web 2.0 та його цифрових інновацій дозволили поширити блоги по всьому Інтернету. Це означає, що приватні особи були не єдиними, хто зрозумів, що блоги можуть бути потужним інструментом для обміну думкою. Раптом здається, що компанія, якої немає в Інтернеті, або та, яка не використовує соціальні мережі для спілкування зі своїми клієнтами, знаходиться на цьому взаємопов'язаному глобальному ринку майже «мертвою». Традиційні маркетингові інструменти майже застаріли, і кмітливі маркетологи адаптуються до поточної цифрової ери, зосереджуючись більше на онлайн-медіа разом із впровадженням SEM та SEO. Як стверджують Армеліні та Віллануева, 79% із 100 найкращих компаній, перелічених у списку Fortune 500, використовують соціальні мережі, включаючи блоги, щоб залишатися ближче до своїх клієнтів.

Очевидно, що маркетинг переживає революцію. Тому компаніям частіше включають блоги разом з іншими соціальними мережами. Сьогодні корпоративні блоги можуть чергувати традиційну рекламу та брендинг, оскільки маркетологи можуть збільшити знання про продукт, створити або зміцнити бренд, а компанії можуть демонструвати досвід через свої дописи в блозі. У цьому випадку компанії мають владу над своїми посадами, і вони можуть зміщувати інформацію в потрібному напрямку. Вони також можуть залучити величезну світову онлайн-спільноту [5].

Однак роль блогів може відрізнятися, і це відбувається особливо щодо приватних блогерів. Було проаналізовано, що сарафанне радіо поширилося в Інтернеті, і блогери можуть бути його прихильниками. Блогери описують і діляться інформацією про свій особистий досвід. Деякі з них можуть мати близько мільйона відвідувачів, і вони підтримують міцні зв'язки між собою. Тому онлайн-сарафанне радіо може бути дуже потужним, оскільки Інтернет дозволяє негайно поширювати інформацію по всьому світу. Приватні блогери є такими ж споживачами, як і все суспільство, і, очевидно, вони інформують своїх читачів, задоволені вони чи ні продуктом або послугою, яку вони споживають. Хто тоді має більше влади над брендом? Чи все-таки його власник, якщо на спілкування, яке відбувається в блогосфері, вже не можна вплинути? Незважаючи на те, що інформація в блогах може бути поверхневою, неперевіреною або суб'єктивною, деякі дописи в блозі можуть завдати шкоди продукту або вплинути на репутацію бренду. Крім того, 21 відсоток людей вважають блоги найбільш надійним джерелом інформації. Тому компанії повинні діяти негайно. У деяких країнах були створені хаби для блогерів (наприклад, Elite Bloggers в Чехії), де можуть працювати блогери, і які допомагають компаніям і блогерам співпрацювати разом. Зокрема, ці хаби допомагають компаніям знайти відповідних блогерів і створити індивідуальну маркетингову кампанію. Всі ці розробки ще більше сприяли веденню блогів до нової професії в сучасній рекламі і брендингу. Це також може призвести до безпрограшної ситуації. Компанії можуть в якійсь мірі втручатися в

комунікацію навколо свого продукту і в той же час блогери можуть отримати додаткові вигоди від компанії, на яку вони працюють. З іншого боку, читачам іноді стає важко розпізнати такі спонсорські дописи, але це скоріше пов'язано з іншим аспектом ведення блогів – етикою [5].

Співпраця між блогерами та компаніями стала звичайною практикою в різних галузях, таких як мода чи медіа. З точки зору ведення блогів, останній приклад може насправді досліджувати роль журналістики. Наприклад, одна з відомих газет The Economist прийняла десять блогів (наприклад, блокнот Баттонвуда, Erasmus, теорію ігор, Gulliver тощо) як свою додаткову послугу. Це може означати, що межа між веденням блогів (написанням онлайн-контенту) і журналістикою починає бути розмитою. Тим не менш, зміст блогів є дискусійним, оскільки він може відображати або обговорювати лише певну тему, і це не слід сприймати як належне. Таким чином, очевидно, що існують різні погляди на те, чи може ведення блогу дорівнювати журналістиці. З цієї причини корисно порівняти і побачити відмінності між цими двома професіями. Одна з найважливіших відмінностей включає сферу фокусування: журналісти повинні зосереджуватися на фактах, тоді як блогери - ні. Блогерам дозволено писати про різні типи контенту, у них більше свободи для експериментів і вони можуть бути більш особистими, щоб апелювати до емоцій. Серед інших суттєвих аспектів, які можуть провести межу між журналістом і блогером, є освіта. Журналісти зазвичай мають вищу освіту в галузі журналістики, тоді як блогери можуть отримати освіту в абсолютно іншій галузі або в будь-якій галузі [5].

Як правило, на успіх блогерів може впливати їхня здатність до особистого розповіді, оскільки це підвищує автентичність. Люди схильні читати історії, за допомогою яких вони можуть ідентифікувати. Це може привести до досягнення високої кількості відвідувачів і далі до створення власної спільноти. Тому блоги можуть бути потужним інструментом на основі інформації, яку він містить, а це означає, що влада здійснюється через вміст. Блогер, який здатний створити хороший контент і набрати значну кількість

відвідувачів, може бути тим, хто диктує тенденції. Таким чином, він може бути тим, хто представляє загрозу або можливість для компаній і брендів [5].

1.2 Аналіз відомих програмних рішень

Проблему створення блогу вирішують велика кількість компаній. Розглянемо деякі з існуючих рішень вирішення проблеми створення блогу.

Першим прикладом є система WordPress.com, найпопулярніша CMS для корпоративних блогів по всьому світу. На ній працюють 30% сайтів. Перевагами цієї системи є безкоштовна реєстрація і створення блогу, створення власного дизайну, можливість заробляти на рекламі. У головній перевазі системи криється і найвагоміший недолік. Монетизація можлива тільки при використанні тарифів Premium і Business. Також для створення більш гнучкого блогу необхідні базові знання програмування [6].

У зв'язку з тим, що WordPress є однією з найбільш широко використовуваних систем управління контентом, було б дивно не включати його в цей список. WP вважається стандартом для веб-сайту, який має необмежену функціональність і з яким приємно працювати. Хоча WordPress вимагає помірної кривої навчання, він все ще надзвичайно гнучкий по відношенню до новачків, а також до професіоналів. Користувачі WP можуть розширити функціональність своїх веб-сайтів і зробити їх унікальними за допомогою тисяч тем і плагінів. Щоб почати блог на WordPress, вам потрібно буде зареєструвати доменне ім'я та вибрати веб-хостинг. Після цього у вашому розпорядженні буде сайт для його налаштування, додавання контенту і т.д. [7].

Головну сторінку WordPress.org наведена на рисунку 1.1.

Рисунок 1.1 – Загальний вигляд головної сторінки WordPress.org

WordPress.com - це розміщена версія WordPress, що означає, що все, що вам потрібно зробити, щоб почати вести блог, - це створити обліковий запис. Тут ви можете публікувати свої пости відразу. Ведення блогу на цій платформі є безкоштовним, але якщо є необхідність у розширених функціях та опціях, ви можете вибрати один із платних планів. Це CMS, яку можна налаштувати, яка робить речі простими та зрозумілими. Є багато тем на вибір, а також плагіни. WordPress.com — найкраще рішення для тих, хто піклується лише про ведення блогів і хоче поєднання потужності, налаштування та зручності використання. WP є найпопулярнішою платформою для ведення блогів для розробників – існує понад 100 плагінів і рішень для індивідуальних переваг блогів [7].

Головну сторінку WordPress.com наведено на рисунку 1.2.

Рисунок 1.2 – Загальний вигляд головної сторінки WordPress.com

Основні переваги та недоліки використання WordPress.com занесено в таблицю 1.1.

Таблиця 1.1 – Переваги та недоліки WordPress.com

Переваги WordPress.com	Недоліки WordPress.com
Відсутність потреби у налаштуванні	Обмежені можливості розширення сайту. Відсутність можливості використовувати власні теми та плагіни для налаштування блогу.
Простота у використанні і управлінні	Відсутність можливості запускати рекламу у своєму блозі. Натомість WordPress.com відображає свою рекламу на безкоштовних веб-сайтах.

Продовження табл. 1.1

Безкоштовність за умови використання піддомену WordPress.com	Відсутність повного контролю над блогом. WordPress.com може призупинити дію облікового запису, якщо виявить порушення їх умов використання.
Доступ до більш ніж 55 000 безкоштовних плагінів	
різні чудові функції, які сподобаються кожному блогеру:інтернет-магазин, платне членство, форум	
SEO-плагінита інструменти, які дуже допомагають	
Велика спільнота, яка надає велику підтримку	

Іншим прикладом є Medium, що позиціонує себе як платформу для творчості і самореалізації. Її переваги – це безкоштовна реєстрація і ведення аккаунта, відсутність потреби знання дизайну і коду. ЇЇ недоліками є слабка монетизація та необхідність знання англійської мови для участі в партнерській програмі [8].

Програмна платформа забезпечує повний WYSIWYG користувацький інтерфейс для писання онлайн на основі одного шаблону із великою кількістю вільного простору та простими можливостями візуально привабливого форматування тексту, виділяння цитат та розміщення фото і відео матеріалів. Після того як матеріал опублікований його можуть рекомендувати чи поширити інші особи за зразком Twitter'a. Дописи можна оцінювати так само як на Reddit, а контент може бути долученим до конкретної теми, так само як у Tumblr. На відміну від Blogger публікації сортується за темами, а не

авторами. У платформі використовується система рекомендацій, яка нагадує «подобається» у Facebook і являє собою систему тегів. Вони розподіляють матеріали на різні категорії, що дає можливість користувачам легше орієнтуватись та знаходити цікаві матеріали. У Medium, система кількості «уподобань», характерна для соціальних мереж, замінена на систему оцінки кількості часу потраченого на читання матеріалу користувачами. Публікації зберігаються на хості, як газета чи журнал. Закладена можливість працювати над матеріалами спільно з іншими авторами, та робити особисті помітки на полях, а також зберігати чернетки. Користувач також може визначати основне зображення, яке буде виникати при перепублікації матеріалу в соціальних мережах, задавати ключові слова, або скорочувати великі вебадреси публікацій що написані не латинським шрифтом. Можна також визначати час коли матеріали мають бути оприлюднені. Medium дає підказки початківцям, які допомагають швидше освоїти процес створення публікації [7].

Головну сторінку Medium наведено на рисунку 1.3.

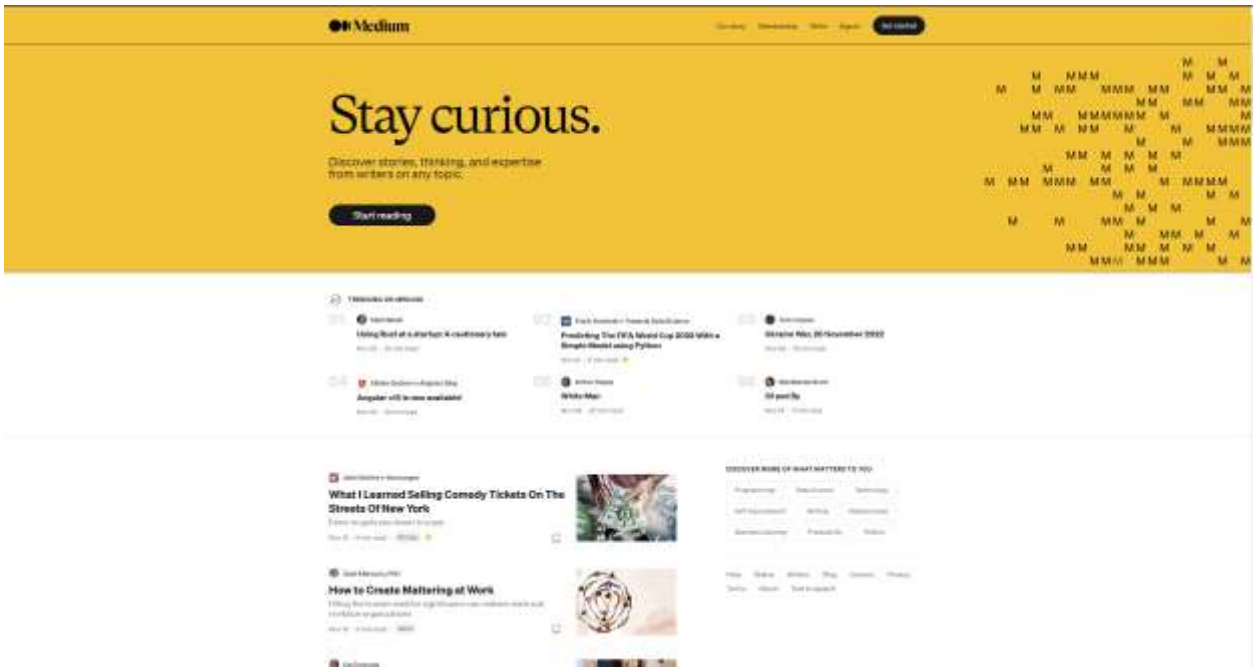


Рисунок 1.3 – Загальний вигляд головної сторінки Medium

Основні переваги та недоліки використання Medium занесено в таблицю 1.2.

Таблиця 1.2 – Переваги та недоліки Medium

Переваги Medium	Недоліки Medium
Простота у використанні, без необхідності установки і без навичок кодування.	Обмеженість з точки зору дизайну або побудови бренду.
Дозволяє охопити існуючу онлайн-спільноту людей, що мають схожі інтереси	Medium володіє вашою аудиторією, тому втрата вашого блогу означає втрату всіх ваших послідовників.
Можливість зосередитись на створенні контенту, а не на розробці веб-сайту.	Відсутність можливості використовувати власне ім'я домену. Натомість ви просто отримаєте сторінку профілю, як у Facebook, наприклад, https://medium.com/@yourname.
	Відсутність можливості запускати власні рекламні оголошення, щоб заробити гроші.

Не менш цікавим прикладом є платформа Tumblr. Tumblr трохи відрізняється від інших платформ для ведення блогу. Це платформа для мікроблогів із функціями соціальних мереж, включаючи перегляд інших блогів, реблогінг, вбудовані інструменти спільного використання тощо[9].

«Tumblr» неофіційно відомий своєю концепцією «мікроблогінгу». Це чудова безкоштовна платформа для ведення блогів, розроблена для тих, хто вважає за краще публікувати короткий вміст (переважно візуальний: зображення, відео, інфографіку, цитати, анімовані GIF-файли тощо). Ця платформа для ведення блогів поставляється з вбудованою опцією спільного доступу, що означає, що немає необхідності ділитися вмістом вручну. Tumblr — ідеальне місце для блогерів, які хочуть одразу почати публікувати свої

дописи. Це також чудове місце для тих, хто не хоче перетворювати свій блог на бізнес-проект, а також не проти обмежених можливостей налаштування. Але як тільки ви вирішитеперемістити свій блог з Tumblr на WordPress або іншу платформу, ви можете зробити це автоматично за допомогою CMS2CMS [7].

Головну сторінку Tumblr наведено на рисунку 1.4.

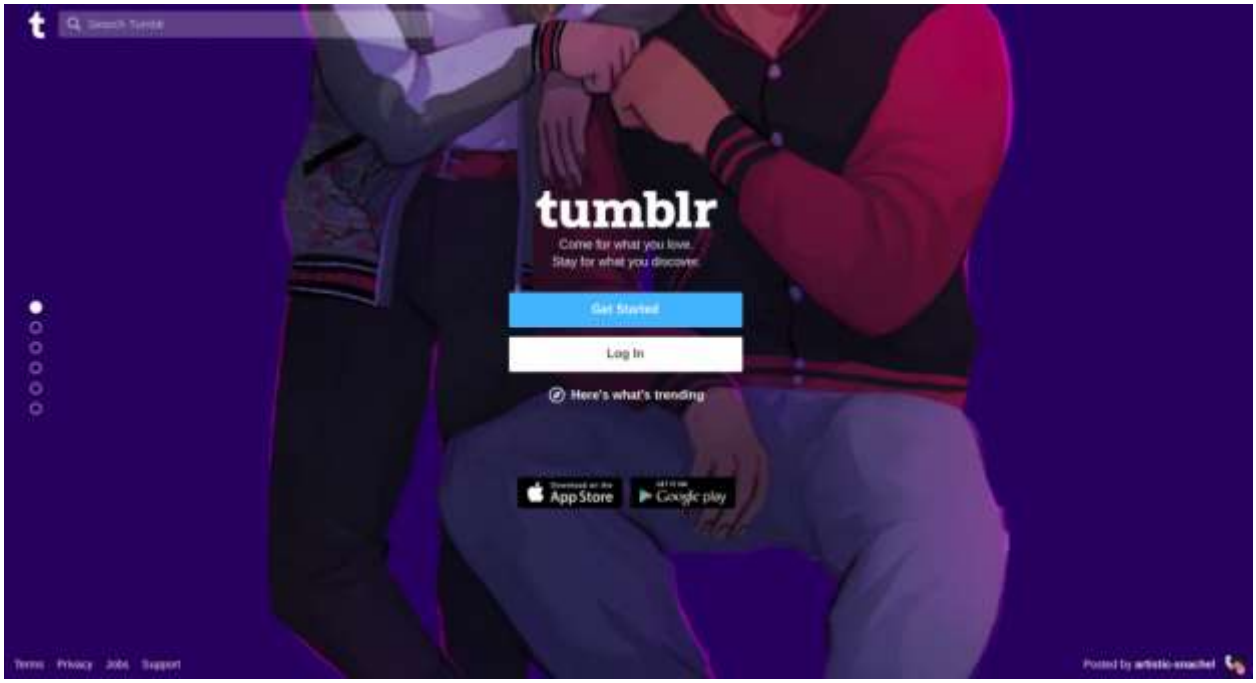


Рисунок 1.4 – Загальний вигляд головної сторінки Tumblr

Основні переваги та недоліки використання Tumblr занесено в таблицю 1.3.

Таблиця 1.3 – Переваги та недоліки Tumblr

Переваги Tumblr	Недоліки Tumblr
Безкоштовність за умови використання піддомену Tumblr. Можливість платно підключити власний домен.	Tumblr поставляється з обмеженим набором функцій, які ви не можете розширити в міру зростання вашого блогу.

Продовження табл. 1.3

Простота у використанні і налаштуванні	Для Tumblr доступно багато тем, але вони не можуть запропонувати додаткові функції.
Наявність інтегрованого компоненту соціальних мереж.	Досить важко створити резервну копіювання блогу Tumblr або імпортувати його на інші платформи.
Як інструмент мікроблогів, Tumblr дозволяє легко швидко викладати відео, GIF-файли, зображення та аудіо формати.	

Можна побачити, що існуючі рішення є вузькоспеціалізованими та не покривають більшість випадків необхідності монетизації. Тому створення компромісної системи може вирішити вищенаведені проблеми існуючих програмних рішень.

1.3 Постановка задачі і формулювання вимог до інформаційної технології організації блог-систем

Оскільки більшість блогів є веб-орієнтованими та їх бази даних оновлюються в реальному часі, доцільно буде розробити інформаційну технологію у вигляді веб додатку. Істотною перевагою побудови веб-застосунків для підтримки стандартних функцій браузера є те, що функції повинні виконуватися незалежно від операційної системи клієнта. Замість того, щоб писати різні версії для Microsoft Windows, Mac OS X, GNU/Linux й інших операційних систем, застосунок створюється один раз для довільно обраної платформи і на ній розгортається.

Вхідними даними для інформаційної технології блог-систем є: користувачі, їх пости та коментарі, кількість символів у пості не більша за 750,

кількість символів у коментарі не більша за 250, кількість користувачів не менш ніж 100.

Вихідними даними є користувацький блог, профілі користувачів, пости та коментарі до них. Так як технологія матиме форму веб-додатка, він буде кросплатформеним і не залежатиме від конкретного апаратного забезпечення.

Отже створювана інформаційна технологія має володіти рядом функціональних можливостей:

- можливість реєстрації та аутентифікації;
- можливість редагування профілю;
- можливість створення, редагування та видалення постів;
- можливість створення, редагування та видалення коментарів.

1.4 Висновок до розділу 1

Під час огляду проблеми організації блог-систем було розглянуто сучасні підходи до розробки додатків та визначено, що найбільш оптимальною формою такого додатку є веб-застосунок.

Під час постановки задачі було визначено основні вимоги до інформаційної технології організації блог-систем. Проведено аналіз програм-аналогів: WordPress.com, Medium, Tumblr, визначено їхні характеристики, переваги та недоліки. Зокрема, проблемами даної сфери є слабка монетизація та необхідність знання програмування, відсутність універсальних рішень, що покриватимуть більшість вимог користувачів. З цього можна сформувати основну задачу даної роботи – розробку компромісного рішення, що дозволить користувачу повністю отримувати кошти, зароблені з допомогою монетизації, та обрати гнучке рішення для себе, без потреби у програмуванні.

Отже в даному розділі було визначено основні проблеми які виникають при організації блог-систем, та вимоги до інформаційної технології організації блог-систем.

2 РОЗРОБКА ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ОРГАНІЗАЦІЇ БЛОГ-СИСТЕМ

2.1 Вибір підходу до організації блог-систем

У залежності від рівня сервісу блог-платформи можна умовно розділити на три групи [10]:

1. Професійні: користувачеві надається індивідуальний рушій блогу, включаючи необхідні плагіни, налаштований відповідно запитам користувача. Доступу до коду рушія користувач, як правило, не має. Крім того, надається хостинг файлів і обмежена можливість запуску своїх скриптів (або їх підключення з готового переліку).
2. Напівпрофесійні: користувачеві надається можливість оренди рушія (нерідко можливий вибір одного з декількох рушіїв). Можливостей індивідуального налаштування немає. Для зберігання файлів також надається хостинг.
3. Масові: користувачеві надається обліковий запис і оренда ресурсів сервера. Прямого доступу до даних у користувача немає, тільки з використанням штатних засобів рушія.

Професійні і напівпрофесійні блог-платформи зазвичай платні, оскільки використовують модель надання хостингу, адаптованого для ведення блогу, плюс оренду програми та її обслуговування. Масові блог-платформи рідко бувають платними, оскільки надають не хостинг, а масовий веб-сервіс. Через це в професійних блог-платформах соціальних зв'язків між користувачами менше, але вони більш таргетовані (цільові). На масових блог-платформах соціальна зв'язність виходить на перший план, підпорядковуючи професійну [10].

Для розробки інформаційної технології організації блог-систем було обрано використання професійного підходу.

2.2 Обґрунтування вибору платформи для розробки інформаційної технології організації блог-систем

Зазвичай платформа відноситься до операційної системи та комп'ютерного обладнання. Платформа є набором стандартів, які дозволяють розробникам розробляти програмні додатки на основі правильного стеку технологій. Ці стандарти дозволяють власникам і менеджерам придбати відповідні програми та обладнання.

Веб-застосунок — це програма, яку можна переглядати в Інтернеті. На перший погляд вони виглядають так само, як і звичайні сайти, але є величезна різниця. Два типи веб-додатків в тренді. Це SPA (односторінкова програма) та PWA (прогресивний веб-додаток) [11].

