Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Аналіз та впровадження мікросервісної архітектури для створення масштабованого блогу «Цифровий світ»

Виконав: студент 6 курсу, групи СПм-61
спеціальності 121 «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

_____ Москалик В.І
(підпис) (прізвище та ініціали)

Керівник _____ Стоянов Ю.М.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Стоянов Ю.М.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Петрик М.Р.
(підпис) (прізвище та ініціали)

Рецензент _____ Марценко С.В.
(підпис) (прізвище та ініціали)

Тернопіль 2024

АНОТАЦІЯ

Дана кваліфікаційна робота призначена для здобуття ступеня магістра студента кафедри програмної інженерії Тернопільського національного технічного університету імені Івана Пулюя.

Тема: Аналіз та впровадження мікросервісної архітектури для створення масштабованого блогу «Цифровий світ».

Робота містить: 46 рисунків та 7 таблиць.

Метою даної кваліфікаційної роботи є дослідження мікросервісної архітектури для створення масштабованого блогу «Цифровий світ». У межах роботи будуть розглянуті основні аспекти проєктування та реалізації системи на базі мікросервісів, а також проведено аналіз їхніх переваг, недоліків і можливих викликів. Окрему увагу приділено використанню фреймворку NestJS для впровадження мікросервісного підходу.

Розроблено мікросервіси для серверної частини блогу за допомогою фреймворку NestJS та клієнтську частину за компонентним підходом бібліотеки React.

Перелік ключових слів: СЕРВЕР, КЛІЄНТ, МІКРОСЕРВІС, ДІАГРАМА ПОСЛІДОВНОСТІ, ДІАГРАМА КЛАСІВ, ДІАГРАМА ВИКОРИСТАННЯ.

ANNOTATION

This qualification work is intended for obtaining a Master's degree by a student of the Software Engineering at Ternopil Ivan Puluj National Technical University.

The topic: Analysis and Implementation of Microservices Architecture for Creating a Scalable Blog "Digital World".

The work includes: 46 figures and 7 tables.

The main purpose of this qualification work is to explore microservices architecture for creating a scalable blog, "Digital World". The work examines the fundamental aspects of designing and implementing a system based on microservices and analyzes their advantages, disadvantages, and potential challenges. Special attention is given to the use of the NestJS framework for implementing the microservices approach.

Microservices for the server-side of the blog were developed using the NestJS framework, while the client-side was implemented based on the component-based approach of the React library.

List of keywords: SERVER, CLIENT, MICROSERVICE, SEQUENCE DIAGRAM, CLASS DIAGRAM, USE CASE DIAGRAM.

ЗМІСТ

ВСТУП

У сучасному цифровому середовищі розробка веб-застосунків стала складним завданням, бо вимагає надійності та масштабованості системи. Використання монолітної архітектури стає менш ефективним, бо постійно зростають вимоги до якості обслуговування користувачів, швидкості обробки даних та забезпеченні цілодобового функціонування системи. Високі вимоги заставляють шукати альтернативи. Одна з най популярніших – мікросервісна архітектура. Вона розділяє систему на автономні частини, кожна з яких відповідає за окремі частини бізнес логіки.

Метою даної кваліфікаційної роботи є аналіз мікросерівсної архітектури та її впровадження для побудови масштабованого блогу «Цифровий світ». Буде розглянуто ключові моменти проєктування та розробки системи з використанням мікросервісів, а також аналіз переваг, недоліків та викликів. Також, буде детально розглянуто використання фреймворку NestJS для реалізації мікросервісної архітектури.

"Цифровий світ" це веб-застосунок у вигляді блогу, що складається з кількох мікросервісів, таких як "Авторизація та управління користувачами", "Пости", "Коментарі", "Пошук та фільтрація" і "Аналітика". Метою є створення системи, здатної легко масштабуватись, інтегруватись та ефективно функціонувати в умовах динамічних змін і зростаючих навантажень.

Результати роботи можуть бути корисними для розробників, що хочуть використовувати мікросервіси для створення гнучких та масштабованих систем.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність проблеми

Під час масової цифровізації, створення масштабованих програм це важливе завдання. Оскільки, зростає попит на платформи, які можуть обробляти велику кількість запитів, забезпечувати надійну взаємодію користувачів та швидко адаптуватися до змін. Отже, використання мікросервісів для створення блогу "Цифровий світ" це актуальне завдання.

Даний підхід декомпозиціонує систему на окремі сервіси. Це спрощує процес розростання блогу та зменшує ризики завдяки локалізації помилок, на відміну від монолітних системами, які мають обмеження у гнучкості [1].

1.2 Аналіз існуючих архітектурних рішень

Для проєктування блогу «Цифровий світ» існує два основних варіанти архітектури – монолітна та мікросервісна. Дані підходи мають свої переваги та недоліки, які потрібно взяти до уваги при визначенні найкращого варіанту.