На відміну від веб-сайтів, додаток для однієї сторінки не з'єднується з сервером після кожної взаємодії — код завантажується лише один раз, а взаємодія з додатком фактично виконується на сервері.

PWA - це веб-сайт, який поводиться як мобільний додаток. Він дозволяє розміщувати іконку на головному екрані, надсилати push-сповіщення тощо. Крім того, він функціонує в автономному режимі. Його легко знайти, він не перевантажує ваш смартфон сховищем — взагалі, є різні переваги, які ви можете отримати від впровадження таких технологій [11].

Для розробки веб-сайту можуть бути використані готові шаблонні рішення з коробки (CMS) або користувальницькі рішення.

CMS, а саме налаштування готового шаблону, - це швидкий спосіб розробки вашого продукту, так як бібліотека, що використовується розробником, вже містить великий список стандартних елементів.

Основні переваги та недоліки використання CMS занесено в таблицю 2.1.

Таблиця 2.1 – Переваги та недоліки CMS

Переваги CMS	Недоліки CMS
швидка розробка сайтів;	CMS не підходить для складних продуктів. Якщо ви плануєте розробити систему SRM, краще подати заявку на індивідуальну розробку, оскільки налаштування існуючих рішень в CMS може зайняти набагато більше часу, ніж писати оригінальний код;
дозволяє легко адмініструвати і додавати нові елементи через панель адміністратора;	сайти на CMS не гарантують безпеку;
можливість створювати красиву конструкцію без особливих складнощів;	низький рівень оптимізації для пошукових систем (потрібно уважно вивчити особливості кожної CMS);
простий контент, доступний навіть людям без спеціальних знань.	обмежені можливості в налаштуванні інтерфейсу, тому сайти виглядають досить схоже;
	відсутність можливостей для розширення продукту. Необхідно заздалегідь дізнатися поріг допустимого навантаження на кожну з можливих CMS.

Розробка користувальницьких веб-сайтів - це перспективна можливість висловити свою індивідуальність. Цікавий і оригінальний виріб завжди привертає більше уваги, ніж подібний, але типовий аналог.

На відміну від CMS, користувальницький веб-сайт надає майже нескінченні можливості

Основні переваги та недоліки розробки користувальницького веб-сайту занесено в таблицю 2.2.

Таблиця 2.2 – Переваги та недоліки розробки користувальницького веб-сайту

Переваги	Недоліки
високий рівень безпеки даних	час розробки довший у порівнянні з CMS, а тому для реалізації проекту потрібно закласти більший бюджет.
можливість розробки унікального інтерфейсу	
здатність до належного SEO;	
сайт легко масштабувати і додати нову функціональність	

На основі наведених вище переваг і недоліків було обрано розробку користувальницького веб-сайту. В перспективі система додатково буде розроблена CMS-система, яка буде пришвидшувати процес розробки веб-сайтів шляхом створення початкових шаблонів.

2.3 Обґрунтування вибору композитної архітектури інформаційної технології організації блог-систем

Композитна архітектура - це підхід "зроби сам" до побудови надійної архітектури з використанням найкращих у своєму роді API. Композитні архітектури забезпечують покращений досвід користувача, без усієї роботи та обслуговування саморобної системи або монолітного рішення.

Завдяки складній архітектурі команди можуть вибрати API, які найкраще відповідають їхньому випадку використання, розміру команди та бюджету, і з'єднати їх разом, щоб мати всю необхідну функціональність, не сплачуючи за функції, яких вони не мають. Оскільки ці потреби змінюються,

легко підключати або відключати послуги, звільняючи команди від дорогого блокування постачальників у минулому.

Команди отримують вигоду від постійного надання функцій та технічного обслуговування, спільних для продуктів SaaS, і більше не доведеться турбуватися про швидкі виправлення розробки homebrew або підтримки критично важливого програмного забезпечення з часом. Натомість вони можуть бути гнучкими та створювати продукти, які вони хочуть створювати, за допомогою надійних, безпроблемних послуг. Щоб створити ефективну складену архітектуру, команди повинні використовувати основний центр вмісту, який керує потоком даних між системами, а потім підключати інші найкращі в своєму роді служби, які відповідають вашому випадку використання.

Монолітні системи, на відміну від композитних архітектур, є потужними інструментами; однак вони дорогі, жорсткі, і часто команди платять за функціональність, яка їм не потрібна або не потрібна. Коли команди створюють універсальний інструмент, який може зробити все, вони часто є відображенням цифрових продуктів минулого, а не майбутнього [32].

Завдяки композитній архітектурі команди мають можливість вибирати спеціалізовані інструменти, які враховують найкращі практики для свого конкретного випадку використання. Команди можуть будувати потрібні робочі процеси саме з тією функціональністю, яка їм потрібна, не сплачуючи за більше. Після того, як основні сервіси будуть обрані і початкова настройка буде завершена, командам не доведеться турбуватися про підтримку сервісів. Команда може приступити до роботи над створенням сучасного цифрового продукту, який вони уявляють, використовуючи інструменти, призначені для цієї мети. Монолітні системи часто розглядають світ з веб-поглядом. Сервіси, що працюють на API, поділяють сучасне розуміння цифрових продуктів, що мають фронтенди, починаючи від мобільних додатків і закінчуючи смарт-годинниками та розумними помічниками [12].

Композитні архітектури дозволяють цифровим продуктам бути гнучкими та дозволяють командам працювати гнучко. Працюючи з інструментами, які були створені, щоб бути основою інноваційних рішень проблем, команди можуть мати надійні компоненти, які відповідають потребам команди. Однією з найбільших переваг композитних архітектур є те, що вони припускають, що дані можуть бути актуальними в декількох різних рівнях презентації [12].

Цифрові можливості більше не є синонімом веб-браузерів. Наприклад, компанія електронної комерції може мати веб-сайт, мобільний сайт, мобільний додаток, додаток для смарт-годинника та навички особистого помічника. Уникнення обмежень вмісту та забезпечення вільного потоку інформації має вирішальне значення для створення безперебійного та відповідає очікуванням споживачів. Композитні архітектури дозволяють усім цим фронтендам витягувати вміст з одного центру вмісту, що працює на основі федерації вмісту. При композитних архітектурах існуючі монолітні системи не відразу стають марними. Натомість цінні дані, що зберігаються в цих системах, можуть бути передані в нову композитну архітектуру, щоб збагатити центр вмісту та зменшити кількість часу, необхідного для міграції всіх даних у нові системи. З розділенням вмісту подача даних в CMS стає миттєвою [12].

Є різні підходи по подубови композитних архітектур на стороні клієнту, такі як: односторінкові програми, статичні генератори сайтів (SSG) і програмами для візуалізації на стороні сервера (SSR). Вони складають основу сучасного веб-досвіду. Кожен підхід має випадки використання, коли вони ідеальні і де тип вмісту виграє від фронтенд-шаблону. Динамічний вміст, який вимагає високого рівня персоналізації, ймовірно, краще підходить для підходу SSR, тоді як статичний вміст, оптимізований для SEO, може краще підходити для сайту SSG.

Односторінковий додаток (SPA) - це широкий загальний термін для програм, які надаються, коли клієнт їх запитує. SPA структуровані як єдина

HTML-сторінка, яка не має попередньо завантаженого вмісту. Вміст завантажується через файли Javascript для всієї програми та розміщується в межах однієї HTML-сторінки. У файлах Javascript зберігаються всі дані, що стосуються логіки програми, інтерфейсу користувача та зв'язку з сервером. Популярні фреймворки та бібліотеки Javascript для створення SPA включають всіх звичайних підозрюваних у React, AngularJS, Vue.js, Ember.JS та Svelte, серед інших [13].

Коли користувачі переміщаються по різних частинах SPA, між різними елементами програми не буде ніякого додаткового часу завантаження. SSG також можуть потрапити в цю категорію після того, як вони були завантажені в браузер. Оскільки все завантажено на стороні клієнта, команди повинні враховувати широке коло клієнтів, забезпечуючи при цьому швидку, безперебійну роботу користувачів. Завдяки сучасним фреймворкам розділення коду дозволяє завантажувати деякі елементи на вимогу, що може допомогти усунути цю проблему [13].

Основні переваги та недоліки використання SPA занесено в таблицю 2.3.

Таблиця 2.3 – Переваги та недоліки SPA

Переваги	Недоліки
Хоча початкове завантаження може бути довшим, як тільки програма повністю завантажиться, додаткове завантаження не буде потрібно.	Оскільки програма зростає в розмірах і складності, це може серйозно вплинути на початковий час завантаження, що може призвести до погіршення користувацького досвіду
Хороший вибір для динамічного досвіду, коли командам потрібне індивідуальне відчуття для свого користувацького досвіду	Підтримка хорошого SEO майже неможлива через час завантаження та відсутність початкового вмісту на HTML

Продовження табл. 1.1

Команди мають великий контроль над своєю архітектурою і можуть використовувати сучасні веб-фреймворки	Великі файли для складних веб-додатків можуть стати складними в обслуговуванні та організації
Може використовуватися в тандемі з іншими технологіями	Проблеми зі SPA-підходом вимагають обхідних шляхів, які можуть бути дорогими та трудомісткими

У той час як SPA завантажують всі свої дані на єдиний HTML-сайт, який надається тільки після запиту клієнта, генератори статичних сайтів використовують зовсім інший підхід до вмісту і до створення сторінок в цілому. Генератори статичних сайтів генерують вміст під час створення нових сторінок або під час внесення змін до вмісту. Оскільки SSG створюють статичні сайти, немає необхідності завантажувати сторінки на основі запитів користувачів. Вміст залишатиметься незмінним незалежно від користувачів. Використання SSG як частини технологічного стека дозволяє командам отримувати дані з декількох джерел даних і дозволяє командам скористатися сучасними підходами до веб-розробки. Варіанти використання, які ідеально підходять для підходу SSG, - це ті, де вміст не потрібно сильно персоналізувати [13].

Генератори статичних сайтів зазвичай використовуються спільно з CMS без голови, статичним хостинговим сайтом і CDN для кешування всіх даних. Webhooks спрацьовує до SSG, що відбулися зміни у вмісті, і зміни розгортаються на сайті, який зберігається в кеші. CDN дозволяють командам зберігати попередньо візуалізовані HTML-файли в місцях, які географічно ближче до запиту, ще більше скорочуючи час завантаження сторінки [13].

Основні переваги та недоліки використання SSG занесено в таблицю 2.4.

Таблиця 2.4 – Переваги та недоліки SSG

Переваги	Недоліки
Легко створювати відокремлену архітектуру з кількома джерелами вмісту	Для персоналізації та динамічного вмісту потрібні обхідні шляхи або додаткові служби
Швидкий час завантаження сторінки через те, що більша частина вмісту попередньо візуалізується, і статичний характер вмісту	Коли вміст змінюється, ви повинні перебудувати сайт, щоб ці зміни відображалися на сайті
Краще для SEO	

Рендеринг на стороні сервера дозволяє командам надавати динамічний вміст, який можна персоналізувати. Вони ідеально підходять для персоналізованого досвіду, де можна переглядати живі зміни даних. За допомогою SSR клієнти отримують повністю візуалізовану сторінку на вимогу, замість того, щоб чекати кілька секунд, поки завантажаться певні елементи. Рендеринг відбувається на сервері перед передачею їх в браузер. Коли вміст запитується на клієнті, дані отримуються з бази даних або CMS під час навігації користувача по сторінці. Завантаження всього на вимогу робить це повільніше, але гарантує, що вміст оновлюється і що будь-які зміни доступні в прямому ефірі. Деякі елементи можуть бути кешовані, такі як активи та файли CSS, і навіть деякі сторінки, візуалізовані сервером, але зазвичай дані витягуються безпосередньо з бази даних за запитом [13].

Візуалізовані сайти на стороні сервера (або візуалізовані програми на стороні сервера (SRAs)) є відмінним вибором для вмісту, який залежить від часу, і додатків, які покладаються на велику кількість взаємодії з користувачем. З SSR персоналізація набагато простіше і може бути хорошим варіантом для електронної комерції. З SSR важливо переконатися, що ваша

інфраструктура може обробляти запити до серверів і що сервери можуть легко масштабуватися, оскільки трафік продовжує зростати [13].

Основні переваги та недоліки використання SSR занесено в таблицю 2.5.

Таблиця 2.5 – Переваги та недоліки SSR

Переваги	Недоліки
SSR дозволяє командам створювати динамічний, персоналізований контент без трудомістких обхідних шляхів	SSR зазвичай вимагають більше викликів API на сервер
Зміни вмісту відображаються миттєво, на відміну від SSG, де команди повинні перебудувати сайт, щоб побачити зміни у вмісті	SSR за замовчуванням часто повільніші, ніж SPA та SSG
Сайти SSR є клієнтськими агностичними, відмінними від SPA, де клієнти можуть визначити час завантаження або якість сторінки	

Оптимальний підхід залежить від випадку використання та контенту на додаток до аудиторії, команди розробників, бюджету тощо. SPA з рендерингом на стороні клієнта можуть бути більш ефективними для створення динамічного веб-досвіду; однак команди зіткнуться з проблемами часу завантаження сторінки і можуть боротися з SEO. SSG дозволяють командам створювати статичні сайти, які швидко завантажуються та добре працюють із SEO, але можуть обмежити кількість персоналізації та динамічного вмісту, доступного без трудомістких обхідних шляхів [13].

Є деякі інструменти, які можуть допомогти об'єднати сучасні фреймворки з більшою гнучкістю. Next.js дозволяє створювати статичні сайти та використовувати рендеринг на стороні сервера, використовуючи їх гібридний підхід. Це означає, що команди можуть скористатися перевагами SSR або SSG залежно від ідеального варіанту використання елементів свого проекту. Робота з Next.js може бути корисною для забезпечення того, щоб проекти підтримували хороше SEO [13].

Для інформаційної технології організації блог-систем було обрано SPA підхід так як він дає користувачу кращу динаміку у використанні порівняно з іншими підходами.

2.4 Обґрунтування вибору архітектури проектування серверної частини інформаційної технології організації блог-систем

Існує два підходи до побудови сучасних серверних застосунків: монолітні застосунки і застосунки на базі мікросервісів. Монолітний додаток будується як єдиний уніфікований блок, тоді як архітектура мікросервісів - це сукупність менших, незалежно розгорнутих служб.

Монолітна архітектура - це традиційна модель програмної програми, яка будується як єдине ціле, автономне і незалежне від інших додатків. Слово «моноліт» часто приписують чомусь великому і льодовиковому, що не далеко від істини монолітної архітектури для проектування програмного забезпечення. Монолітна архітектура - це єдина велика обчислювальна мережа з однією кодовою базою, яка об'єднує всі бізнес-проблеми разом. Щоб внести зміни до такого роду додатків, потрібно оновити весь стек за допомогою доступу до кодової бази та створити та розгорнути оновлену версію інтерфейсу на стороні служби. Це робить оновлення обмежувальними та трудомісткими [14].

Моноліти можуть бути зручними на початку життя проекту для простоти управління кодом, когнітивних накладних витрат та розгортання. Це дозволяє випустити відразу все в моноліті. Структура монолітної архітектури наведена на рис. 2.1.

Основні переваги та недоліки використання монолітної архітектури занесено в таблицю 2.6.

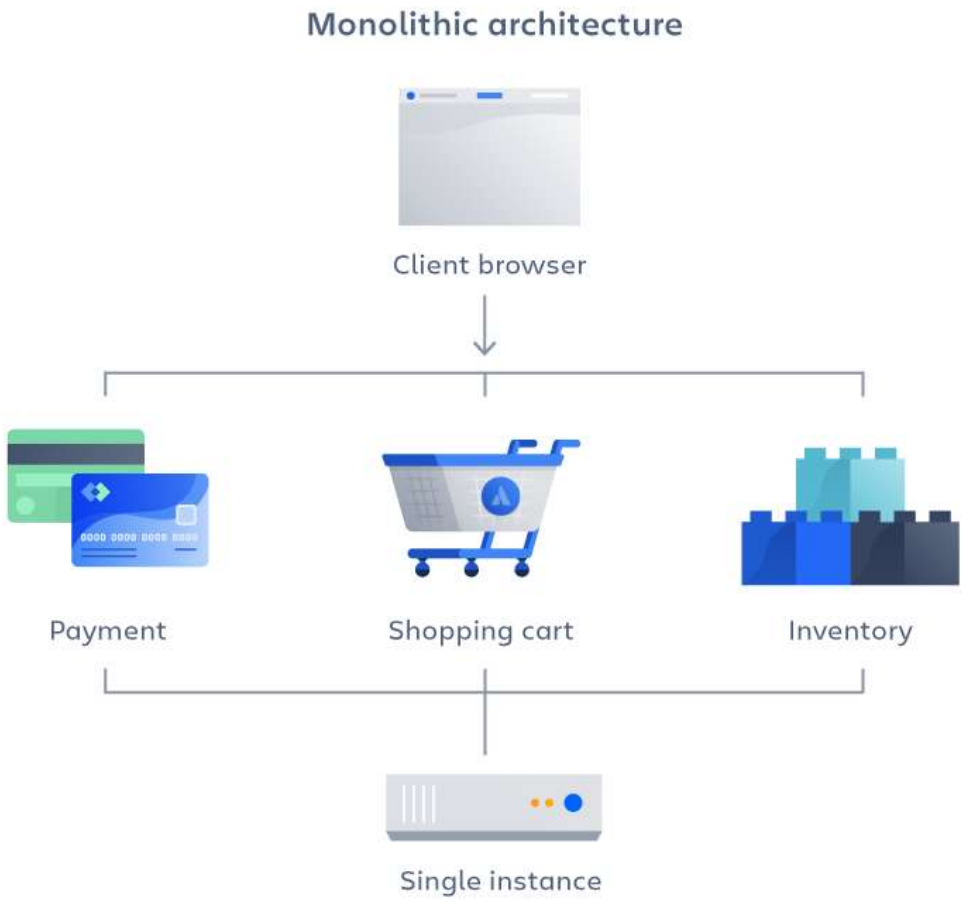


Рисунок 2.1 – Загальна структура монолітної архітектури

Таблиця 2.6 – Переваги та недоліки монолітної архітектури

Переваги монолітної архітектури	Недоліки монолітної архітектури
Просте розгортання – Один виконуваний файл або каталог полегшує розгортання.	Менша швидкість розробки - Велике монолітне застосування робить розробку більш складною та повільною.

Продовження табл. 2.6

Розробка – Коли програма побудована з однією кодовою базою, її легше розробити.	Масштабованість – Ви не можете масштабувати окремі компоненти.
Продуктивність – У централізованій базі коду та сховищі один API часто може виконувати ту саму функцію, яку численні API виконують з мікросервісами.	Надійність – Якщо в будь-якому модулі є помилка, це може вплинути на доступність усієї програми.
Спрощене тестування - Оскільки монолітний додаток є єдиним, централізованим блоком, наскрізне тестування може бути виконано швидше, ніж з розподіленим додатком.	Бар'єр для впровадження технологій - Будь-які зміни в фреймворку чи мові впливають на всю програму, роблячи зміни часто дорогими та трудомісткими.
Легке налагодження – З усім кодом, розташованим в одному місці, легше виконати запит і знайти проблему.	Відсутність гнучкості - Моноліт стримується технологіями, які вже використовуються в моноліті.
	Розгортання - Невелика зміна монолітного додатка вимагає передислокації всього моноліту.

Архітектура мікросервісів, також відома як мікросервіси, - це архітектурний метод, який спирається на серію незалежних служб, що розгортаються. Ці сервіси мають власну бізнес-логіку і базу даних з певною метою. Оновлення, тестування, розгортання та масштабування відбуваються в кожній службі. Мікросервіси об'єднують великі бізнес-проблеми, специфічні для домену, в окремі незалежні кодові бази. Мікросервіси не зменшують

складність, але вони роблять будь-яку складність видимою і більш керованою, розділяючи завдання на більш дрібні процеси, які функціонують незалежно один від одного і сприяють загальному цілому [14].

Впровадження мікросервісів часто йде пліч-о-пліч з DevOps, оскільки вони є основою для безперервної практики доставки, яка дозволяє командам швидко адаптуватися до вимог користувачів. Структура монолітної архітектури наведена на рис. 2.2.

Основні переваги та недоліки використання монолітної архітектури занесено в таблицю 2.7.

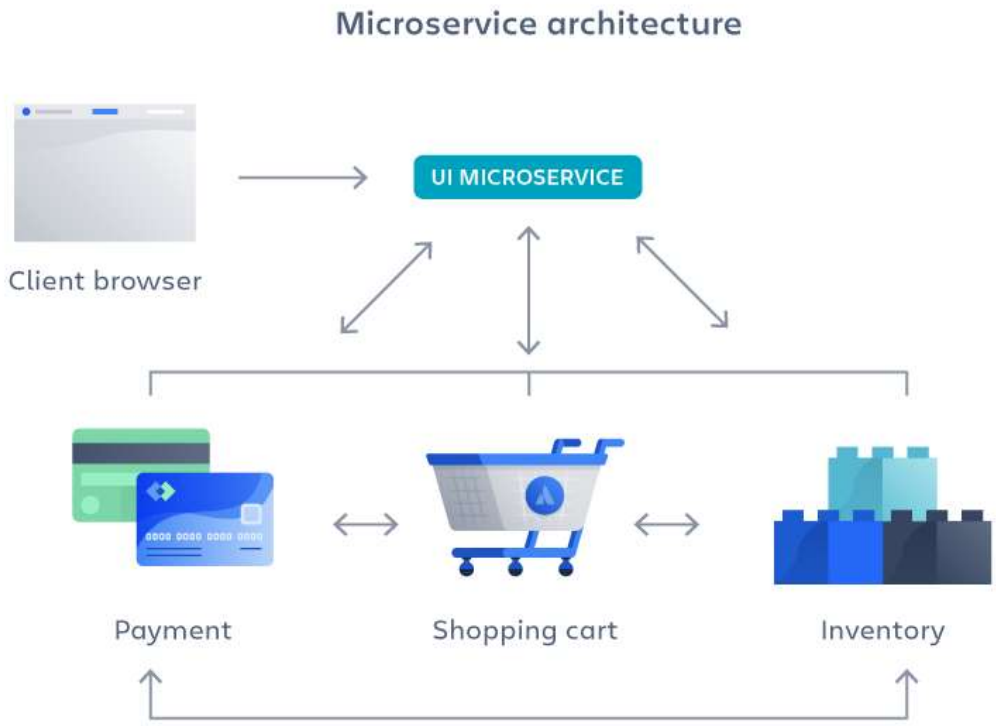


Рисунок 2.2 – Загальна структура мікросервісної архітектури

Таблиця 2.7 – Переваги та недоліки монолітної архітектури

Переваги	Недоліки
Спритність – Пропагуйте гнучкі способи роботи з невеликими командами, які часто розгортають додатки.	Розподілення розробки – Мікросервіси додають більшої складності в порівнянні з монолітною архітектурою, оскільки в більшій кількості місць, створених кількома командами, більше сервісів. Якщо розподілення розробки не керується належним чином, це призводить до уповільнення швидкості розвитку та низької експлуатаційної продуктивності.
Гнучке масштабування – Якщо мікросервіс досягає своєї вантажопідйомності, нові екземпляри цього сервісу можуть бути швидко розгорнуті в супровідному кластері, щоб допомогти знизити навантаження. Завдяки цьому можна одночасно працювати з багатьма клієнтами розподіленими на різні сервери та підтримувати набагато більші розміри серверів.	Експоненціальні витрати на інфраструктуру – Кожен новий мікросервіс може мати свою вартість для набору тестів, планів розгортання, інфраструктури хостингу, інструментів моніторингу тощо.
Безперервне розгортання – Використання мікросервісів дозволяє пришвидшити цикл випуску.	Додаткові організаційні накладні витрати – Командам потрібно додати ще один рівень комунікації та співпраці для координації оновлень та інтерфейсів.

Продовження табл. 2.7

Легкість тестувати та підтримки – Команди можуть експериментувати з новими функціями та відкочуватися, якщо щось не працює. Це полегшує оновлення коду та прискорює час виходу на ринок нових функцій. Плюс до всього, легко виділити і виправити несправності і баги в окремих сервісах.	Проблеми налагодження - Кожен мікросервіс має свій набір логів, що ускладнює налагодження. Крім того, один бізнес-процес може працювати на декількох машинах, що ще більше ускладнює налагодження.
Незалежна розгортка – Оскільки мікросервіси є окремими додадтками, вони дозволяють швидко і легко самостійно розгортати індивідуальні функції.	Відсутність стандартизації - Без спільної платформи може спостерігатися розділення мов, стандартів реєстрації та моніторингу.
Гнучкість технологій – Архітектура мікросервісів дає командам свободу вибору інструментів, які вони бажають.	Відсутність чіткої власності - Оскільки вводиться більше послуг, так само як і кількість команд, які керують цими службами. З часом стає важко дізнатися, які доступні послуги може використовувати команда, і до кого звернутися за підтримкою.
Висока надійність – Можна розгорнути зміни для певної служби, без загрози збити всю програму.	

Порівняння схем побудови монолітної та мікросервісної архітектури наведено на рис. 2.3.

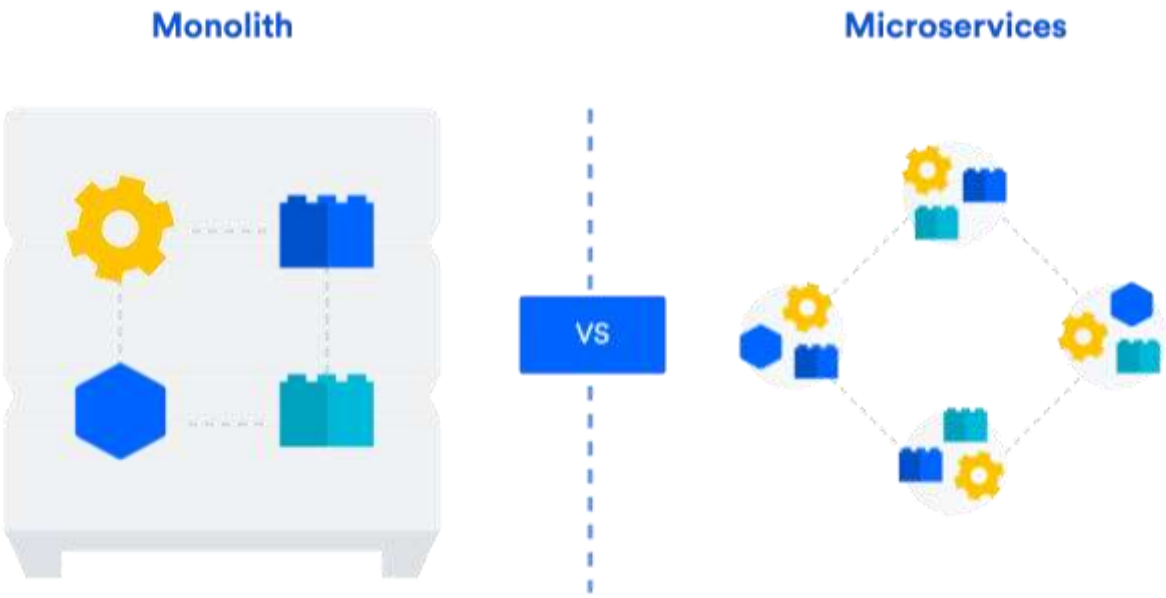


Рисунок 2.3 – Порівняння монолітної та мікросервісної архітекутур

Для інформаційної технології організації блог-систем було обрано монолітний підхід так як простіший у побудові, тестуванні та розгортуванні.

2.5 Обґрунтування вибору патернів проектування інформаційної технології організації блог-систем

Патерн проектування — це типовий спосіб вирішення певної проблеми, що часто зустрічається при проектуванні архітектури програм [15].

На відміну від готових функцій чи бібліотек, патерн не можна просто взяти й скопіювати в програму. Патерн являє собою не якийсь конкретний код, а загальний принцип вирішення певної проблеми, який майже завжди треба підлаштовувати для потреб тієї чи іншої програми.

Описи патернів зазвичай дуже формальні й найчастіше складаються з таких пунктів [15]:

– проблема, яку вирішує патерн;

– мотивація щодо вирішення проблеми способом, який пропонує патерн;

– структура класів, складових рішення;

– приклад однією з мов програмування;
– особливості реалізації в різних контекстах;
– зв’язки з іншими патернами.

Такий формалізм опису дозволив зібрати великий каталог патернів, додатково перевіривши кожен патерн на дієвість.

Патерни відрізняються за рівнем складності, деталізації та охоплення проектованої системи. Проводячи аналогію з будівництвом, ви можете підвищити безпеку на перехресті, встановивши світлофор, а можете замінити перехрестя цілою автомобільною розв’язкою з підземними переходами.

Найбільш низькорівневі та прості патерни — ідіоми. Вони не дуже універсальні, позаяк мають сенс лише в рамках однієї мови програмування.

Найбільш універсальні — архітектурні патерни, які можна реалізувати практично будь-якою мовою. Вони потрібні для проектування всієї програми, а не окремих її елементів [15].

Розглянемо деякі з патернів проектування, які використовуються у мові програмування JavaScript.

Модуль. Патерн представляє собою незалежний фрагмент коду, який можна змінювати, не впливаючи на інший код проекту. Модулі також уникають забруднення ділянок видимості, створюючи окремі ділянки видимості для змінних, які вони оголошують. Модулі, написані для одного проекту, можуть бути повторно використовуватися в інших проектах, якщо їх механізми універсальні і не прив'язані до специфіки конкретного проекту [16].

Модулі є невід'ємною частиною будь-якої сучасної програми JavaScript.

Вони допомагають підтримувати код в чистоті, розділяти код на значущі частини і допомагають організувати його. JavaScript має безліч способів створення модулів, одним з яких є шаблон модуля.

На відміну від інших мов програмування, JavaScript не має модифікаторів доступу.

Тобто змінні не можуть бути оголошені приватними або публічними. В результаті, патерн «Модуль» також використовується для емуляції концепції інкапсуляції.

Одинак — це породжувальний патерн проектування, який гарантує, що клас має лише один екземпляр, та надає глобальну точку доступу до нього[10].

Одинак вирішує відразу дві проблеми (порушуючи принцип єдиного обов’язку класу) [16]:

1. Гарантує наявність єдиного екземпляра класу. Найчастіше за все це корисно для доступу до якогось спільного ресурсу, наприклад, бази даних.
2. Надає глобальну точку доступу. Це не просто глобальна змінна, через яку можна дістатися до певного об’єкта. Глобальні змінні не захищені від запису, тому будь-який код може підмінити їхнє значення без вашого відома.

Одним з найбільш часто використовуваних патернів для роботи з даними є шаблон "Репозиторій". Репозиторій дозволяє абстрагуватися від конкретних з'єднань з джерелами даних, з які працює програма, і є проміжним зв'язком між класами, які взаємодіють безпосередньо з даними та рештою програми[9].

Спостерігач — це поведінковий патерн проектування, який створює механізм підписки, що дає змогу одним об’єктам стежити й реагувати на події, які відбуваються в інших об’єктах [15].

Конструктор. Патерн використовується для створення нових об'єктів в їх власній області видимості і дозволяє всім екземплярам об’єкта посилатися на нього без повторення функції [16].

Для розробки проектування інформаційної технології організації блог-систем було обрано такі патерни, як: Модуль, Конструктор, Спостерігач.

2.6 Розробка структури інформаційної технології організації блог-систем

Розробимо загальну структурну схему інформаційної технології організації блог-систем. Технологія складається з 5 основних компонентів:

– веб-клієнт;
– інтерфейс користувача;
– сервер;
– база даних;
– модуль автентифікації

Опишемо кожен із компонентів.

Веб-клієнт представлятиме собою систему, де зареєстровані користувачі зможуть публікувати власні пости та коментарі. Клієнт отримуватиме дані з сервера та надсилатиме опубліковані дописи на аналіз.

Інтерфейс користувача представляє собою візуалізацію блога, профілів користувачів, постів та коментарів до них.

Сервер включає обробку даних, валідацію запитів, збереження, редагування та видалення записів у базі даних.

База даних виконує функцію збереження даних про користувачів та опубліковані дописи.

Модуль автентифікації виконує функцію автентифікації та деавтентифікації користувачів.

Основна структурна схема роботи системи показана на рисунку 2.1

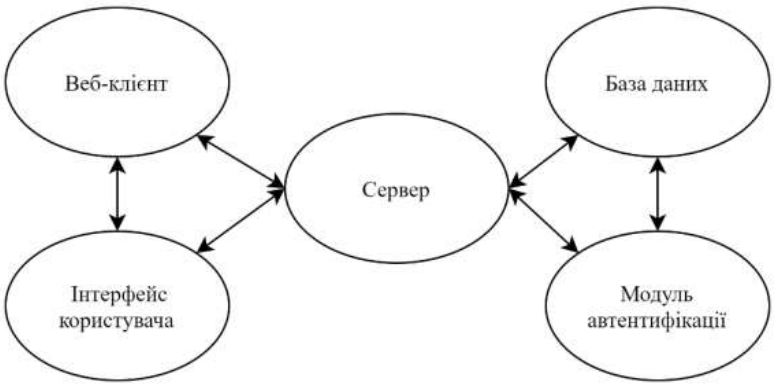


Рисунок 2.1 – Загальна структурна схема інформаційної технології організації блог-систем

2.7 Особливості організації процесу автентифікації

Для забезпечення безпеки та контролю за контентом який потрапляє у блог, в систему було додано модуль автентифікації. автентифікації використовувалась бібліотека Passport.

Passport - це проміжне програмне забезпечення для автентифікації для Node. Він призначений для єдиної мети: аутентифікації запитів. Під час написання модулів інкапсуляція є достоїнством, тому Passport делегує всі інші функції додатку. Таке розділення проблем забезпечує чіткість та підтримку коду та робить Passport надзвичайно простим для інтеграції в додаток [17].

У сучасних веб-додатках автентифікація може приймати різні форми. Традиційно користувачі входять, вводячи ім’я користувача та пароль. З розвитком соціальних мереж єдиний вхід за допомогою постачальника OAuth, такого як Facebook або Twitter, став популярним методом автентифікації. Служби, які надають API, часто вимагають облікових даних на основі маркерів для захисту доступу. Passport визнає, що кожна програма має унікальні вимоги щодо автентифікації. Механізми автентифікації, відомі як стратегії, упаковані як окремі модулі. Додатки можуть вибирати, які стратегії використовувати, не створюючи зайвих залежностей [17].

Для розробки блогу використовувалась локальна стратегія автентифікації, з використанням хешування паролів за допомогою секретного слова, або фрази. Між клієнтом та сервером дані користувача передаються за допомогою хешованого JSON Web Token (JWT).

Автентифікація поєднує у собі 3 основни компоненти:

1. модуль автентифікації – приймає дані з сервера та бази даних і описує стратегію автентифікації;

2. модуль хешування паролів – порівнює паролі з бази даних, з хешованим паролем з сервера, та повертає результат у модуль автентифікації

3. бібліотека Passport.js – опрацьовує результат порівняння паролів та конвертує дані користувача у JWT.

Структурна схема роботи автентифікації показана на рисунку 2.2.

Рисунок 2.2 – Структурна схема автентифікації

2.8 Проектування інформаційної технології організації блог-систем

Проектування – один з найважливіших етапів життєвого циклу створення програмних додатків. В ході проектування архітектором або досвідченим програмістом створюється проектна документація, що включає текстові описи, діаграми, моделі майбутньої програми. У цій нелегкій справі нам допоможе мова UML [18].

UML - є графічною мовою для візуалізації, опису параметрів, конструювання та документування різних систем (програм зокрема). Діаграми створюються за допомогою спеціальних CASE засобів, наприклад Rational Rose і Enterprise Architect. На основі технології UML будується єдина інформаційна модель. Наведені вище CASE засоби здатні генерувати код на різних об'єктно-орієнтованих мовах, а так само мають дуже корисною функцією реверсивного інжинірингу [18].

Основні типи діаграм [18]:

– діаграма варіантів використання (use case diagram);
– діаграма класів (class diagram);
– діаграма станів (statechart diagram);
– діаграма послідовності (sequence diagram);
– діаграма кооперації (collaboration diagram);

– діаграма компонентів (component diagram);
– діаграма розгортання (deployment diagram).

Діаграма класів служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. Діаграма класів може відбивати, зокрема, різні взаємозв'язки між окремими сутностями предметної області, такими як об'єкти і підсистеми, а також описує їх внутрішню структуру (поля, методи ...) і типи відносин (спадкування, реалізація інтерфейсів ...). На даній діаграмі не вказується інформація про тимчасові аспектах функціонування системи. З цієї точки зору діаграма класів є подальшим розвитком концептуальної моделі проектованої системи. На цьому етапі принципово знання ООП підходу і патернів проектування[18].

Саме діаграму класів було обрано для проектування інформаційної технології організації блог-систем, тому що вона найбільш детально описує зв’язки у спроектованій системі. Для її візуалізації було використано мову UML та інструмент PlantUML.

Діаграму класів інформаційної технології організації блог-систем зображено на рисунку 2.3. Додаток складається з шести основних класів: Blog, Post, Comment, User, API, Database.

У класі Blog зберігаються пости, дані про користувача, реалізовано завантаження постів.

У класі Post містяться поля, що описують основні параметри поста: назву, опис, коментарі, реалізовано додавання та видалення коментарів.

Клас Comment містить інформацію про коментар залишений користувачем, реалізовано редагування коментаря.

Клас User містить інформацію про користувача.

В класі API реалізовано завантаження даних з бази даних.

Клас Database містить дані, які використовуються системою.

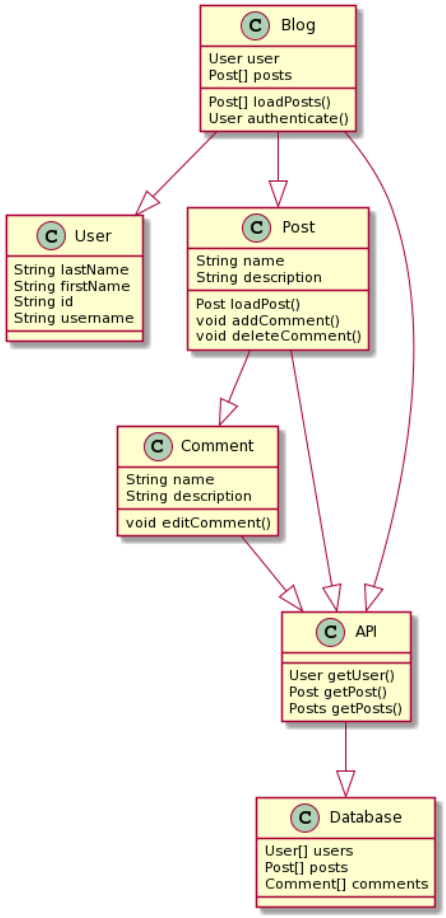


Рисунок 2.3 – Діаграма класів інформаційної технології організації блог-систем

2.9 Висновок до розділу 2

Під час проектування інформаційної технології організації блог-систем було розглянуто класифікацію блог-платформ. Проведено аналіз існуючих підходів, зокрема: професійних, напівпрофесійних та масових блог-платформ. Було обрано підхід до розробки блог-системи, а саме професійний.

Було розглянуто сучасні платформи для розробки блог систем, такі як: CMS, тобто використання готових шаблонів, та повноцінна реалізація блогу, як користувальницького веб-сайту. Було обрано мішаний підхід, на даному етапі, орієнтований на розробку користувальницького веб-сайту.

Обґрунтувано необхідність використання патернів проектування для розробки системи. Розглянуто найбільш використовувані патерни

проектування мови JavaScript, а саме: Модуль, Одинак, Репозиторій, Спостерігач, Конструктор. Обрано патерни, які найбільше підходять для розробки блогу: Модуль, Конструктор, Спостерігач.

Описано 5 основних компоненти системи: веб-клієнт, інтерфейс користувача, сервер, база даних, модуль автентифікації. Розроблено структурну схему їх зв’язків.

Обґрунтовано необхідність проектування та розглянуто основні види діаграм для проектування програмних додатків. Для проектування блогу було обрано діаграму класів, розроблену за допомогою мови UML використовуючи інструмент PlantUML.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ОРГАНІЗАЦІЇ БЛОГ-СИСТЕМ

3.1 Обґрунтування вибору технологій розробки

На сьогоднішній день найпопулярнішими мовами програмування для розробки веб-додатків є: Java, Python та TypeScript.

Java – об'єктно-орієнтована мова програмування. В офіційній реалізації Java-програми компілюються у байт-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи. Мова значно запозичила синтаксис із C і C++. Зокрема, взято за основу об'єктну модель C++, проте її модифіковано. Усунуто можливість появи деяких конфліктних ситуацій, що могли виникнути через помилки програміста та полегшено сам процес розробки об'єктно-орієнтованих програм. Мова пристосована для виконання на різних платформах, тому поступається у швидкості іншим мовам програмування, наприклад, C++[19].

Python – інша мова програмування, що швидко розвивається. Мова має простий і зрозумілий синтаксис. Це значно спрощує процес розробки. Має динамічну типізацію та автоматичний контроль пам’яті та механізм обробки виключень. Має велику кількість готових пакетів для машинного навчання, створення нейронних мереж, аналізу даних, тому часто використовується у даних напрямках.

TypeScript (TS) — типізована мова програмування, яка компілюється в мову програмування JavaScript, що є свою чергу динамічною, об'єктно-орієнтованою мовою програмування. Найчастіше використовується як частина браузера, що надає можливість коду на стороні клієнта (такому, що виконується на пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб-сторінки. Мова JavaScript також

використовується для програмування на стороні сервера (подібно до таких мов програмування, як Java і C#), розробки ігор, стаціонарних та мобільних додатків. Незважаючи на схожість назв, мови Java та JavaScript є двома різними мовами, що мають відмінну семантику, хоча й мають схожі риси в стандартних бібліотеках та правилах іменування[20].

У таблиці 3.1 наведена порівняльна характеристика мов програмування.

Таблиця 3.1 – Порівняльна характеристика мов програмування

Критерій/мова	Java	TypeScript	Python
Ручне управління пам’яттю	+	-	-
Необхідність компіляції	+	+	-
Необхідність у роботі з браузером	-	+	-
Виконання на клієнтській частині браузера	-	+	-
Пряме підключення скриптів	-	+	-
Використання на сервері	+	+	+
Наявність пакетів для машинного навчання	-	+	+
Підтримка на всіх платформах	-	+	-

На даний момент TypeScript є головною мовою розробки веб-додатків після JavaScript. Це спричинило появу великої кількості бібліотек, фреймворків та компіляторів на основі JS, як для клієнтської частини так і для серверу. При виборі клієнтських технологій слід звернути увагу на популярні підходи на ринку, так як вони мають високу підтримуваність розробниками і постійно розвиваються. Розглянемо такі популярні технології побудови веб-додатків: Svelte, React, Vue.js.

Svelte - це інструмент для створення швидких web-додатків.

У Svelte додаток складається з одного або декількох компонентів. Компонент - це багаторазово використовуваний автономний блок коду, який

ізолює розмітку, стилі і поведінку, які пов'язані один з одним і записані в єдиний файл .svelte.

Svelte забезпечує інший підхід до створення веб-додатків, ніж деякі інші фреймворки. У той час як фреймворки, такі як React і Vue, виконують основну частину своєї роботи в браузері користувача під час роботи програми, Svelte переводить цю роботу на крок компіляції, який відбувається лише при розробці власного додатку, створюючи високооптимізований звичаний JavaScript [21].

Результатом такого підходу є не тільки менші пакети додатків і краща продуктивність, але й досвід розробника, який є більш доступним для людей, які мають обмежений досвід сучасної екосистеми інструментів [21].

Будучи компілятором, Svelte може розширювати HTML, CSS та JavaScript, генеруючи оптимальний код JavaScript без накладних витрат на виконання. Щоб досягти цього, Svelte розширює звичайні веб-технології наступними способами [21]:

– розширення HTML, дозволяючи вирази JavaScript у розмітці та надаючи директиви для використання умов та циклів, подібно до Handlebars.js.

– розширення CSS, додаючи механізм scoping, що дозволяє кожному компоненту визначати власні стилі без ризику зіткнення зі стилями інших компонентів.

– розширення JavaScript шляхом переосмислення конкретних директив мови для досягнення справжньої реактивності та полегшення управління компонентним станом.

Компілятор втручається тільки в дуже конкретні ситуації і тільки в контексті компонентів Svelte. Розширення для мови JavaScript мінімальні і ретельно підібрані, щоб не порушити синтаксис JavaScript або не відштовхнути розробників. Насправді, ви будете в основному працювати з звичайним JavaScript [21].

Він схожий на такі JavaScript фреймворки, як React і Vue, які переслідують спільну мету - полегшити створення зручних призначених для користувача інтерфейсів [21].

Але є суттєва відмінність: Svelte перетворює ваш додаток в ідеальний JavaScript під час збирання, а не інтерпретує код додатка в своєму середовищі виконання. Це означає, що продуктивність не втрачається за рахунок абстракції фреймворка і час першого завантаження програми значно менший [21].

За допомогою Svelte можна зібрати додаток цілком або поступово додати його в існуючий проект. Також можна створювати компоненти у вигляді автономних пакетів, які можуть працювати де завгодно, не вимагаючи зайвих залежностей, на відміну від звичайних фреймворків [21].

Тег в нижньому регістрі, наприклад - це звичайний HTML елемент. Тег, написаний з великої літери, такий як <Widget> або <Namespace.Widget>, позначає компонент [22].

Переваги Svelte [23]:

– Svelte – компілятор. Інші популярні веб-фреймворки включають великі бібліотеки середовища виконання для підтримки всіх їх функцій. Але Svelte - це не бібліотека виконання. Це компілятор веб-додатків, реалізований у TypeScript.

– Svelte вимагає менше коду ля реалізації тієї ж функції

– Svelte забезпечує реактивність без використання віртуального DOM

– Svelte швидкий. Результати бенчмарків представлено у таблиці 3.2.

– Svelte вимагає менше пам'яті. Використання меншої кількості пам'яті є суттєвою перевагою, коли веб-програми працюють на старих комп’ютерах або мобільних пристроях, які, як правило, мають менше доступної пам’яті для запущених програм.

Таблиця 3.2 – Порівняльна характеристика продуктивності основних технологій

Ознака	vue- v3.2.37	svelte - v3.50.1	preact - v10.7.3
постійна інтерактивність (песимістичний TTI - коли процесор і мережа безумовно простоюють)	2,142 ± 47.4	1,865.1 ± 5.4	1,916.4 ± 128.5
повний розбір в кілобайтах	196.5 ± 0.0	146.2 ± 0.0	154.9 ± 0.0

Недоліки Svelte:

– Складність у налагоджені. Скомпільований додаток важче протестувати. Також у Svelte досить невелика спільнота розробників, що робить відлагодження компілятора важчим.

– Svelte не є особливо вдалим варіантом для веб-програм, які повинні працювати в Internet Explorer.

– Відсутність підтримки багатьох бібліотек, доступних для інших фрейморків.

Vue - це прогресивний фреймворк для створення користувацьких інтерфейсів. На відміну від фреймворків-монолітів, Vue створений придатним для поступового впровадження. Його ядро в першу чергу вирішує завдання рівня уявлення (view), що спрощує інтеграцію з іншими бібліотеками та існуючими проектами. З іншого боку, Vue повністю підходить і для створення складних односторінкових додатків (SPA, Single-Page Applications), якщо використовувати його спільно з сучасними інструментами та додатковими бібліотеками [24].

Vue сучасний фреймворк JavaScript, який надає корисні засоби для прогресивного вдосконалення — на відміну від багатьох інших фреймворків, ви можете використовувати Vue для покращення існуючого HTML. Це дозволяє використовувати Vue як заміну для такої бібліотеки, як jQuery [21].

З огляду на це, аткож можна використовувати Vue для написання цілих односторінкових програм (SPA). Це дозволяє створювати розмітку, керовану повністю Vue, що може поліпшити досвід розробника і продуктивність при роботі зі складними додатками. Це також дозволяє використовувати переваги бібліотек для маршрутизації на стороні клієнта та управління державою, коли це потрібно. Крім того, Vue використовує підхід «золотої середини» до таких інструментів, як маршрутизація на стороні клієнта та управління станом. Хоча основна команда Vue підтримує запропоновані бібліотеки для цих функцій, вони не об'єднані безпосередньо в Vue. Це дозволяє вибрати іншу бібліотеку маршрутизації / управління станом, якщо вони краще відповідають вашій програмі [21].

На додаток до того, що дозволяє поступово інтегрувати Vue у існуючі програми, Vue також забезпечує прогресивний підхід до написання розмітки. Як і більшість фреймворків, Vue дозволяє створювати багаторазові блоки розмітки за допомогою компонентів. У більшості випадків компоненти Vue пишуться за допомогою спеціального синтаксису шаблону HTML. Якщо потрібно більше елементів керування, ніж дозволяє синтаксис HTML, можна написати JSX або звичайні функції JavaScript для визначення компонентів [21].

Переваги Vue [25]:

– Невеликий розмір. Завантажений zip-файл із фреймворком важить 18 КБ. Це покращує швидкість завантаження та встановлення бібліотеки, а також позитивно впливає на SEO та UX.

– Віртуальний DOM і продуктивність DOM. Document Object Model (DOM) - це представлення HTML-сторінок з їх стилями, елементами та вмістом сторінки як об’єктами. Об'єкти, що зберігаються як деревоподібна структура. Коли користувач взаємодіє зі сторінкою, об’єкти змінюють свій стан, так що браузер повинен оновити інформацію та відобразити її на екрані. Але, оновлення всього DOM є громіздким. Для швидкості Vue.js використовує віртуальний DOM: копію оригінального DOM, який визначає, які елементи

оновлювати, не відтворюючи цілий DOM. Цей підхід робить візуалізацію сторінок досить швидким та покращує продуктивність програми., створюються браузером під час завантаження сторінки.

– Реактивне двостороннє прив'язка даних. Двостороння прив'язка даних була успадкована Vue від Angular. Двостороння прив'язка даних - це зв'язок між оновленням даних моделі та поданням (UI). Зв’язані компоненти містять дані, які можна час від часу оновлювати. За допомогою двостороннього прив’язки даних простіше оновлювати пов’язані компоненти та відстежувати оновлення даних. У Vue пов'язані дані оновлюються реактивно, як і об'єкти DOM, що чудово підходить для будь-якої програми, яка вимагає оновлення в режимі реального часу. З точки зору розробників, реактивність Vue зробить оновлення даних зрозумілішим та простішим для заповнення.

– Велика екосистема інструментів. За 5 років існування Vue.js отримав потужний набір інструментів для роботи. Vue також має власні засоби налагодження браузера, візуалізатор сервера та менеджер стану.

– Підтримка спільноті. Vue.js існує завдяки своїй спільноті. Через це учасники спільноти досить активні як у чаті Discord, так і на форумі. Як доказ діяльності спільноти, просто подивіться на кількість тегів Vue.js у Stack Overflow, яка наразі нараховує понад 41 тисячу .

Недоліки Vue [18]:

– Мовний бар'єр. Впровадження Vue на таких підприємствах, як Xiaomi та Alibaba, допомогло популяризувати структуру та створило попит на ринку праці. Зважаючи на те, що Vue.js стає все більш популярним у Китаї, значна частина контенту та дискусій відбувається, як не дивно, китайською.

– Складність реактивності. Програма Vue.js складається з компонентів, з якими користувач може взаємодіяти. Кожен компонент має свого спостерігача, який рендерить дані кожного разу, коли користувач запускає компонент. Система реактивності відображає лише ті фрагменти даних, які були ініційовані. Справа в тому, що він не такий розумний і часто

допускає помилки під час зчитування даних, тому вимагає вирівнювання даних.

– Відсутність підтримки масштабних проектів. Vue.js - все ще молодий фреймворк. Розмір спільноти та команди розробників досі незрівнянний із більш зрілим React. Він також не користується фінансовою підтримкою великих підприємств. Ця технологія має бути стабільною та підтримуватися, щоб швидко вирішити проблеми. Хоча у Vue не так багато проблем з цим, і навіть існує попит з боку таких підприємств, як IBM та Adobe, він все ще використовується у відносно невеликих проектах.

– Відсутність досвідчених розробників. Vue.js - відносно молода технологія, яка тільки почала набирати популярність. Але, здається, нам доведеться почекати пару років, поки його масове прийняття на ринку праці заповниться досвідченими розробниками Vue.js.

Preact - це бібліотека JavaScript, створена для створення швидких та інтерактивних користувальницьких інтерфейсів для веб- та мобільних додатків. Це відкрита, компонентна, інтерфейсна бібліотека, відповідальна лише за рівень перегляду програми. В архітектурі Model View Controller (MVC) рівень подання відповідає за те, як виглядає та відчуває себе програма[26].

Preact – це менша (3kb) альтернатива React, розроблена Джейсоном Міллером та купою інших розробників. Preact був розроблений з метою створення фреймворку JavaScript, який має невеликий розмір, але при цьому пропонував той самий API та функції, з якими поставляється React [35].

Із загальним розміром 3 кб не доведеться турбуватися про те, що бібліотека/фреймворк JavaScript займе значну частину загального розміру JavaScript додатка.

Preact швидкий, і не тільки через своїх розмірів. Це одна з найшвидших віртуальних бібліотек DOM завдяки простій і передбачуваній реалізації порівняння.

Preact має на меті досягнення кількох ключових цілей:

– Продуктивність: Візуалізація відбувається швидко та ефективно

– Невеликий розмір: Невеликий розмір

– Ефективність: Ефективне використання пам'яті

– Зрозумілість: Опановування кодової бази має зайняти не більше декількох годин

– Сумісність: Preact має на меті бути значною мірою сумісним з React API. preact-compat намагається досягти якомога більшої сумісності з React.

Preact добре працює з браузерами (підтримує всі браузери), хоча для нього можуть знадобитися деякі поліфіли для IE7 і IE8. Preact також отримав велике схвалення спільнотою з безліччю плагінів, і тепер компанії починають переходити на бібліотеку JavaScript.

Preact зараз використовується такими компаніями, як Lyft, Housing.com tam.uber.com. Інженерна команда Uber нещодавно написала статтю, в якій висвітлила, як Preact використовувався для створення мобільної версії Uber, що, у свою чергу, призвело до кращої продуктивності багато в чому завдяки її дуже мінімальному розміру.

Особливості Preact[27]:

– JSX. JSX - це розширення синтаксису для JavaScript. Він використовується з Preact для опису того, як повинен виглядати користувальницький інтерфейс. Використовуючи JSX, ми можемо писати HTML-структури в одному файлі, що містить код JavaScript. Це полегшує розуміння та налагодження коду, оскільки дозволяє уникнути використання складних структур DOM JavaScript

– Віртуальний DOM. React зберігає в пам'яті полегшене уявлення про «реальний» DOM, який відомий як «віртуальний» DOM (VDOM). Маніпулювання реальним DOM відбувається набагато повільніше, ніж маніпулювання VDOM, оскільки на екрані нічого не малюється. Коли стан об'єкта змінюється, VDOM змінює лише цей об'єкт у реальному DOM, замість того, щоб оновлювати всі об'єкти.

– Продуктивність. React використовує VDOM, що змушує веб-програми працювати набагато швидше, ніж ті, що розробляються з альтернативними інтерфейсними платформами. React розбиває складний користувальницький інтерфейс на окремі компоненти, дозволяючи декільком користувачам одночасно працювати над кожним компонентом, тим самим прискорюючи час розробки.

– Розширення. Preact виходить за рамки простого дизайну інтерфейсу та має безліч розширень, які пропонують повну підтримку архітектури додатків. Він забезпечує візуалізацію на стороні сервера, що передбачає рендеринг веб-програми, яка зазвичай виконується лише на стороні клієнта, на сервері, а потім надсилає клієнтові повністю відтворену сторінку. Він також широко використовує Flux та Redux у розробці веб-додатків.

– Одностороння прив'язка даних. Одностороннє прив’язування даних Preact забезпечує модулярність та швидкість. Односпрямований потік даних означає, що коли розробник розробляє додаток React, він часто вкладає дочірні компоненти в батьківські компоненти. Таким чином, розробник знає, де і коли виникає помилка, надаючи їм кращий контроль над усією веб-програмою.

– Налагодження. Додатки Preact легко протестувати завдяки великій спільноті розробників. Facebook навіть пропонує невелике розширення браузера, що робить налагодження Preact швидшим та простішим.

Для розробки інформаційної технології організації блог-систем було обрано використання бібліотеки Preact, як найшвидшої та найлегшої у розширені.

Як асинхронне подієве JavaScript–оточення, Node.js спроектований для побудови масштабованих мережевих додатків. Додаток може одночасно обробляти багато з’єднань. Для кожного з’єднання викликається функція зворотнього виклику, проте коли з’єднань немає Node.js засинає[28].

Це контрастує з більш загальною моделлю в якій використовуються паралельні OS потоки. Такий підхід є відносно неефективним та дуже важким у використанні. Більше того, користувачі Node.js можуть не турбуватись про блокування процесів, оскільки немає жодних блокувань. Майже жодна з функцій у Node.js не працює напряму з I/O, тому процес не блокується ніколи. Оскільки нічого не блокується на Node.js легко розробляти масштабовані системи [28].

Node.js створений під впливом таких систем як Event Machine в Ruby або Twisted в Python. Node.js використовує подієву модель значно ширше, він приймає цикл подій (event loop) за основу оточення, замість того, щоб використовувати його в якості бібліотеки. В інших системах завжди стається блокування виклику, щоб запустити цикл подій [28].

Зазвичай поведінка визначається через функції зворотнього виклику на початку скрипта і в кінці запускає сервер через блокуючий виклик, як от EventMachine::run(). В Node.js немає нічого подібного на виклик початку циклу подій. Node.js просто входить в подієвий цикл після запуску скрипта на виконання. Node.js виходить з подієвого циклу тоді, коли не залишається зареєстрованих функцій зворотнього виклику. Така поведінка схожа на поведінку браузерного JavaScript: подієвий цикл прихований від користувача [28].

HTTP є об'єктом першого роду в Node.js, розробленим з потоковістю та малою затримкою. Це робить Node.js хорошою основою для веб–бібліотеки або фреймворку [28].

Те що Node.js спроектований без багатопоточності, не означає, що ви не можете використовувати можливості кількох ядер у вашому середовищі. Ви можете створювати дочірні процеси, якими легко керувати з допомогою API child_process.fork(). Модуль cluster побудований на цьому інтерфейсі і дозволяє вам ділитись сокетами між процесами та розподіляти навантаження між ядрами [28].

На стороні серверу популярними підходи на ринку, є використання таких фреймворків, як: Express.js та Koa.js.

Express - це мінімалістичний та гнучкий фреймворк для веб-застосунків, побудованих на Node.js, що надає широкий набір функціональності. Маючи в своєму розпорядженні безліч допоміжних HTTP-методів та проміжних обробників, створювати надійні API можна легко і швидко. Express забезпечує тонкий прошарок базової функціональності для веб-застосунків, що не спотворює звичну та зручну функціональність Node.js [29].

Для налаштування сервера з Express потрібно дуже мало типового коду. Завдяки зрілості та популярності Express, він підтримується великою документацією та має чудову підтримку спільноти, тому, якщо при розробці виникнуть проблеми, є безліч ресурсів, які допоможуть у налагодженні. Він також дуже стабільний і підтримується Node.js Foundation Incubator [30].

Попри всі свої переваги, Express не позбавлений і своїх недоліків. Він занадто трудомісткий, тобто вимагає від розробників вручну створювати всі кінцеві точки, що означає, що чим більший ваш код, тим важче його змінити. Згодом вам потрібно бути дуже організованими, щоб підтримувати код, тому що легко загубитися у всьому проміжному програмному забезпеченні. Також він не має вбудованої обробки помилок, на відміну від Koa [30].

Koa - це новий веб-фреймворк, розроблений командою Express, яка має на меті стати меншою, виразнішою та надійнішою основою для веб-додатків та API. Використовуючи асинхронні функції, Koa дозволяє відмовитись від зворотних викликів і значно збільшити обробку помилок. Koa не поєднує жодних проміжних програм у своєму ядрі, і він пропонує елегантний набір методів, які роблять написання серверної частини швидкимм та приємним[31].

За допомогою Koa простіше писати проміжне програмне забезпечення завдяки гнучкості фреймворку. Він був розроблений, для приємності написання та читання. Він також використовує функції ES6 async / await, замість використання зворотних викликів. Іншим важливим фактором є невеликий розмір Koa. Він дуже легкий - близько 500 рядків коду[30].

Хоча Koa може новіший, він також менш стабільний, ніж старіший Express. Спільнота з відкритим кодом також порівняно менша, що означає меншу підтримку. Хоча async / await виключає необхідність зворотних викликів, декілька асинхронних викликів одночасно все одно можуть призвести до проблем з async / await. Проміжне програмне забезпечення Koa також не сумісне з іншими фреймворками [30].

Для розробки інформаційної організації блог-систем було обрано використання фреймворку Express з фреймворком Nest.js, як більш розширюваного та функціонального.

Розглянемо три системи управління базами даних: MongoDB, PostgreSQL та DynamoDB.

MongoDB це крос-платформна, потужна, гнучка, документно-орієнтована база даних, що легко масштабується. Вона також має вбудовану підтримку MapReduce-style Aggregation та Geospatial індексів. MongoDB складається з баз даних, які зберігають в собі колекції. Кожна колекція складається з документів, які в свою чергу складаються з полів. Поля є парами ключ-значення. Колекції також можуть бути індексовані, що покращує швидкість вибірки та сортування [32].

Перевага використання баз даних подібних до MongoDB полягає в тому, що вона надає можливість легкої роботи з даними у форматі JSON у будь-який частині вашої програми. Це значить, що ви можете передавати і зберігати дані у форматі JSON на на всіх рівнях: клієнт, сервер і, власне, база даних – MongoDB [32].

PostgreSQL - це об'єктно-реляційна система управління базами даних.

PostgreSQL - СУБД з відкритим вихідним кодом, вона підтримує більшу частину стандарту SQL і пропонує безліч сучасних функцій:

– складні запити
– зовнішні ключі
– тригери
– змінювані уявлення

– транзакційна цілісність
– багатоверсійність

Крім того, користувачі можуть всіляко розширювати можливості PostgreSQL, наприклад створюючи свої

– типи даних
– функції
– оператори
– агрегатні функції
– методи індексування
– процедурні мови

А завдяки вільній ліцензії, PostgreSQL дозволяється безкоштовно використовувати, змінювати і поширювати всім і для будь-яких цілей - особистих, комерційних чи навчальних [33].

Amazon DynamoDB - це база даних пар «ключ-значення» і документів, яка забезпечує затримку менше 10 мілісекунд при роботі в будь-якому масштабі. Це надійна повністю керована база даних для додатків в масштабі всього Інтернету, яка працює в декількох регіонах з кількома активними серверами і має убудовані засоби забезпечення безпеки, резервного копіювання та відновлення, а також кешування в пам'яті. DynamoDB може обробляти більше 10 трлн запитів в день і справлятися з піковими навантаженнями, що перевищують 20 млн запитів в секунду [34].

DynamoDB використовує JSON для свого синтаксису через його популярність у розробників. Створення таблиці вимагає лише трьох аргументів: TableName, KeySchema — список, що містить ключ секції та необов'язковий ключ сортування — та AttributeDefinitions — список атрибутів записів, серед яких повинні бути атрибути, що використовуються як ключі секції та сортування. Тоді як реляційні бази даних пропонують надійні мови запитів, DynamoDB пропонує лише операції Put, Get, Update та Delete. Запити додавання запитів (Put) містять атрибут TableName та атрибут Item, який складається з усіх атрибутів та значень, які має запис. Запит на оновлення

Update відповідає тому ж синтаксису. Аналогічно, щоб отримати записи (Get) або видалити (Delete) запис, просто вказують TableName та Key [34].

Після аналізу та порівняння вищенаведених систем управління базами даних, для розробки інформаційної технології було обрано СУБД MongoDB, яка забезпечує простоту розробки, підтримки та використання при роботі з мовою JavaScript, так як не потребує додаткових інструментів для приведення даних у необхідний формат, легко масштабується та має гнучку схему даних, що дозволяє спростити подальшу підтримку розробленого технології.

3.2 Розробка алгоритму функціонування інформаційної технології організації блог-систем

Для відображення функціонування інформаційної технології організації блог-систем використаємо текстовий опис та опис у вигляді схеми алгоритму.

Кроки алгоритму:

1. Автентифікація користувача.
2. Перевірка результату автентифікації, якщо користувача не було автентифіковано, буде увімкнено анонімний перегляд блогу.
3. Отримання дописів з сервера.
4. Якщо перегляд блогу ведеться анонімно, перейти на крок 9.
5. Введення нового допису.
6. Відправка допису на сервер.
7. Валідація допису. Якщо допис не є валідним, повернутись на крок 5.
8. Збереження даних у базі даних.
9. Отримання детальних даних про допис.
10. Візуалізація отриманих даних.
11. Якщо перегляд блогу ведеться анонімно, перейти на крок 18.
12. Вибір дії, якщо вибрано видалення, перейти на крок 14, якщо ні – перейти на крок 13.

13. Вибір дії, якщо вибрано редагування, перейти на крок 15, якщо ні, перейти на крок 18.

14. Видалення допису.

15. Редагування допису.

16. Оновлення даних у базі даних.

17. Оновлення даних на клієнті.

18. Завершення роботи

Графічна інтерпретація описаного алгоритму наведена на рисунку 3.16.

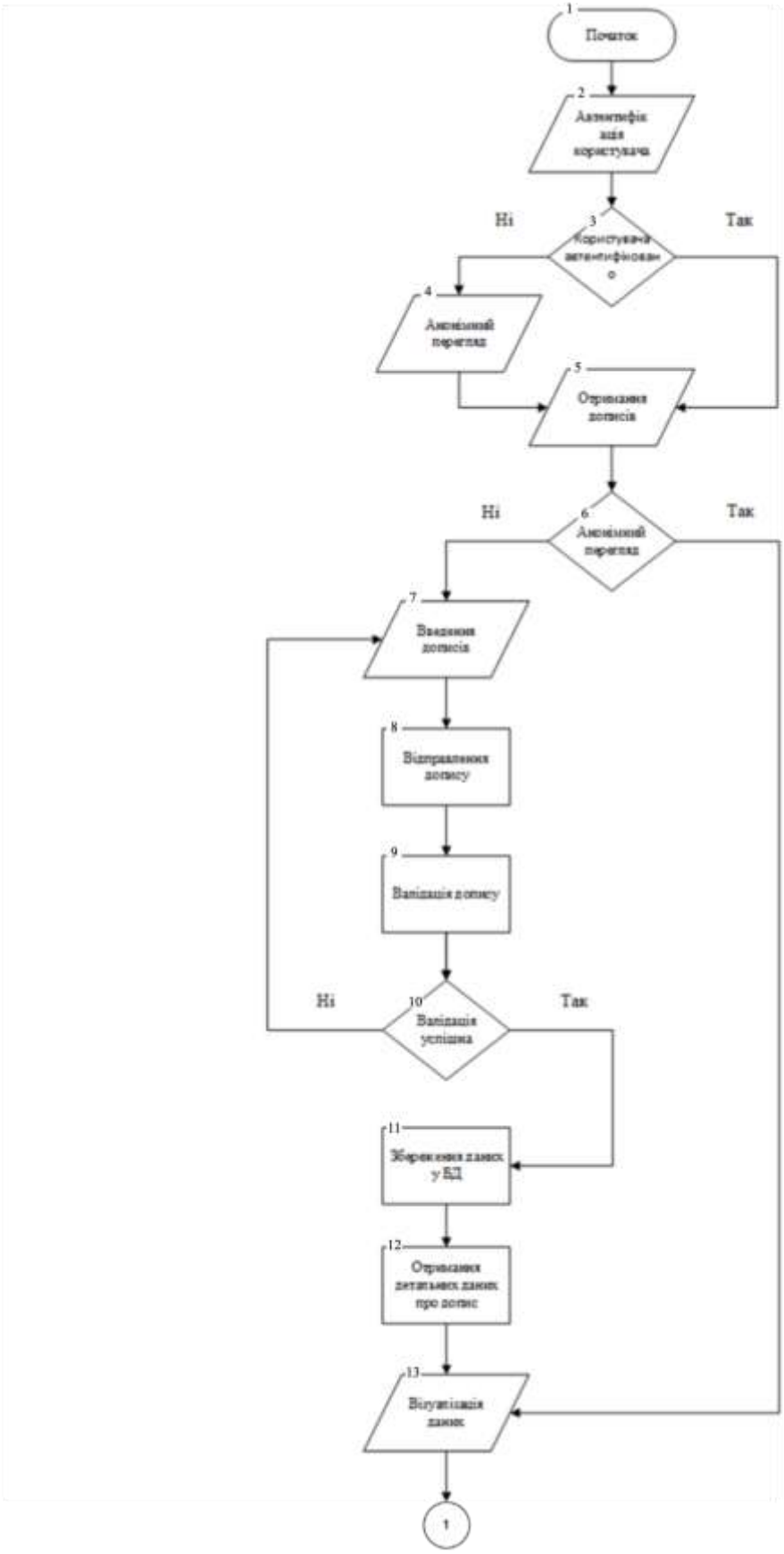


Рисунок 3.16 – Схема основного алгоритму роботи інформаційної технології організації блог-систем

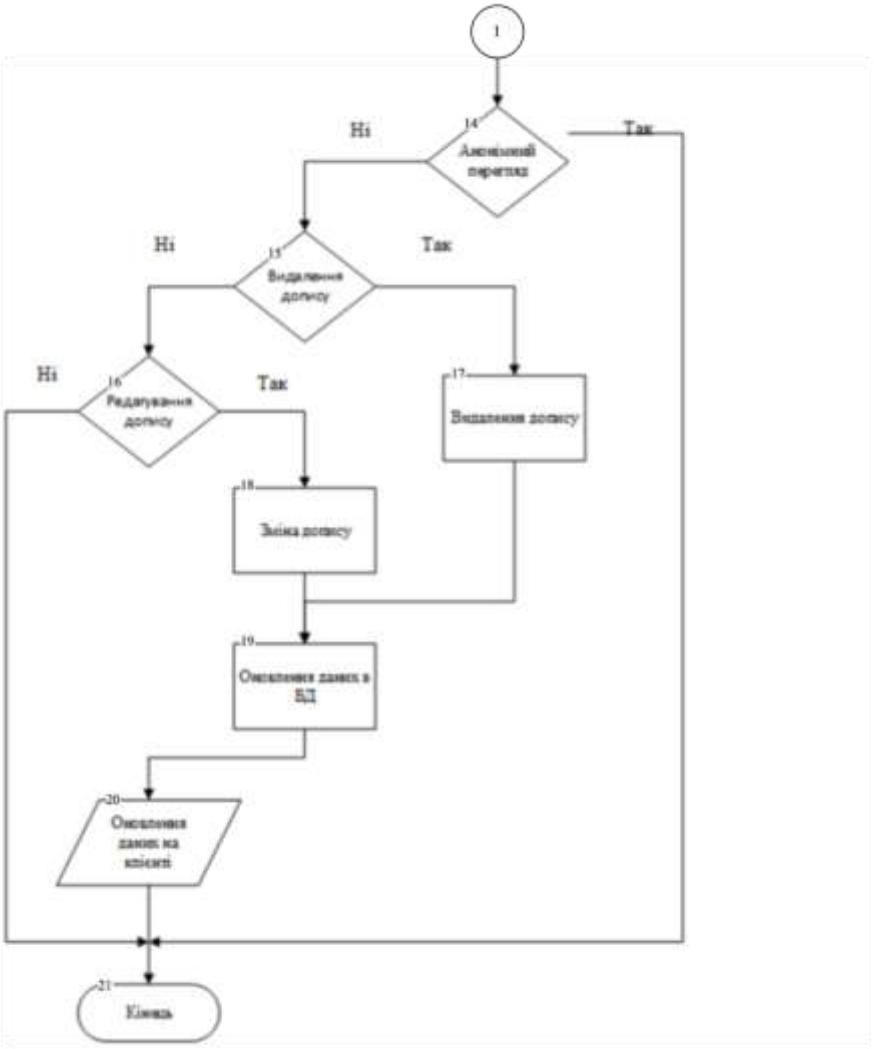


Рисунок 3.16, аркуш 2

3.3 Тестовий приклад роботи програми і аналіз результатів

Тестування клієнтської частини проводилось перевіркою дотримання необхідних вимог, а саме:

- кількість символів у пості не менша за 200;
- кількість символів у коментарі не менша за 100;

Для забезпечення гнучкості системи, ці вимоги можуть бути змінені за запитом клієнта.

Щоб перевірити дотримання вимог по кількість символів у пості, відкриємо форму створення поста. Для цього на головному екрані додатку, потрібно натиснути кнопку з зображенням олівця (рис. 3.17)

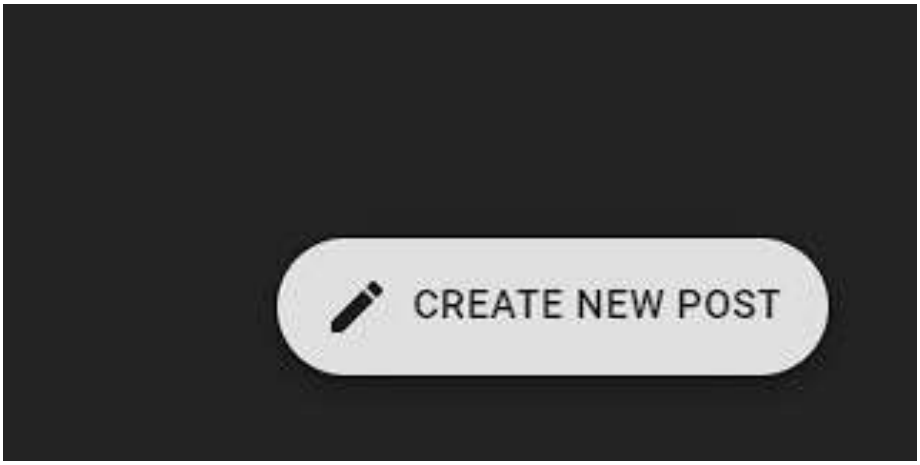


Рисунок 3.17 – Загальний вигляді інтерфейсного вікна «Кнопка створення нового поста»

Після цього відкриється форма, в якій користувач повинен ввести необхідні поля (рис. 3.18), а саме: заголовок та текст поста. За допомогою кнопок вставки зображення та силки, користувач з легкістю зможе вставити рекламний банер, або силку на спонсора цього поста. Під полем вводу зображено індикатор заповненості поста, який є відношенням поточної кількості символів до максимальної (рис. 3.19)

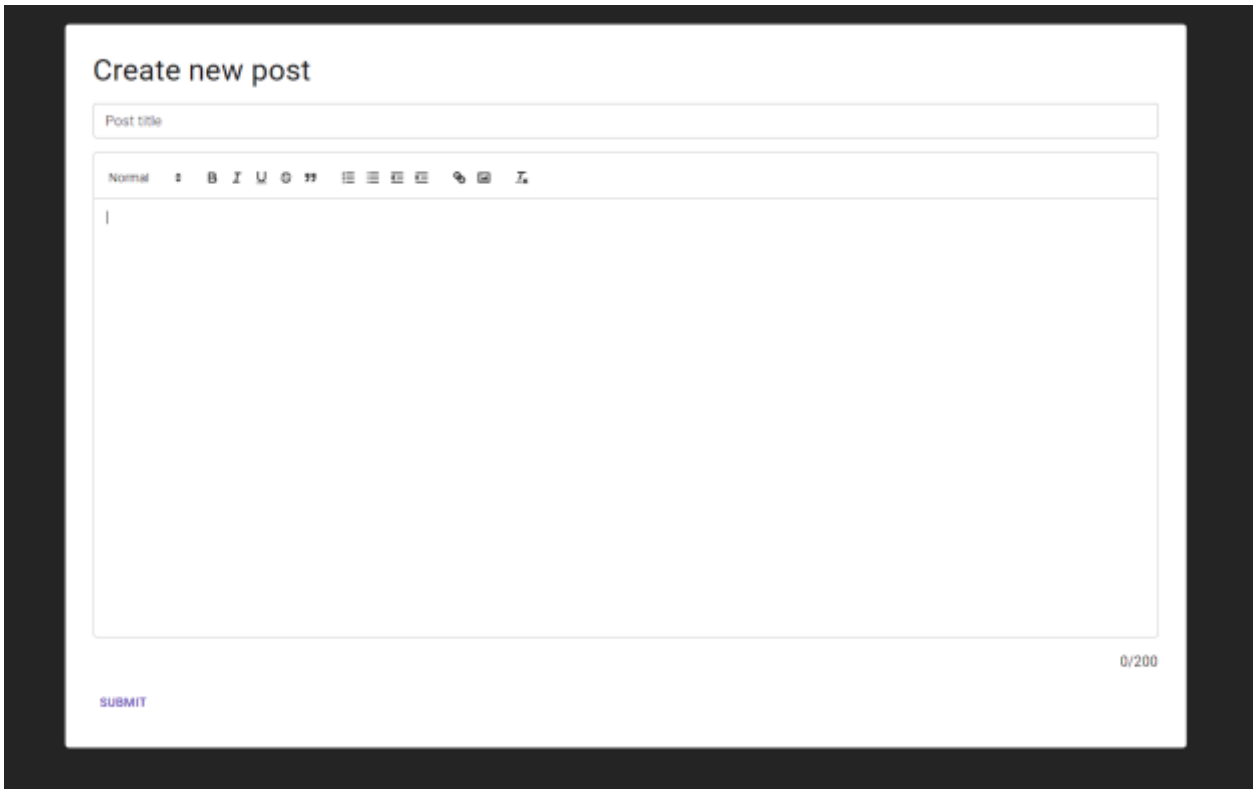


Рисунок 3.18 – Загальний вигляді інтерфейсного вікна «Форма створення нового поста»

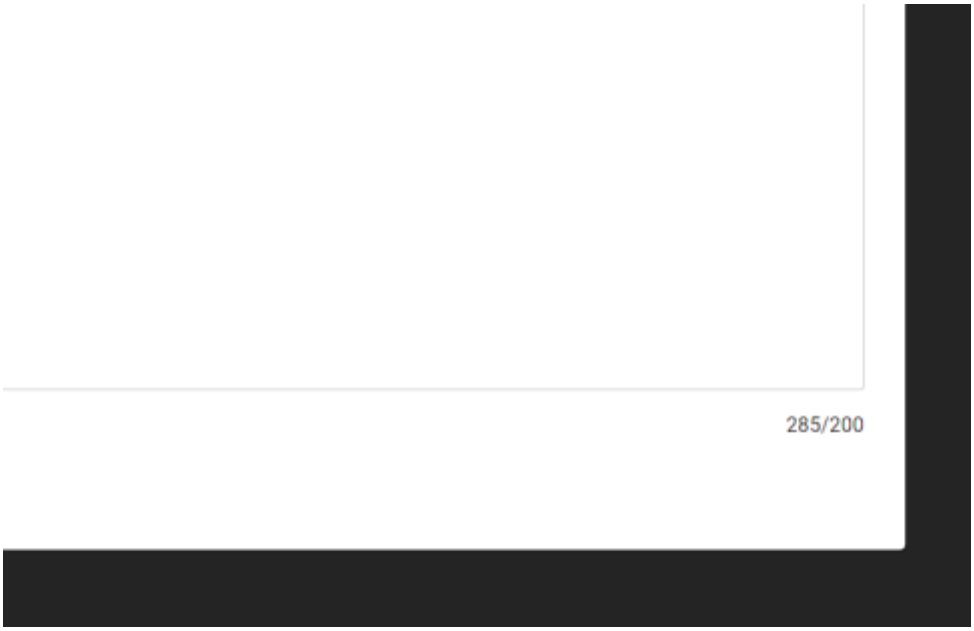


Рисунок 3.19 – Загальний вигляді інтерфейсного вікна «Індикатор заповненості поста»

Окрім цього потрібно перевірити чи працює обмеження в режимі редагування поста. Для цього на головній сторінці блогу потрібно знайти необхідний пост і натиснути кнопку (Read more…) (рис. 3.6).

Рисунок 3.20 – Відображення поста на головній сторінці

Для переходу в режим редагування, на сторінці поста (рис. 3.21) потрібно натиснути кнопку Edit.

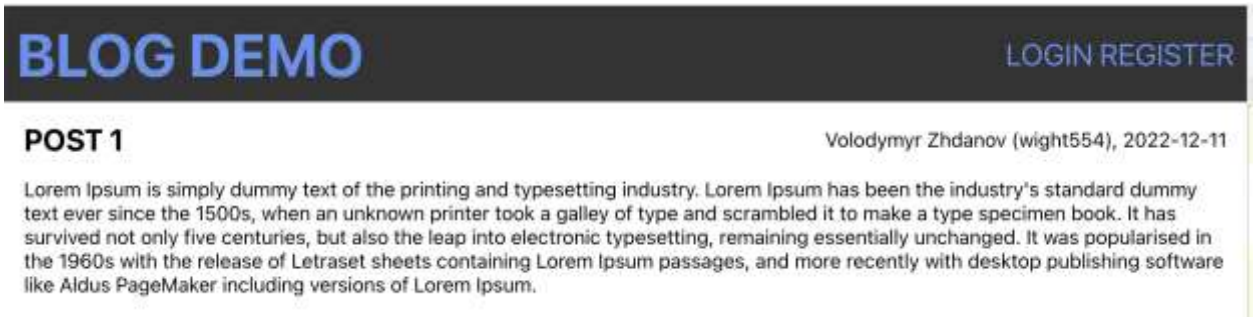


Рисунок 3.21 – Загальний вигляді інтерфейсного вікна «Сторінка поста»

Інтерфейс режиму редагування поста зображено на рис. 3.22. В ньому можна побачити індикатор, зображений на рис. 3.19.

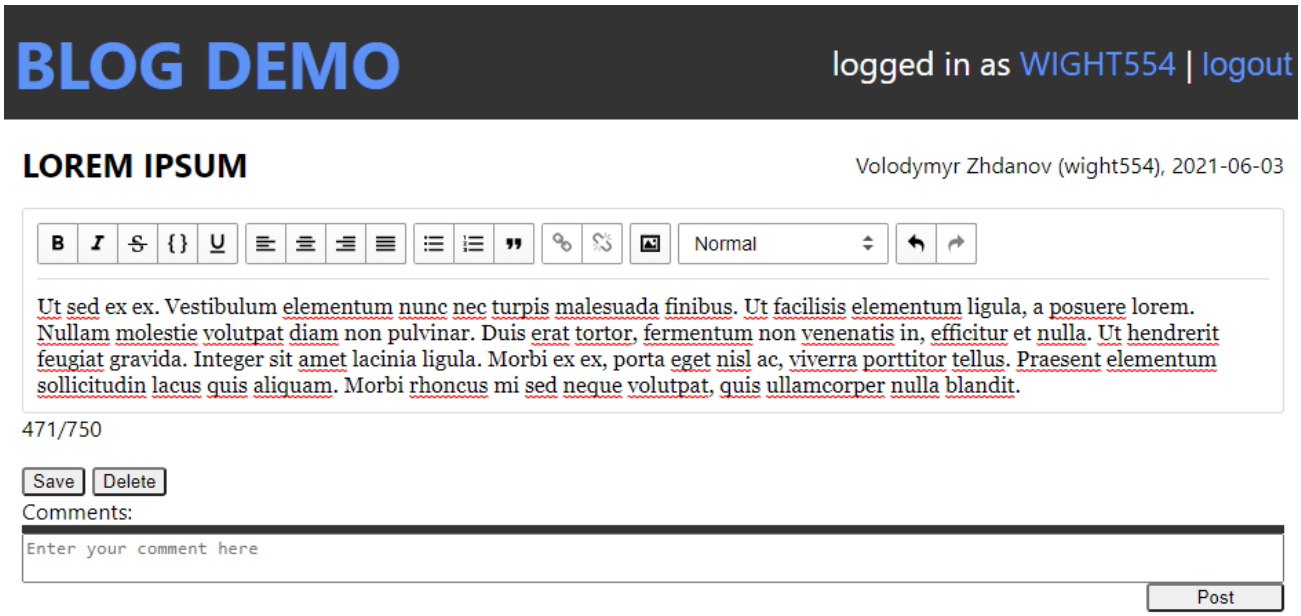


Рисунок 3.22 – Загальний вигляді інтерфейсного вікна «Режим редагування поста»

Для перевірки дотримання вимог по кількості символів у коментарі, на сторінці поста заповнимо поле вводу коментаря (рис. 3.23).

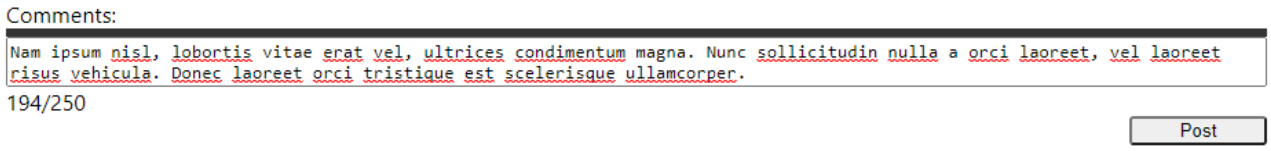


Рисунок 3.23 – Загальний вигляді інтерфейсного вікна «Поле вводу коментаря»

В ньому теж можна побачити індикатор, зображений на рис. 3.19.

Перевіримо сценарій редагування коментаря. Для цього на сторінці поста знаходимо коментар і натискаємо кнопку Edit (рис 3.24)

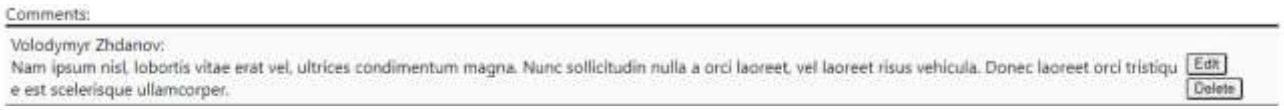


Рисунок 3.24 – Загальний вигляді інтерфейсного вікна «Коментар на сторінці поста»

В полі редагування коментаря (рис. 3.25) для зручності було використано індикатор, зображений на рис. 3.24.

Рисунок 3.25 – Загальний вигляді інтерфейсного вікна «Поле редагування коментаря»

Крім ручного тестування, систему також було протестовано за допомогою unit-тестів.

Unit-тест - це спосіб тестування найменшого шматка коду, який можна логічно виділити в системі. У більшості мов програмування це функція, підпрограма, метод або властивість. Важлива ізольована частина визначення. У своїй книзі «Ефективна робота зі застарілим кодом» автор Майкл Пірс стверджує, що такі тести не є модульними тестами, коли вони покладаються на зовнішні системи: «Якщо він розмовляє з базою даних, він розмовляє по мережі, він торкається файлової системи, це вимагає конфігурації системи, або його не можна запустити одночасно з будь-яким іншим тестом» [35].

Unit-тестування системи проводиться за допомогою бібліотеки Vitest. Це фреймворк для unit-тестування тестування на основі Vite – компілятора для TypeScript, який забезпечує найлегшу конфігурацію і є найшвидшим серед інших

Інтерфейс тестування наведено на рис. 3.26.

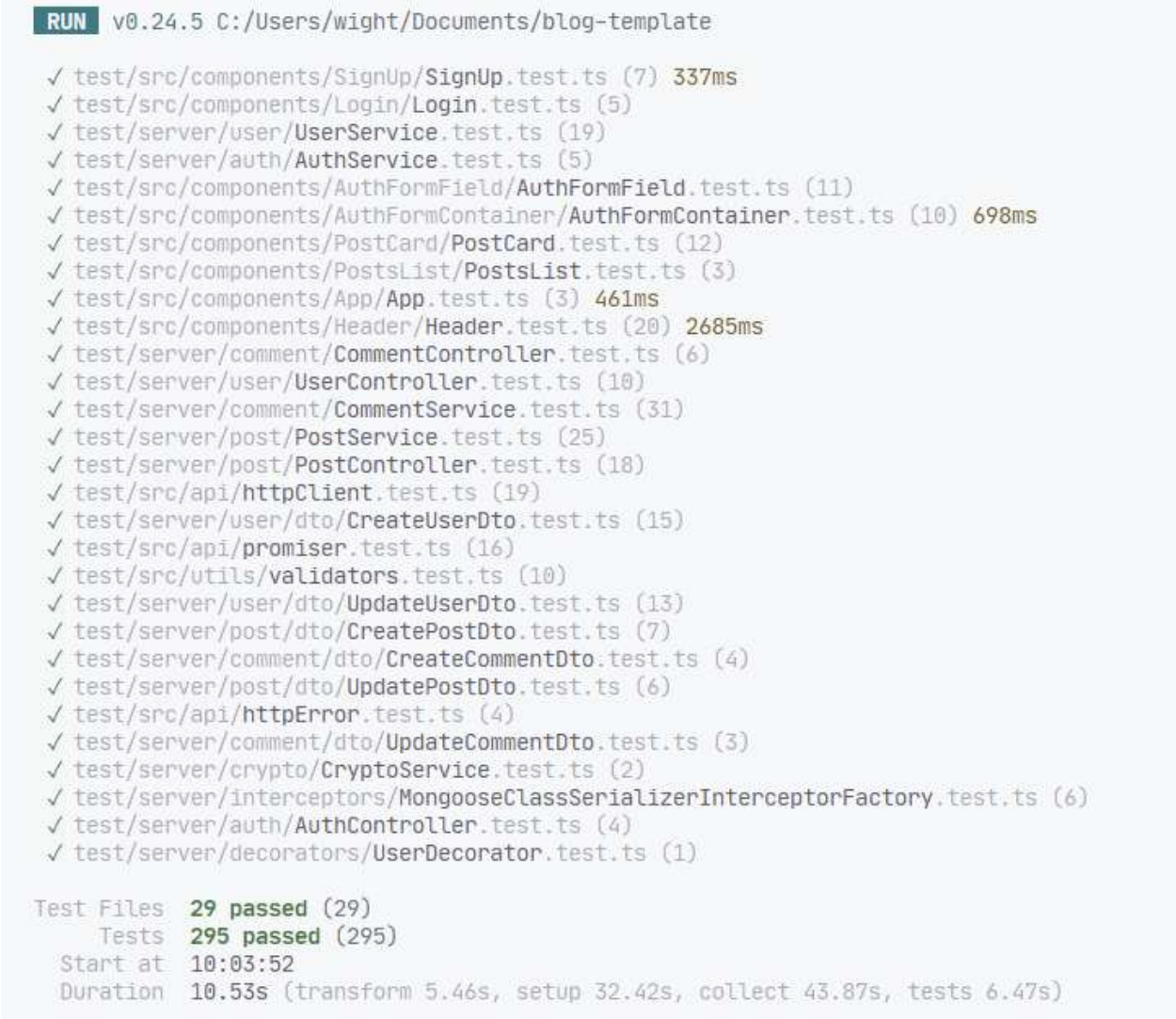


Рисунок 3.26 – Загальний вигляд інтерфейсу тестування

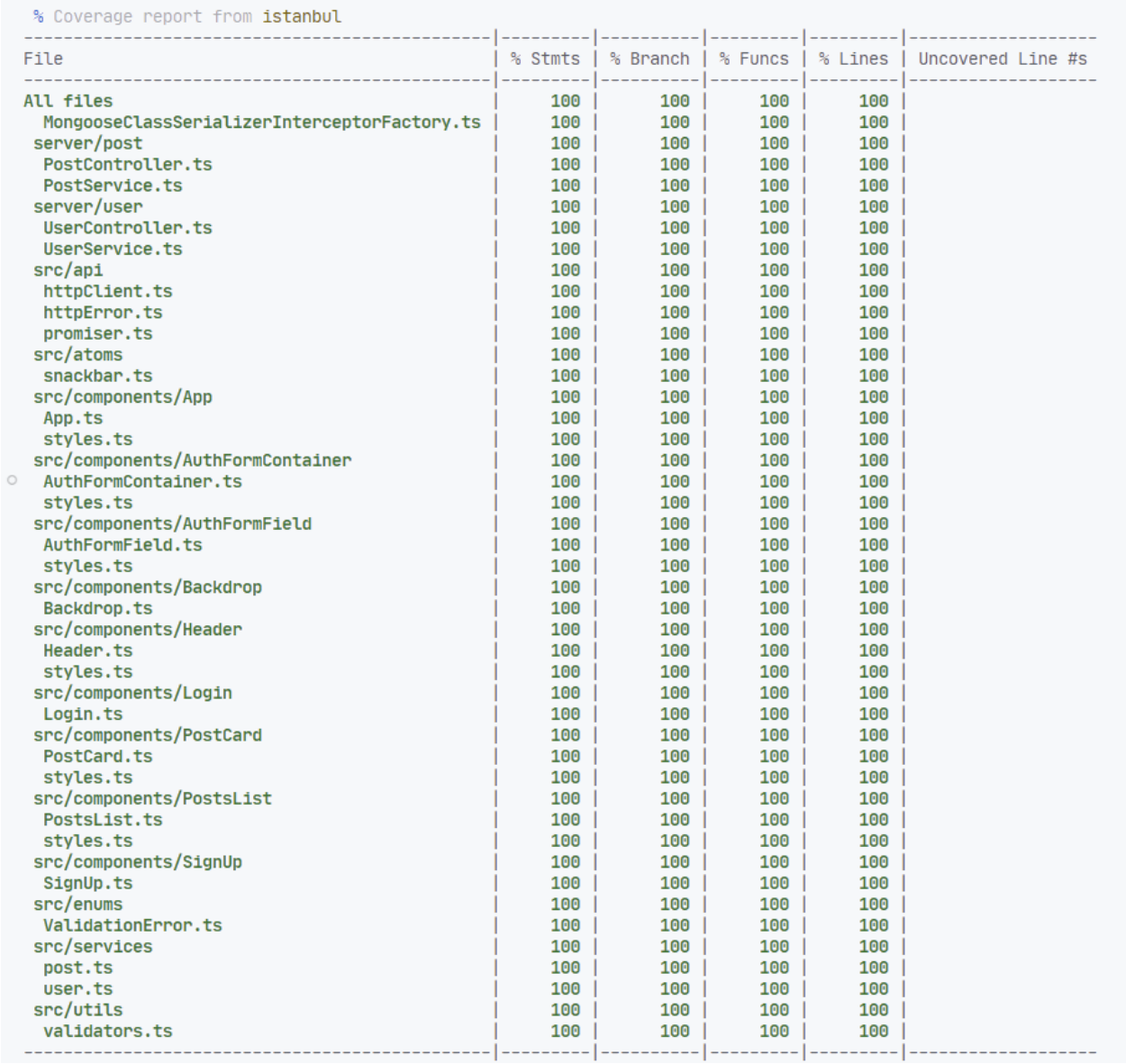
Для прикладу можна навести тест обробки помилки автентифікації:

```
describe('user authentication error', () => {
  beforeEach(() => {
    server.use(
      rest.get('*/api/v1/users', (_req, res, ctx) => {
        return res(
          ctx.status(StatusCodes.INTERNAL_SERVER_ERROR),
          ctx.json({ message: ReasonPhrases.INTERNAL_SERVER_ERROR }),
        );
      }),
    );
  });

  it('should render snackbar with error', async () => {
    render(html`<${App} />`);
```

```
      await waitFor(() => {
        expect(screen.getByText(ReasonPhrases.INTERNAL_SERVER_ERROR)).toBeInTheDo
cument();
      });
    });

    it('should hide alert when close button clicked', async () => {
      render(html`<${App} />`);

      await waitFor(() => {
        expect(screen.getByText(ReasonPhrases.INTERNAL_SERVER_ERROR)).toBeInTheDo
cument();
      });

      const close = screen.getByTestId('CloseIcon');

      fireEvent.click(close);

      await waitFor(() => {
        expect(screen.queryByText(ReasonPhrases.INTERNAL_SERVER_ERROR)).not.toBeI
nTheDocument();
      });
    });
  });
```

Тестування було проведено методом 100% покриття. Інструментом перевірки покриття є Istanbul. Він дозволяє аналізувати покриття тестами як всього проекту так і окремих файлів і виводити результати тестування у різних форматах.

Результат перевірки покриття наведено на рис. 3.27.



```
% Coverage report from istanbul
-----------------------------------------------|---------|----------|---------|---------|-------------------
File                                           | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-----------------------------------------------|---------|----------|---------|---------|-------------------
All files                                      |     100 |      100 |     100 |     100 |
  MongooseClassSerializerInterceptorFactory.ts |     100 |      100 |     100 |     100 |
 server/post                                   |     100 |      100 |     100 |     100 |
  PostController.ts                            |     100 |      100 |     100 |     100 |
  PostService.ts                               |     100 |      100 |     100 |     100 |
 server/user                                   |     100 |      100 |     100 |     100 |
  UserController.ts                            |     100 |      100 |     100 |     100 |
  UserService.ts                               |     100 |      100 |     100 |     100 |
 src/api                                       |     100 |      100 |     100 |     100 |
  httpClient.ts                                |     100 |      100 |     100 |     100 |
  httpError.ts                                 |     100 |      100 |     100 |     100 |
  promiser.ts                                  |     100 |      100 |     100 |     100 |
 src/atoms                                     |     100 |      100 |     100 |     100 |
  snackbar.ts                                  |     100 |      100 |     100 |     100 |
 src/components/App                            |     100 |      100 |     100 |     100 |
  App.ts                                       |     100 |      100 |     100 |     100 |
  styles.ts                                    |     100 |      100 |     100 |     100 |
 src/components/AuthFormContainer              |     100 |      100 |     100 |     100 |
  AuthFormContainer.ts                         |     100 |      100 |     100 |     100 |
  styles.ts                                    |     100 |      100 |     100 |     100 |
 src/components/AuthFormField                  |     100 |      100 |     100 |     100 |
  AuthFormField.ts                             |     100 |      100 |     100 |     100 |
  styles.ts                                    |     100 |      100 |     100 |     100 |
 src/components/Backdrop                       |     100 |      100 |     100 |     100 |
  Backdrop.ts                                  |     100 |      100 |     100 |     100 |
 src/components/Header                         |     100 |      100 |     100 |     100 |
  Header.ts                                    |     100 |      100 |     100 |     100 |
  styles.ts                                    |     100 |      100 |     100 |     100 |
 src/components/Login                          |     100 |      100 |     100 |     100 |
  Login.ts                                     |     100 |      100 |     100 |     100 |
 src/components/PostCard                       |     100 |      100 |     100 |     100 |
  PostCard.ts                                  |     100 |      100 |     100 |     100 |
  styles.ts                                    |     100 |      100 |     100 |     100 |
 src/components/PostsList                      |     100 |      100 |     100 |     100 |
  PostsList.ts                                 |     100 |      100 |     100 |     100 |
  styles.ts                                    |     100 |      100 |     100 |     100 |
 src/components/SignUp                         |     100 |      100 |     100 |     100 |
  SignUp.ts                                    |     100 |      100 |     100 |     100 |
 src/enums                                     |     100 |      100 |     100 |     100 |
  ValidationError.ts                           |     100 |      100 |     100 |     100 |
 src/services                                  |     100 |      100 |     100 |     100 |
  post.ts                                      |     100 |      100 |     100 |     100 |
  user.ts                                      |     100 |      100 |     100 |     100 |
 src/utils                                     |     100 |      100 |     100 |     100 |
  validators.ts                                |     100 |      100 |     100 |     100 |
-----------------------------------------------|---------|----------|---------|---------|-------------------
```

Рисунок 3.27 – Результат перевірки покриття

Додатково було використано функцію деталізації результатів покриття за допомогою HTML. Детальний перевірки покриття конкретного файлу наведено на рис. 3.27.

All files / src/components/App App.ts

100% Statements 9/9 **100%** Branches 7/7 **100%** Functions 3/3 **100%** Lines 9/9

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

```
1    import { Alert, CircularProgress, Grid, Snackbar } from '@mui/material';
2    import { html } from 'htm/preact';
3    import { StatusCodes } from 'http-status-codes';
4    import { useAtom } from 'jotai';
5    import { useEffect } from 'preact/hooks';
6    import { Outlet } from 'react-router-dom';
7
8    import { HttpError } from '#src/api/httpError.js';
9    import { snackbarAtom } from '#src/atoms/snackbar.js';
10   import { Backdrop } from '#src/components/Backdrop/index.js';
11   import { Header } from '#src/components/Header/index.js';
12   import { useUser } from '#src/services/user.js';
13
14   import * as S from './styles.js';
15
16 1x export const App = () => {
17 8x   const { error: getUserError, isInitialLoading: isGetUserLoading } = useUser();
18
19 8x   const [snackbar, setSnackbar] = useAtom(snackbarAtom);
20
21 8x   useEffect(() => {
22 5x     if (getUserError instanceof HttpError && getUserError.code !== StatusCodes.UNAUTHORIZED) {
23 2x       setSnackbar({ open: true, message: getUserError.message, severity: 'error' });
24       }
25     }, [setSnackbar, getUserError]);
26
27 8x   const handleSnackbarClose = () => {
28 1x     setSnackbar({ open: false });
29     };
30
31 8x   return html`
32       <${S.App}>
33         <${Backdrop} open=${isGetUserLoading}>
34           <${CircularProgress} color="inherit" />
35         </>
36         <${Snackbar} open=${snackbar.open} autoHideDuration=${3000} onClose=${handleSnackbarClose}>
37           <${Alert}
38             onClose=${handleSnackbarClose}
39             severity=${snackbar.severity || 'info'}
40             sx=${{ display: !snackbar.open && 'none' }}
41           >
42             ${snackbar.message}
43           </>
44         </>
45         <${Header} />
46         <${Grid} container justifyContent="center" component=${S.MainContent}>
47           <${S.PageWrapper}>
48             <${Outlet} />
49           </>
50         </>
51       </>
52     `;
53   };
54
```

Рисунок 3.27 – Детальний результат перевірки покриття файлу

Ручне тестування серверної частини проводилось шляхом перевірки конкретних маршрутів серверу за допомогою розширення Thunder Client для середовища розробки Visual Studio Code. Воно доволі легке, має простий інтерфейс та дозволяє тестувати Rest API не виходячи з середовища розробки.

Результат тестування маршруту «posts» наведено на рис. 3.28.

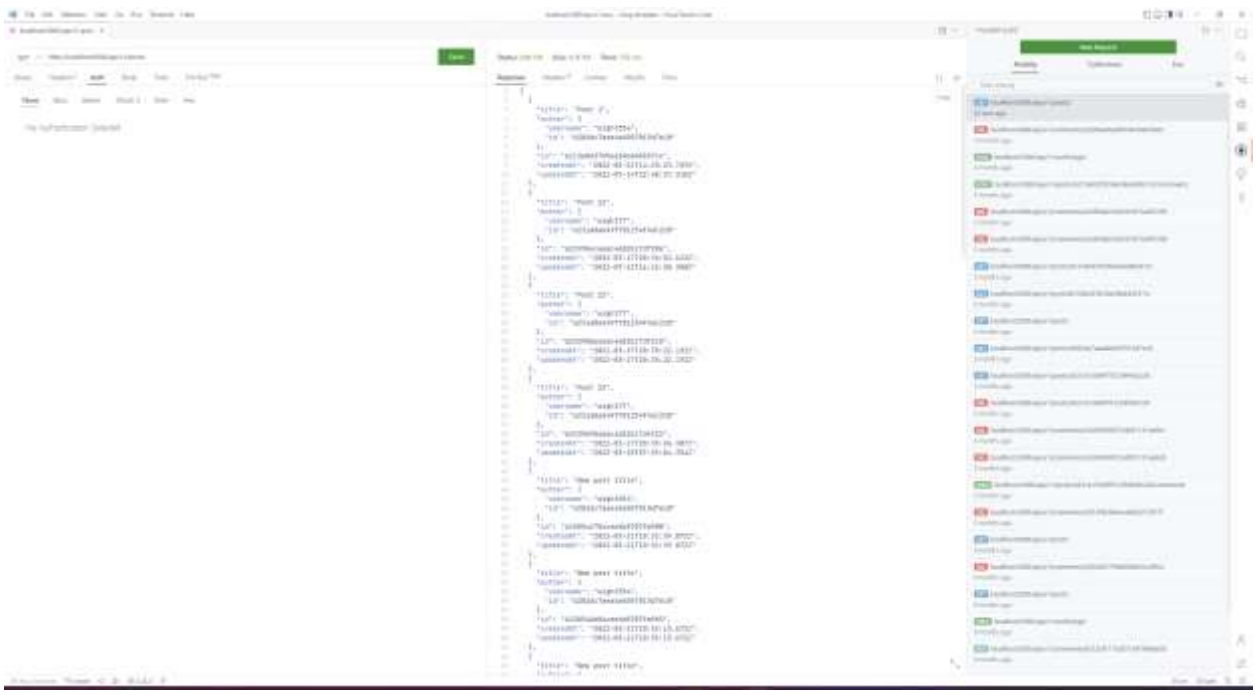


Рисунок 3.28 – Результат тестування маршруту «posts»

Тестування серверної чистини проводилось шляхом навантажувального тестування.

Навантажувальне тестування (load testing) – це проста форма тестування продуктивності. Воно зазвичай проводиться для того, щоб оцінити поведінку програми (додатка) із заданим очікуваним навантаженням. Цим навантаженням може бути, наприклад, кількість користувачів, які будуть одночасно працювати з програмою. Такий вид тестування дозволяє отримати час відгуку всіх найважливіших бізнес-транзакцій [36].

Для тестування використовувалась бібліотека AutoCannon. Це інструмент тестування HTTP / 1.1 з підтримкою конвеєризації HTTP та HTTPS, написаний за допомогою Node.js та має відкритий вихідний код.

Для тестування використовувались наступні набори параметрів:

1. кількість користувачів: 100, кількість запитів в секунду на 1 користувача: 10;
2. кількість користувачів: 100, кількість запитів в секунду на 1 користувача: 100;

3. кількість користувачів: 1000, кількість запитів в секунду на 1 користувача: 1000;

Для візуалізації та діагностики результатів було використано бібліотеку Clinic.js. Це набір інструментів для тестування, написаний за допомогою Node.js та має відкритий вихідний код. Для тестування продуктивності, використовується разом з бібліотекою AutoCannon. Результати тестування показані на рисунках 3.26, 3.27, 3.28.

Рисунок 3.26 – Графіки тестування навантаження (1 набір параметрів)

Рисунок 3.27 – Графіки тестування навантаження (2 набір параметрів)

Рисунок 3.28 – Графіки тестування навантаження (3 набір параметрів)

Користуючись графіками наведеними вище, можна зробити висновок, що система здатна впоратись з необхідним навантаженням, а також, з навантаженням в сотні разів перевищуючим заплановані очікування. Також з графіків видно, що з ростом кількості запитів ростуть затрати на пам’ять, що є цілком нормальним для систем з використанням нереляційних БД, та легко вирішується шляхом горизонтального масштабування [37].

Тестування БД проводилось з використанням інструментарію MongoDB Atlas. Це хмарне рішення розроблене командою MongoDB в якому є можливість створювати, підключатись та керувати БД на базі MongoDB.

Загальний вигляд колекції «posts» наведено на рис. 3.29.

Також за допомгою MongoDB Atlas можна створювати нові документи в колекції, для цього потрібно скористатись функцією «Insert document». Загальний інтерфейс додавання нових документів наведено на рис 3.30.

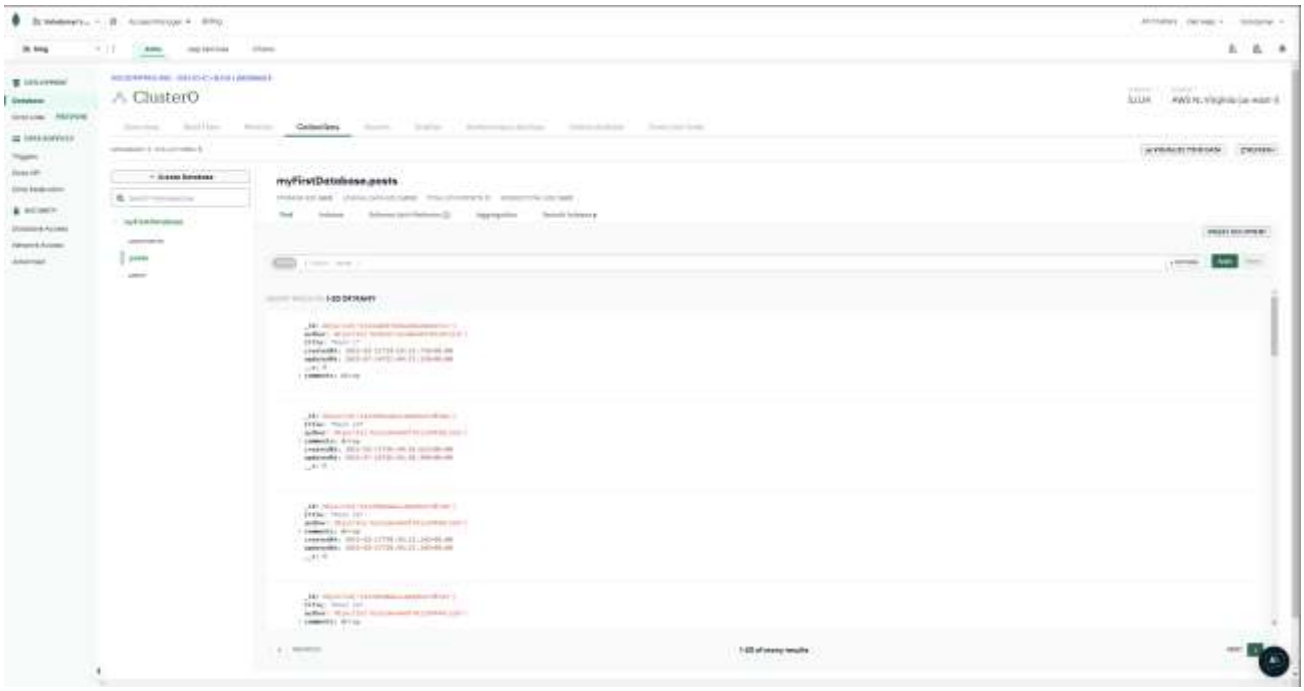


Рисунок 3.29 – Загальний вигляд колекції «posts»

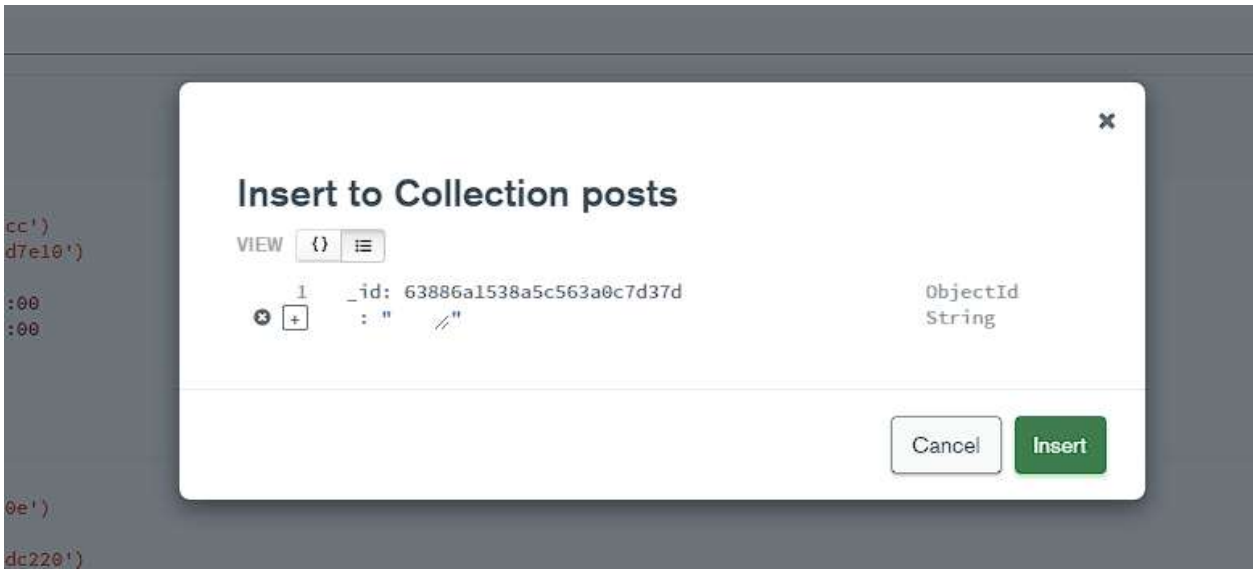


Рисунок 3.30 – Загальний вигляді інтерфейсного вікна «Insert document»

3.4 Висновок до розділу 3

Таким чином, в даному розділі було розглянуто, проведено порівняльний аналіз мов програмування та технологій для програмної реалізації інформаційної технології організації блог-систем, обрано мову

програмування TypeScript, бібліотеки Preact.js, Express, середовище Node.js та систему управління базами даних MongoDB для програмної реалізації. Використовуючи обрані технології, було розроблено програму, що реалізує функції та виконує задачі, описані у попередніх розділах. Проведено тестування розробленої програми за критеріями, що базуються на задачах дослідження та досягненні мети дослідження, а саме розширення функціональних можливостей в області організації блог-систем. Наступним етапом є тестування розробленого програмного продукту на відповідність усім поставленим вимогам і на коректність виконання всіх поставлених задач.

4 ЕКОНОМІЧНА ЧАСТИНА

4.1 Комерційний та технологічний аудит науково-технічної розробки

Метою даного розділу є проведення технологічного аудиту, в даному випадку нового програмного продукту інформаційної технології організації блог-систем. Особливістю програми є те, що дана технологія є блог-системою для створення блогів під ключ, а також а також надає більше функціональних можливостей у порівнянні з конкурентами.

Аналогом може бути Medium за ціною 2000 грн. (50$), Wordpress за ціною 1000 грн. (25$).

Для проведення комерційного та технологічного аудиту залучають не менше 3-х незалежних експертів. Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням п’ятибальної системи оцінювання за 12-ма критеріями, у відповідності із табл. 4.1.

Таблиця 4.1 – Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Бали (за 5-ти бальною шкалою)					
Кри-те-рій	0	1	2	3	4
Технічна здійсненність концепції					

Продовження табл. 4.1

1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздат-ність продукту в реальних умовах
2	Багато аналогів на малому ринку	Ринкові п Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
Ринкові переваги					
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в ана-логів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кра-щі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів

Продовження табл. 4.1

5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практик на здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так із комерційної реалізації ідеї

Продовження табл. 4.1

9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років

Продовження табл. 4.1

12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Усі дані по кожному параметру занесено в таблиці 4.2

Таблиця 4.2 – Результати оцінювання комерційного потенціалу розробки

Критерії оцінювання	ПІБ експертів		
	Експерт 1	Експерт 2	Експерт 3
	Бали		
Технічна здійсненність концепції	3	3	2
Наявність аналогів на ринку	3	3	2
Цінова політика	4	4	4
Технічні та споживчі властивості виробу	3	3	4
Експлуатаційні витрати	3	4	3
Ринок збуту	4	3	4

Продовження табл. 4.2

Конкурентоспроможність	3	4	3
Фахівці з технічної і комерційної реалізації	4	3	4
Фінансування	4	4	3
Матеріально-технічна база	3	3	3
Термін реалізації ідеї	4	3	3
Супровідна документація	3	3	4
Сума	41	40	39
Середньоарифметична сума балів	(41+40+39) / 3 = 40		

За даними таблиці 4.2 можна зробити висновок щодо рівня комерційного потенціалу даної розробки. Для цього доцільно скористатись рекомендаціями, наведеними в таблиці 4.3.

Таблиця 4.3 - Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ , розрахована на основі висновків	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

Як видно з таблиці, рівень комерційного потенціалу розроблюваного нового програмного продукту є вище середнього, що досягається за рахунок того, що програмний продукт відрізняється від існуючих тим, що дана технологія є блог-системою для створення блогів під ключ, а також надає більше функціональних можливостей у порівнянні з конкурентами.

4.2 Прогнозування витрат на виконання науково-дослідної (дослідно-конструкторської) роботи

4.2.1 Основна заробітна плата розробників, яка розраховується за формулою:

$$З_о = \frac{М}{Т_р} \cdot t, \quad (4.1)$$

де М – місячний посадовий оклад конкретного розробника (дослідника), грн.;

$Т_р$ – число робочих днів в місяці, 23 днів;

t – число днів роботи розробника (дослідника).

Результати розрахунків зведемо до таблиці 4.1.

Таблиця 4.1 – Основна заробітна плата розробників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник проекту	21000	913,04	31	28304,348
Програміст	18000	782,61	31	24260,870
Всього				52565,22

Так як в даному випадку розробляється програмний продукт, то розробник виступає одночасно і основним робітником, і тестувальником розроблюваного програмного продукту.

4.2.2 Додаткова заробітна плата розробників, які приймали участь в розробці обладнання.

Додаткова заробітна плата прийнято розраховувати як 11 % від основної заробітної плати розробників та робітників:

$$З_д = З_о \cdot 11\ \% / 100\ \% \quad (4.2)$$

$З_д = (52565,22 \cdot 11\ \% / 100\ \%) = 5782,17$ (грн.)

4.2.3 Нарахування на заробітну плату розробників.

Згідно діючого законодавства нарахування на заробітну плату складають 22 % від суми основної та додаткової заробітної плати.

$$Н_з = (З_о + З_д) \cdot 22\ \% / 100\% \qquad (4.3)$$

$Н_з = (52565{,}22 + 5782{,}17) \cdot 22\ \% / 100\ \% = 12836{,}43$ (грн.)

4.2.4. Оскільки для розроблювального пристрою не потрібно витрачати матеріали та комплектуючи, то витрати на матеріали і комплектуючі дорівнюють нулю.

4.2.5 Амортизація обладнання, яке використовувалось для проведення розробки.

Амортизація обладнання, що використовувалось для розробки в спрощеному вигляді амортизація обладнання, що використовувалась для розробки розраховується за формулою:

$$A = \frac{Ц}{Тв} \cdot \frac{t_{вик}}{12}\ \text{[грн.]}. \qquad (4.4)$$

де Ц – балансова вартість обладнання, грн.;

Т – термін корисного використання обладнання згідно податкового законодавства, років

$t_{вик}$ – термін використання під час розробки, місяців

Розрахуємо, для прикладу, амортизаційні витрати на комп’ютер балансова вартість якого становить 75000 грн., термін його корисного використання згідно податкового законодавства – 2 роки, а термін його фактичного використання – 1,35 міс.

$$A_{обл} = \frac{75000}{2} \times \frac{1{,}35}{12} = 4211{,}96 \; грн.$$

Аналогічно визначаємо амортизаційні витрати на інше обладнання та приміщення. Розрахунки заносимо до таблиці 4.2. Для розрахунку амортизації нематеріальних ресурсів використовується формула:

$$A_{н.р.} = Ц_{н.р.} * Н_а * \frac{t_{вик.}}{12} \qquad (4.5)$$

Але, так як вартість ліцензійної ОС та спеціалізованих ліцензійних нематеріальних ресурсів менше 20000 грн, то даний нематеріальний актив не амортизується, а його вартість включається у вартість розробки повністю, $В_{нем.ак.}$ = 530 грн. (Windows 10 Pro, VSCode).

Таблиця 4.2 – Амортизаційні відрахування матеріальних і нематеріальних ресурсів для розробників

Найменування обладнання	Балансова вартість, грн.	Строк корисного використання, років	Термін використання обладнання, місяців	Амортиза-ційні відрахування-ня, грн.
Комп'ютер та комп'ютерна переферія	75000	2	1,35	4211,957
Офісне обладнення	20000	4	1,35	561,594
Приміщення	750000	20	1,35	4211,957
Всього				8985,51

5.2.6 Тарифи на електроенергію для непобутових споживачів (промислових підприємств) відрізняються від тарифів на електроенергію для населення. При цьому тарифи на розподіл електроенергії у різних постачальників (енергорозподільних компаній), будуть різними. Крім того, розмір тарифу залежить від класу напруги (1-й або 2-й клас). Тарифи на розподіл електроенергії для всіх енергорозподільних компаній встановлює

Національна комісія з регулювання енергетики і комунальних послуг (НКРЕКП). Витрати на силову електроенергію розраховуються за формулою:

$$В_е = В \cdot П \cdot Ф \cdot К_п, \qquad (4.6)$$

де В – вартість 1 кВт-години електроенергії для 1 класу підприємства, В = 6,2 грн./кВт;

П – встановлена потужність обладнання, кВт. П = 0,4 кВт;

Ф – фактична кількість годин роботи обладнання, годин.

$К_п$ – коефіцієнт використання потужності, $К_п$ = 0,9.

$В_е = 0,9 \cdot 0,4 \cdot 8 \cdot 31 \cdot 6,2 = 553,536$ (грн.)

5.2.7 Інші витрати та загальновиробничі витрати.

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками. Витрати за статтею «Інші витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників:

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \qquad (4.7)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати».

$I_в = 52565,22 * 100\% / 100\% = 52565,22$ (грн.)

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків;

витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін. Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників:

$$H_{нзв} = (З_о + З_р) \cdot \frac{H_{нзв}}{100\%}, \quad (4.8)$$

де $H_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати».

$H_{нзв}$ = 52565,22 * 100 % / 100 % = 52565 (грн.)

5.2.9 Витрати на проведення науково-дослідної роботи.

Сума всіх попередніх статей витрат дає загальні витрати на проведення науково-дослідної роботи:

$B_{заг}$ = 52565,22+5782,17+12836,43+8985,51+530+553,54+52565,22+
+52565 = 186383,30 грн.

5.2.11 Розрахунок загальних витрат на науково-дослідну (науково-технічну) роботу та оформлення її результатів.

Загальні витрати на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховуються *ЗВ*, визначається за формулою:

$$ЗВ = \frac{B_{заг}}{\eta} \text{ (грн)}, \quad (5.9)$$

де η – коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи.

Так, якщо науково-технічна розробка знаходиться на стадії: науково-дослідних робіт, то η=0,1; технічного проектування, то η=0,2; розробки конструкторської документації, то η=0,3; розробки технологій, то η=0,4; розробки дослідного зразка, то η=0,5; розробки промислового зразка, то η=0,7; впровадження, то η=0,9. Оберемо η = 0,5, так як розробка, на даний момент, знаходиться на стадії дослідного зразка:

$ЗВ$ = 186383,30 / 0,5 = 372767 грн.

4.3 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

В ринкових умовах узагальнювальним позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку. Саме зростання чистого прибутку забезпечить потенційному інвестору надходження додаткових коштів, дозволить покращити фінансові результати його діяльності, підвищить конкурентоспроможність та може позитивно вплинути на ухвалення рішення щодо комерціалізації цієї розробки.

Для того, щоб розрахувати можливе зростання чистого прибутку у потенційного інвестора від можливого впровадження науково-технічної розробки необхідно:

а) вказати, з якого часу можуть бути впроваджені результати науково-технічної розробки;

б) зазначити, протягом скількох років після впровадження цієї науково-технічної розробки очікуються основні позитивні результати для потенційного інвестора (наприклад, протягом 3-х років після її впровадження);

в) кількісно оцінити величину існуючого та майбутнього попиту на цю або аналогічні чи подібні науково-технічні розробки та назвати основних суб'єктів (зацікавлених осіб) цього попиту;

г) визначити ціну реалізації на ринку науково-технічних розробок з аналогічними чи подібними функціями.

При розрахунку економічної ефективності потрібно обов'язково враховувати зміну вартості грошей у часі, оскільки від вкладення інвестицій до отримання прибутку минає чимало часу. При оцінюванні ефективності інноваційних проектів передбачається розрахунок таких важливих показників:

- абсолютного економічного ефекту (чистого дисконтованого доходу);
- внутрішньої економічної дохідності (внутрішньої норми дохідності);
- терміну окупності (дисконтованого терміну окупності).

Аналізуючи напрямки проведення науково-технічних розробок, розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором можна об'єднати, враховуючи визначені ситуації з відповідними умовами.

4.3.1 Розробка чи суттєве вдосконалення програмного засобу (програмного забезпечення, програмного продукту) для використання масовим споживачем.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

$$\Delta П_i = (\pm \Delta Ц_0 \cdot N + Ц_0 \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \qquad (4.10)$$

де $\pm \Delta Ц_o$ – зміна вартості програмного продукту (зростання чи зниження) від впровадження результатів науково-технічної розробки в аналізовані періоди часу;

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки;

$Ц_о$ – основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки, $Ц_о = Ц_б \pm \Delta Ц_о$;

Цб – вартість програмного продукту у році до впровадження результатів розробки;

ΔN – збільшення кількості споживачів продукту, в аналізовані періоди часу, від покращення його певних характеристик;

λ – коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт $\lambda = 0{,}8333$.

p – коефіцієнт, який враховує рентабельність продукту;

ϑ – ставка податку на прибуток, у 2022 році ϑ =18%.

Припустимо, що при прогнозованій ціні 350 грн. за одиницю виробу, термін збільшення прибутку складе 3 роки. Після завершення розробки і її вдосконалення, можна буде підняти її ціну на *50* грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року – на 25000 шт., протягом другого року – на 20000 шт., протягом третього року на 15000 шт. До моменту впровадження результатів наукової розробки реалізації продукту не було:

$\Delta П_1 = (0*50 + (350 + 50)*25000)* 0{,}8333* 0{,}33) * (1 - 0{,}18) = 1973124{,}921$ *грн.*

$\Delta П_2 = (0*50 + (350 + 50)*(25000+20000)* 0{,}8333* 0{,}33) * (1 - 0{,}18) = 4058999{,}838$ *грн.*

$\Delta П_3 = (0*50 + (350 + 50)*(25000+20000+15000)* 0{,}8333* 0{,}33) * (1 - 0{,}18) = 5411999{,}784$ *грн.*

Отже, комерційний ефект від реалізації результатів розробки за три роки складе 11444124,54 грн.

Розраховуємо приведену вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (5.11)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої науково-дослідної (науково-технічної) роботи, грн;

Т – період часу, протягом якою виявляються результати впровадженої науково-дослідної (науково-технічної) роботи, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, $\tau = 0,05...0,15$;

t – період часу (в роках).

Збільшення прибутку ми отримаємо починаючи з першого року:

ПП = (1973124,921/(1+0,1)[1])+(4058999,838/(1+0,1)[2])+(5411999,784/(1+0,1)[3]) = 1793749,93 + 3354545,32 + 4066115,54 = 9214410,788 грн.

Далі розраховують величину початкових інвестицій *PV*, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки. Для цього можна використати формулу:

$$PV = k_{інв} * ЗВ, \qquad (4.12)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай $k_{інв}=2...5$, але може бути і більшим;

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$PV = 2 * 372767 = 745533{,}18$ *грн.*

Тоді абсолютний економічний ефект $E_{абс}$ або чистий приведений дохід *(NPV, Net Present Value)* для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV, \tag{4.13}$$

$E_{абс} = 9214410{,}788 - 745533{,}18 = 8468877{,}61$ грн.

Оскільки $E_{абс} > 0$ то вкладання коштів на виконання та впровадження результатів даної науково-дослідної (науково-технічної) роботи може бути доцільним.

Для остаточного прийняття рішення з цього питання необхідно розрахувати внутрішню економічну дохідність або показник внутрішньої но-рми дохідності *(IRR, Internal Rate of Return)* вкладених інвестицій та порівняти її з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку вкладати буде економічно недоцільно.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього використаємо формулу:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{4.14}$$

$T_ж$ – життєвий цикл наукової розробки, роки.

$$E_в = \sqrt[3]{(1 + 8468877{,}61/745533{,}18} - 1 = 1{,}312$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f, \tag{4.15}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2022 році в Україні d = (0,09…0,14);

f – показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,5).

$$\tau_{\min} = 0{,}14 + 0{,}05 = 0{,}19.$$

Так як $E_в > \tau_{min}$, то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{ок} = \frac{1}{E_в}, \tag{4.16}$$

$T_{ок}$ = 1 / 1,312 = 0,76 р.

Оскільки $T_{ок}$ < 3-х років, а саме термін окупності рівний 0,76 роки, то фінансування даної наукової розробки є доцільним.

4.4 Висновок до розділу 4

Економічна частина даної роботи містить розрахунок витрат на розробку нового програмного продукту, сума яких складає 372767 гривень. Було спрогнозовано орієнтовану величину витрат по кожній з статей витрат. Також розраховано чистий прибуток, який може отримати виробник від

реалізації нового технічного рішення, розраховано період окупності витрат для інвестора та економічний ефект при використанні даної розробки. В результаті аналізу розрахунків можна зробити висновок, що розроблений програмний продукт за ціною дешевший за аналог і є висококонкурентоспроможним. Період окупності складе близько 0,76 роки.

ВИСНОВКИ

При виконанні магістерської кваліфікаційної роботи розв’язано задачу розробки інформаційної технології та програмного забезпечення організації блог-систем.

1. Проаналізовано відомі рішення та аналоги інформаційної технології організації блог-систем.
2. Обґрунтовано вибір методу розв’язання задачі.
3. Спроектовано інформаційну технологію організації блог-систем.
4. Програмно реалізовано відповідну інформаційну технологію.
5. Проведено тестування розробленого програмного забезпечення та проведено аналіз результатів тестування.

Мета дослідження досягнута за рахунок розробки користувацьких веб-застосунків користуючись запропонованим шаблоном, що дозволило вирішити проблему гнучкості та монетизації існуючих аналогів розробленої інформаційної технології, без втрат у швидкості розробки. В ході розробки інформаційної технології було оптимізовано час завантаження веб-додатку порівняно з відомими аналогами з 660мс до 440мс (на 30%).

Під час програмної реалізації інформаційної технології організації блог-систем було обгрунтовано мову програмування та технології для розробки. Для реалізації блогу було обрано мову програмування TypeScript, так як она є найбільш підходящою для розробки веб-застосунків. Розроблено та описано схему алгоритму роботи програми. Наведено тестові приклади та аналіз результатів роботи серверу, який показав його здатність витримувати навантаження та перспективи масштабування.

Розроблену інформаційну технологію можливо покращити шляхом розробки спрощеної CMS-системи, що дозволить спростити процес розробки клієнтської частин

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Блог-платформа – [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Блог-платформа

2. Свідоцтво про реєстрацію авторського права на твір № 106104. Комп'ютерна програма «Програмний модуль організації блог-систем» / Жданов В.В., Ярова О.А. Дата реєстрації Державним підприємством «Український інститут інтелектуальної власності» 12.07.2021.

3. Роль та популярність блогів – [Електронний ресурс] – Режим доступу: https://sites.google.com/site/infosite44e/blog/rol-ta-popularnist-blogiv

4. Блог – [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Блог

5. Blogs, bloggers, blogging: the role and the power of blogs – [Електронний ресурс] – Режим доступу: http://www.brandba.se/blog/2016/8/11/blogs-bloggers-blogging-the-role-and-the-power-of-blogs

6. Brad Williams, David Damstra, Hal Stern. Professional WordPress: Design and Development, 3rd Edition.

7. Top 8 most popular cms platforms for blogging in 2021 – [Електронний ресурс] – Режим доступу: https://cms2cms.com/blog/top-8-popular-cms-platforms-blogging/

8. Medium – Where good ideas find you – [Електронний ресурс]. – Режим доступу: https://medium.com/

9. Tumblr – [Електронний ресурс]. – Режим доступу: https://www.tumblr.com/

10. Створення блогу – [Електронний ресурс] – Режим доступу: https://katerinakry.blogspot.com/2020/12/blog-post.html

11. How to Choose Development Platform for Your Project? – [Електронний ресурс]. – Режим доступу: https://cadabra.studio/blog/how-to-choose-development-platform-for-your-project

12. Deliver Better Digital Experiences with Composable Architectures – [Електронний ресурс] – Режим доступу: https://hygraph.com/blog/better-digital-experiences-with-composable-architectures

13. What is the Difference Between SPAs, SSGs, and SSR? – [Електронний ресурс] – Режим доступу: https://hygraph.com/blog/difference-spa-ssg-ssr

14. Microservices vs. monolithic architecture – [Електронний ресурс] – Режим доступу: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith

15. Швець О. Занурення в Патерни Проектування.

16. Паттерны проектирования в JavaScript – [Електронний ресурс]. – Режим доступу: https://habr.com/ru/company/ruvds/blog/427293/

17. Passport.js – [Електронний ресурс] – Режим доступу: http://www.passportjs.org/

18. Проектирование программного обеспечения – [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/74330/

19. Васильев А. Н. Java. Объектно-ориентированное программирование: учебное пособие : базовый курс / Васильев А. Н.; . - Санкт-Петербург [и др.]: Питер, 2014. - 396 с.: ил. - ISBN 978-5-496-00044-4;

20. Д. Фленаган. JavaScript. Подробное руководство.

21. Understanding client-side JavaScript frameworks – [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks

22. Svelte • Cybernetically enhanced web apps – [Електронний ресурс] – Режим доступу: https://svelte.dev/

23. Mark Volkmann. Svelte and Sapper in Action.

24. Vue.js – [Електронний ресурс] – Режим доступу: https://vuejs.org/

25. The Good and the Bad of Vue.js Framework Programming – [Електронний ресурс]. – Режим доступу: https://www.altexsoft.com/blog/engineering/pros-and-cons-of-vue-js/

26. React – A JavaScript library for building user interfaces – [Електронний ресурс]. – Режим доступу: https://reactjs.org/

27. What is ReactJS: Introduction To React and Its Features – [Електронний ресурс] – Режим доступу: https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs

28. Node.js – [Електронний ресурс]. – Режим доступу: https://nodejs.org/

29. Express - Node.js web application framework Features – [Електронний ресурс] – Режим доступу: https://expressjs.com/

30. Is Express the Best Option? – [Електронний ресурс]. – Режим доступу: https://medium.com/@mklum88/is-express-the-best-option-c5990ac9232e

31. Koa - next generation web framework for node.js – [Електронний ресурс] – Режим доступу: https://koajs.com/

32. MongoDB – [Електронний ресурс] – Режим доступу: https://www.mongodb.com

33. PostgreSQL: The world's most advanced open source database – [Електронний ресурс] – Режим доступу: https://www.postgresql.org/

34. Amazon DynamoDB | NoSQL Key-Value Database | Amazon Web Services – [Електронний ресурс] – Режим доступу: https://aws.amazon.com/dynamodb/

35. What Is Unit Testing? – [Електронний ресурс] – Режим доступу: https://smartbear.com/learn/automated-testing/what-is-unit-testing/

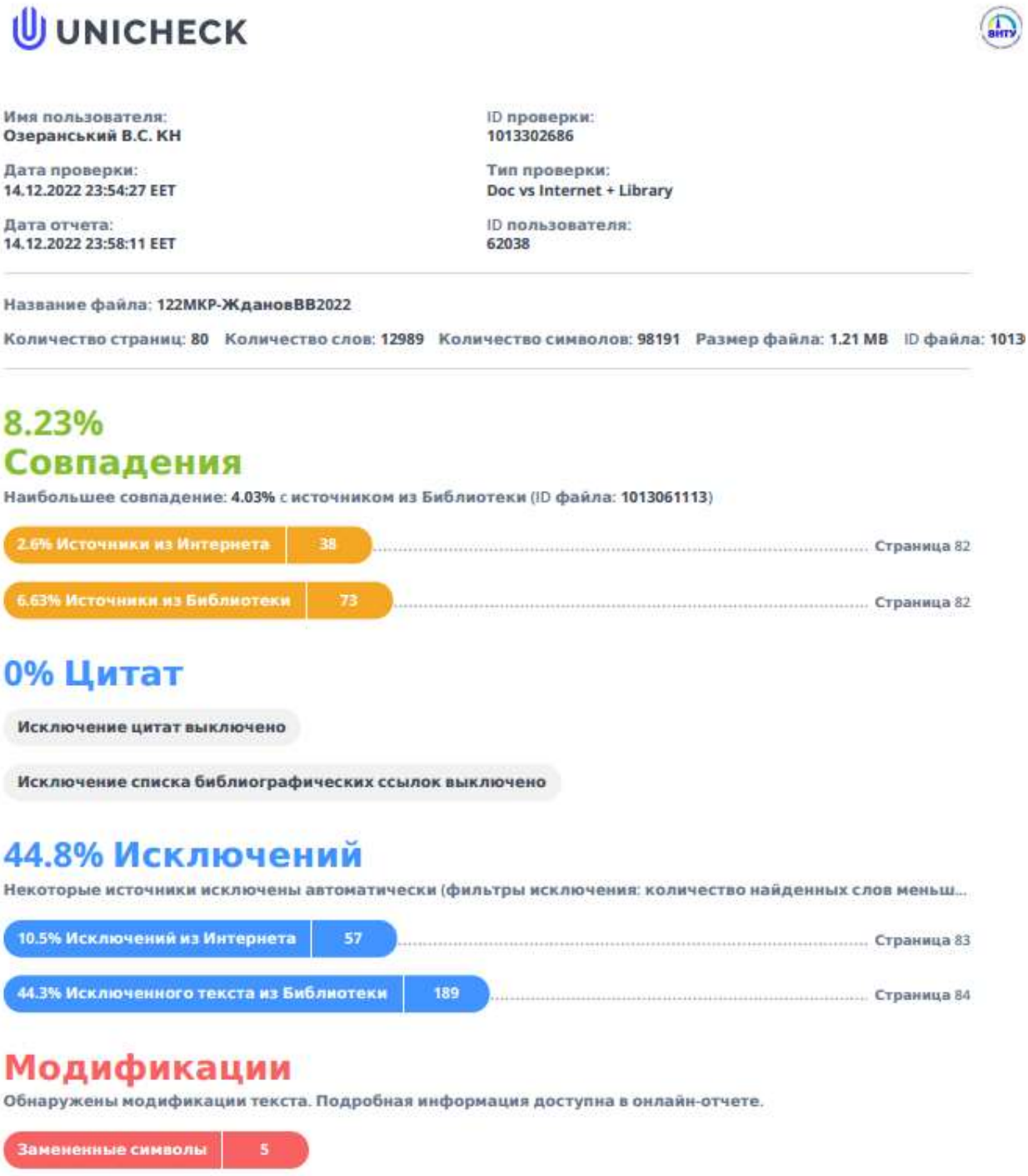
36. Тестування продуктивності – [Електронний ресурс] – Режим доступу: https://training.qatestlab.com/blog/technical-articles/performance-testing/

37. SQL, NoSQL, and Scale: How DynamoDB scales where relational databases don't – [Електронний ресурс] – Режим доступу: https://www.alexdebrie.com/posts/dynamodb-no-bad-queries/

ДОДАТКИ

Додаток А

Протокол перевірки МКР на наявність текстових запозичень



UNICHECK

Имя пользователя:
Озеранський В.С. КН

ID проверки:
1013302686

Дата проверки:
14.12.2022 23:54:27 EET

Тип проверки:
Doc vs Internet + Library

Дата отчета:
14.12.2022 23:58:11 EET

ID пользователя:
62038

Название файла: 122МКР-ЖдановВВ2022

Количество страниц: 80 Количество слов: 12989 Количество символов: 98191 Размер файла: 1.21 MB ID файла: 1013

8.23%
Совпадения

Наибольшее совпадение: 4.03% с источником из Библиотеки (ID файла: 1013061113)

2.6% Источники из Интернета 38 ..... Страница 82

6.63% Источники из Библиотеки 73 ..... Страница 82

0% Цитат

Исключение цитат выключено

Исключение списка библиографических ссылок выключено

44.8% Исключений

Некоторые источники исключены автоматически (фильтры исключения: количество найденных слов меньш...

10.5% Исключений из Интернета 57 ..... Страница 83

44.3% Исключенного текста из Библиотеки 189 ..... Страница 84

Модификации

Обнаружены модификации текста. Подробная информация доступна в онлайн-отчете.

Замененные символы 5

Додаток Б

Лістинг програми

```
import 'preact/debug';

import { createTheme, CssBaseline, ThemeProvider } from '@mui/material';
import { QueryClientProvider } from '@tanstack/react-query';
import { ReactQueryDevtools } from '@tanstack/react-query-devtools';
import { html } from 'htm/react';
import { render } from 'preact';
import { createBrowserRouter, RouterProvider } from 'react-router-dom';

import { queryClient } from '#src/api/queryClient.js';
import { Index, loader as indexLoader } from '#src/routes/index.js';
import { loader as loginLoader, Login } from '#src/routes/login.js';
import { loader as rootLoader, Root } from '#src/routes/root.js';
import { loader as signUpLoader, SignUp } from '#src/routes/sign-up.js';

const theme = createTheme({
  palette: {
    primary: {
      main: '#673ab8',
    },
    secondary: {
      main: '#8BB83A',
    },
    background: {
      default: '#242424',
    },
  },
  components: {
    MuiCssBaseline: {
      styleOverrides: `
        #app,
        body,
        html {
          height: 100%;
          width: 100%;
```

```
          padding: 0;
          margin: 0;
        }

        #app {
          display: flex;
          flex-direction: column;
        }
    `,
    },
  },
});

const router = createBrowserRouter([
  {
    path: '/',
    element: html`<${Root} />`,
    loader: rootLoader(queryClient),
    children: [
      {
        index: true,
        element: html`<${Index} />`,
        loader: indexLoader(queryClient),
      },
      {
        path: 'login',
        element: html`<${Login} />`,
        loader: loginLoader(queryClient),
      },
      {
        path: 'sign-up',
        element: html`<${SignUp} />`,
        loader: signUpLoader(queryClient),
      },
    ],
  },
]);

render(
```

```
        html`
          <${QueryClientProvider} client=${queryClient}>
            <${ThemeProvider} theme=${theme}>
              <${CssBaseline} />
              <${ReactQueryDevtools} position="bottom-right" />
              <${RouterProvider} router=${router} />
            <//>
          <//>
        `,
        // eslint-disable-next-line @typescript-eslint/no-non-null-assertion
        document.getElementById('app')!,
      );
      import 'reflect-metadata';

      import fastifyCookie from '@fastify/cookie';
      import fastifyStatic from '@fastify/static';
      import { ValidationPipe } from '@nestjs/common';
      import { NestFactory } from '@nestjs/core';
      import { FastifyAdapter, NestFastifyApplication } from '@nestjs/platform-
fastify';
      import { fileURLToPath } from 'url';

      import { AppModule } from '#server/app/AppModule.js';
      import { prettyPrintAddress } from '#server/utils/prettyPrintAddress.js';

      const { PORT, HOST } = process.env;
      const port = PORT || 3000;
      const host = HOST || '0.0.0.0';

      const app = await NestFactory.create<NestFastifyApplication>(AppModule, new
FastifyAdapter());

      app.register(fastifyCookie);

      app.useGlobalPipes(new ValidationPipe({ whitelist: true,
forbidNonWhitelisted: true }));

      if (process.env.NODE_ENV !== 'production') {
        import('#server/getViteServer.js').then(async ({ getViteServer }) => {
```

```
    const vite = await getViteServer();

    app.use(/^(?!\/api\/.*)/, vite.middlewares);
  });
} else {
  app.register(fastifyStatic, {
    root: fileURLToPath(new URL('../public', import.meta.url)),
  });
}

await app.listen(port, host, (_, address) => {
  prettyPrintAddress(address);
});
```

Додаток В

ІЛЮСТРАТИВНА ЧАСТИНА

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ОРГАНІЗАЦІЇ БЛОГ-СИСТЕМ

Виконав: студент 1-го курсу,
групи 1КН-21м
спеціальності 122 «Комп'ютерні науки»
(шифр і назва напряму підготовки, спеціальності)
Жданов В.В.
(прізвище та ініціали)

Керівник: д.т.н., професор каф. КН
Яровий А.А.
(прізвище та ініціали)
«15» 12 2022 р.

Вінниця ВНТУ - 2022 рік

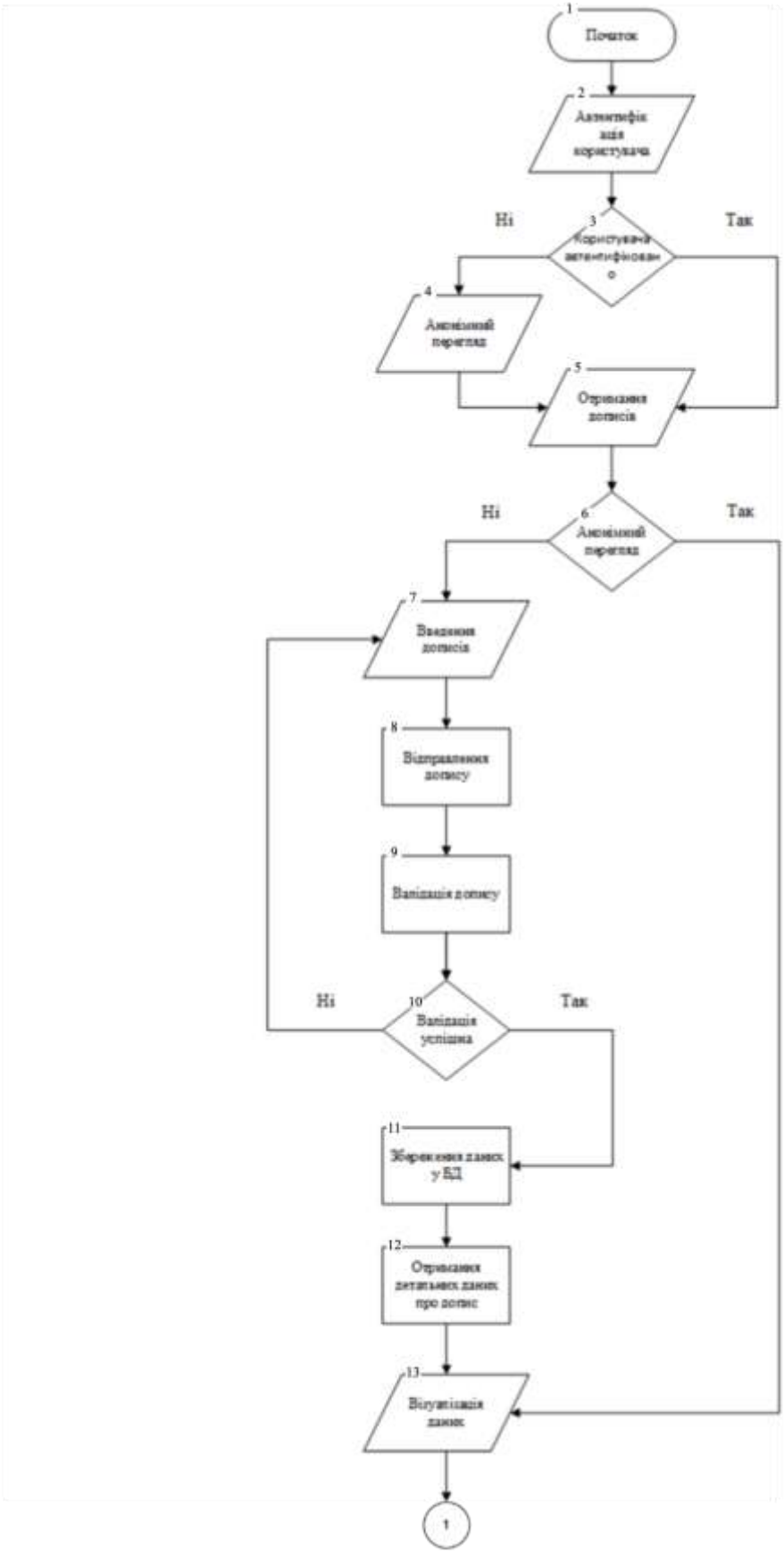


Рисунок В.1 – Алгоритм інформаційної технології роботи інформаційної технології організації блог-систем

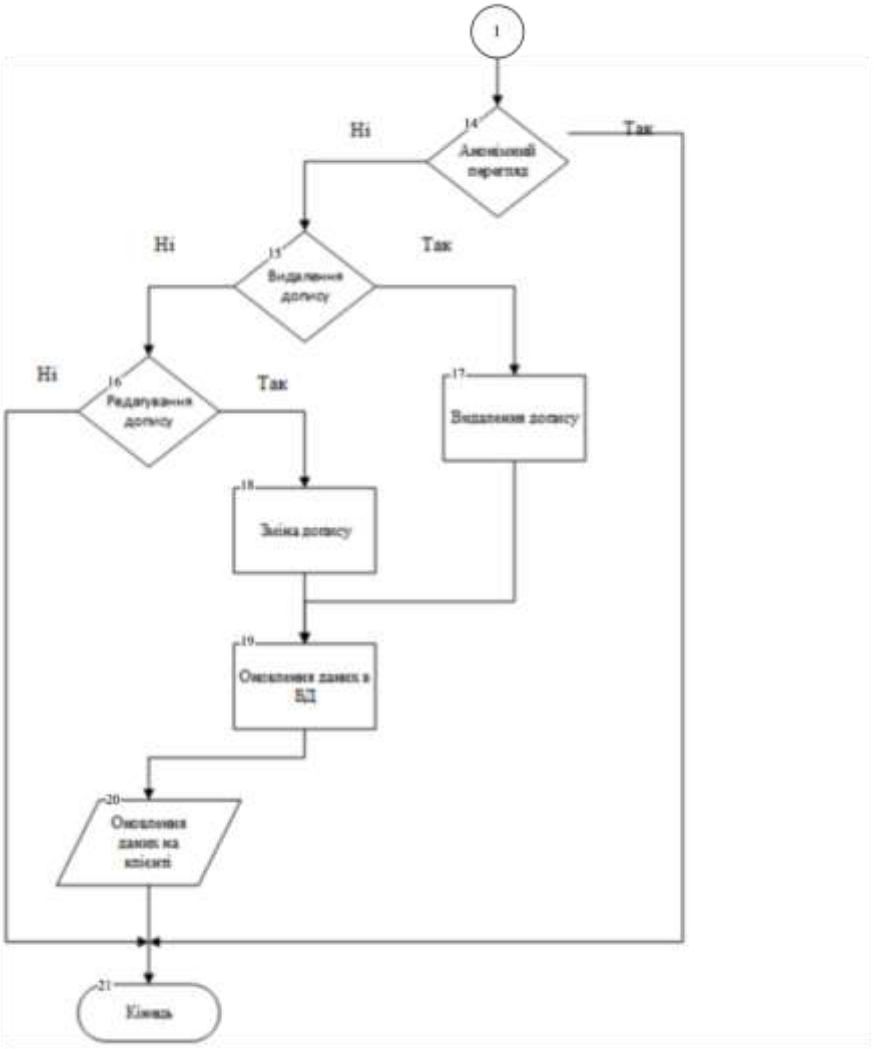


Рисунок В.1, аркуш 2

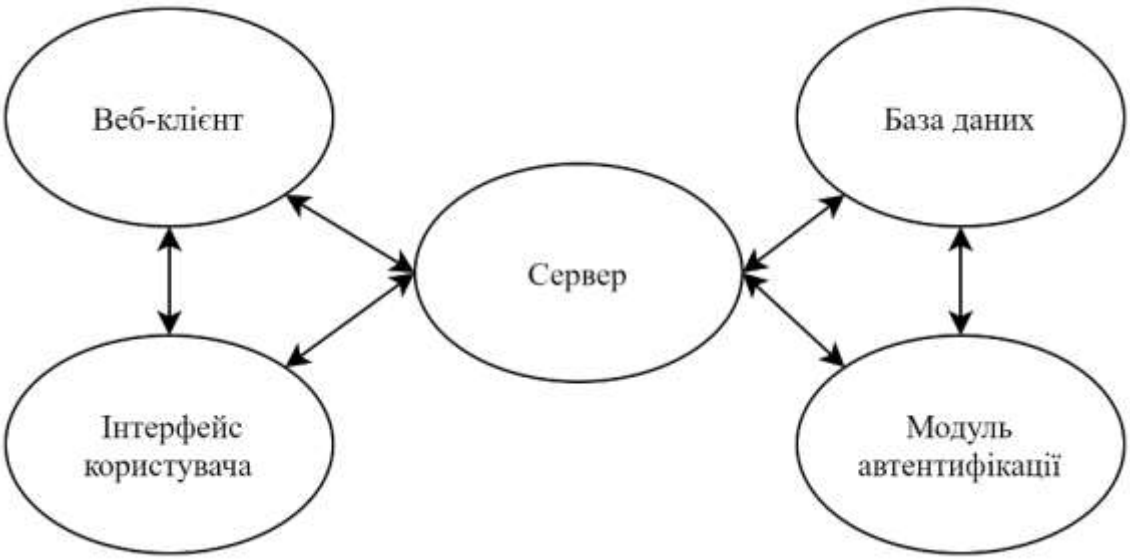


Рисунок В.2 – Загальна структурна схема інформаційної технології організації блог-систем

Рисунок В.3 – Структурна схема модуля автентифікації інформаційної технології організації блог-систем

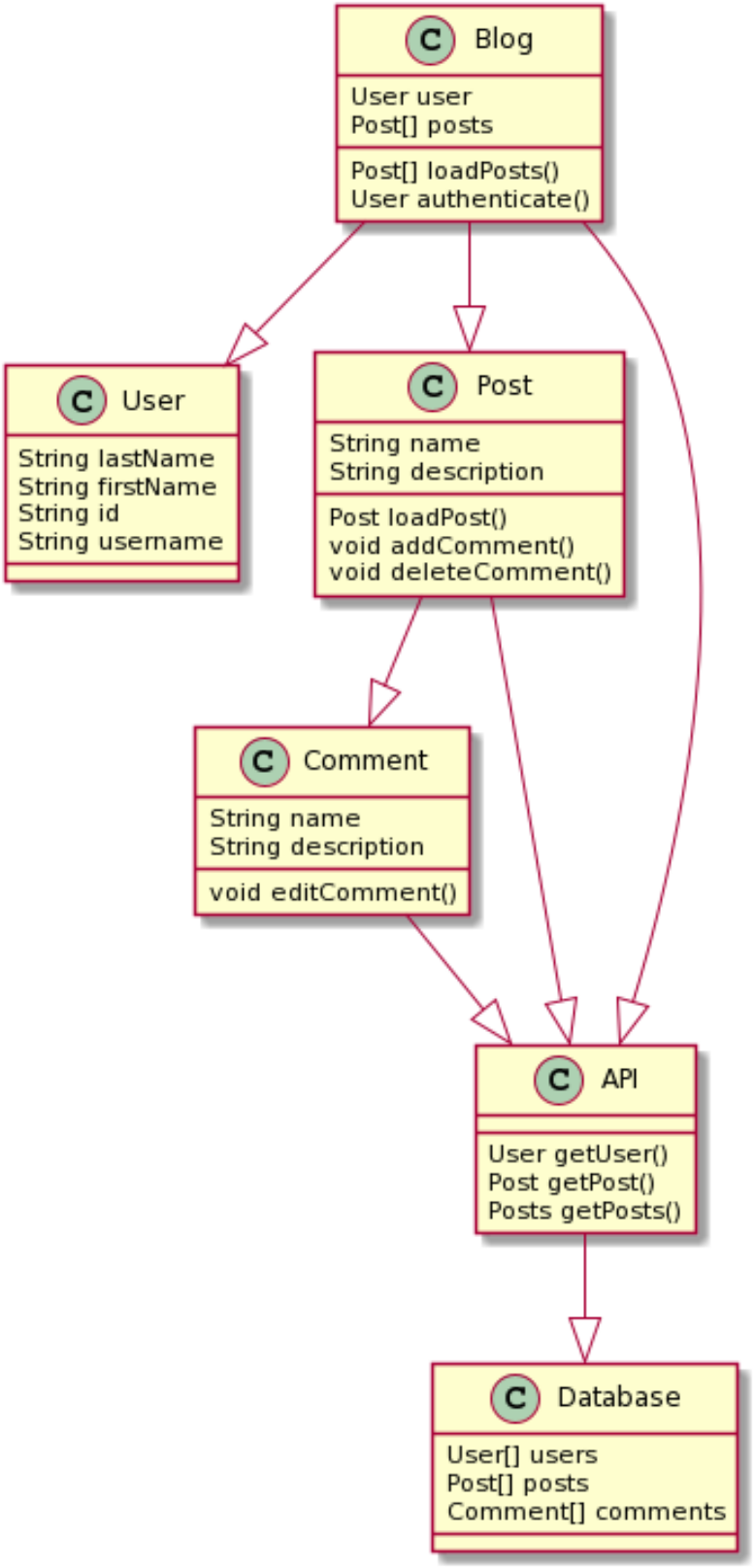


Рисунок В.4 – Діаграма класів інформаційної технології організації блог-систем

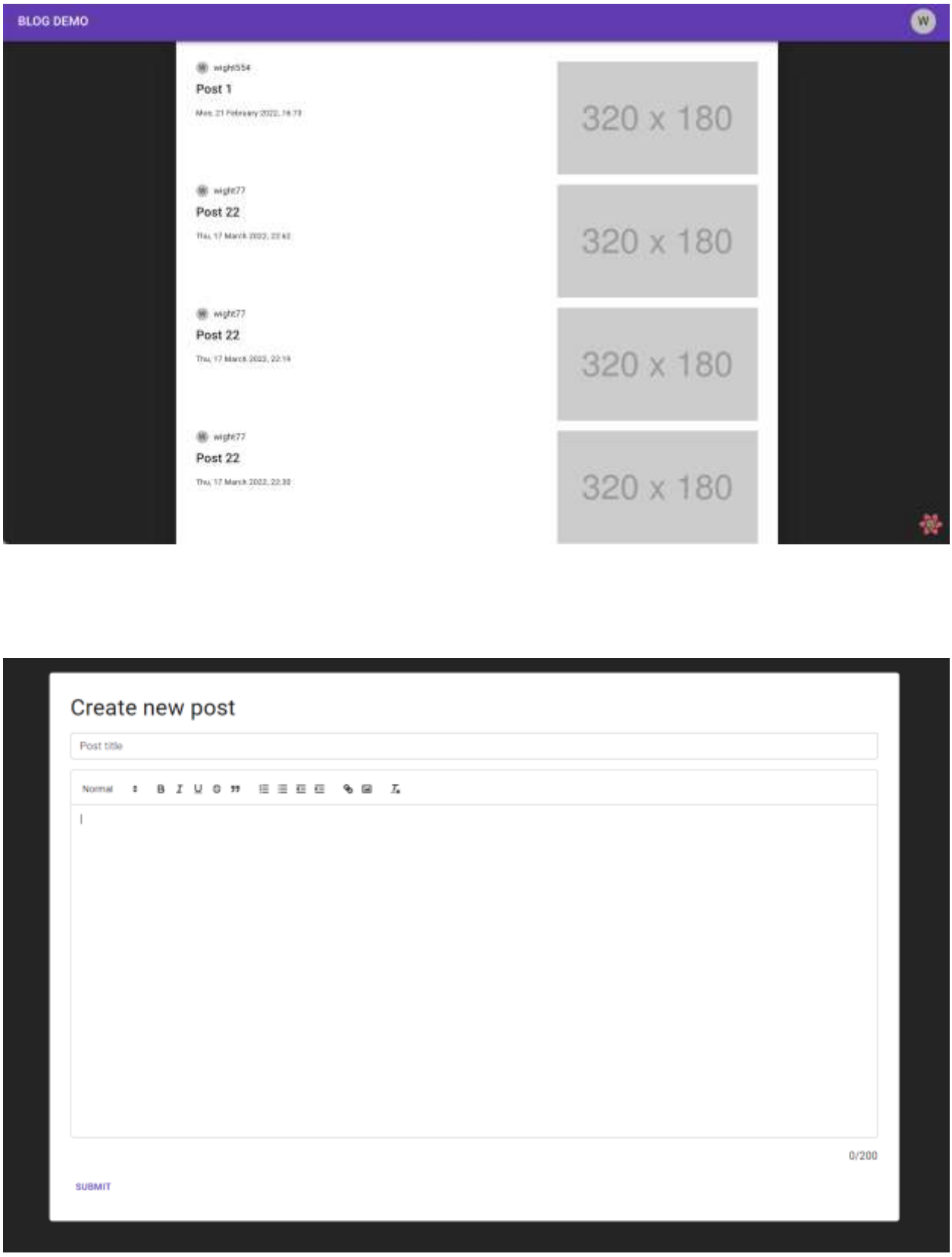


Рисунок В.5 – Робочі вікна інформаційної технології організації блог-систем

Рисунок В.6 – Результати тестування навантаження інформаційної технології організації блог-систем