При монолітному підході весь функціонал додатку реалізовується в одному компоненті.

На рисунку 1.2.1 зображено монолітний підхід.

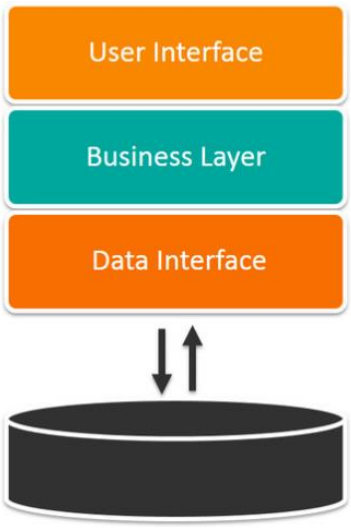

Рисунок 1.2.1 – Монолітна архітектура

Бізнес-логіка, користувацький інтерфейс та робота з базою даних працюють як одна структура, які тісно взаємопов’язана. Оскільки, код тісно пов’язаний, то зміни в будь-якому модулі можуть вплинути на інші модулі. Це ускладнює процес модифікацій, особливо в проєктах, які зростають. Монолітна архітектура є правильним вибором для малих проєктів, або на початкових етапах розробки, оскільки вона дозволяє розробляти та розгортати додаток як один блок, що спрощує управління та контроль. Проте, головним недоліком є труднощі масштабування. Таким чином, монолітні системи обмежені у гнучкості та розвитку і не є придатними для великих і динамічних проєктів [2].

Альтернативний підхід це використання мікросервісів, що передбачає розділення програми на невеликі, незалежні сервіси, які виконують свої завдання.

На рисунку 1.2.2 зображено схему даного підходу.

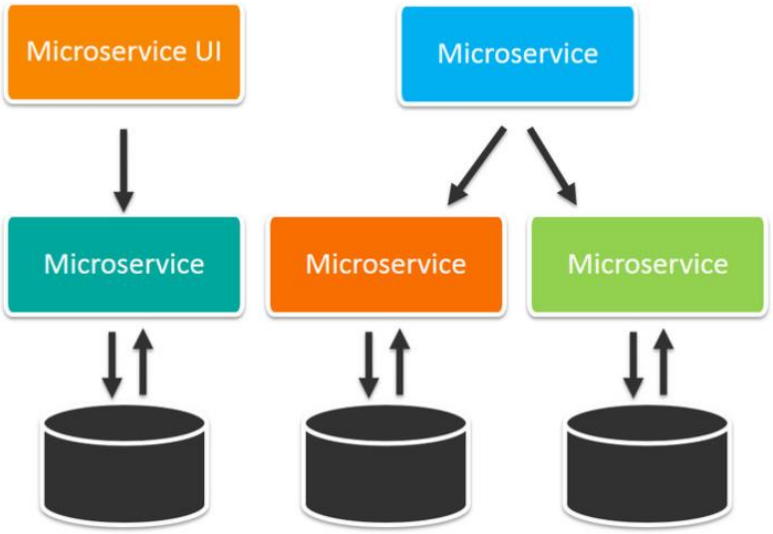

Рисунок 1.2.2 – Мікросервісна архітектура

Всі мікросервіси є окремими автономними компонентами, які мають власні ресурси. Вони взаємодіють між собою за допомогою різних механізмів комунікацій, що забезпечують злагоджену роботу між всіма компонентами системи [3].

Розробники можуть працювати паралельно над різними мікросервісами. Також легко масштабуються: можна масштабувати компоненти, які цього потребують, не чіпаючи цілою системи.

Переваги мікросервісної архітектури [4]:

– Масштабованість;
– Гнучкість;
– Технологічна незалежність;
– Можливість незалежного розгортання;
– Висока стійкість до збоїв.

Недоліки міркосервісної архітектури:

– Складність розробки та управління системою;
– Висока вартість реалізації;
– Складність розгортання, яка потребує досвідчених працівників.

Враховуючи вимоги до масштабованості та гнучкості блогу "Цифровий світ", було прийнято рішення використовувати мікросервісну архітектуру. Хоча цей

підхід є більш складним в реалізації, він забезпечує необхідний рівень масштабованості та гнучкості для майбутнього розвитку блогу.

1.3 Аналіз вимог

В процесі аналізу для блогу «Цифровий світ» було визначено функціональні та нефункціональні вимоги.

Функціональні вимоги блогу:

- Користувачі повинні мати можливість створювати нові пости в блозі.
- Користувачі повинні мати можливість редагувати існуючі пости.
- Система повинна підтримувати форматування тексту (жирний, курсив, заголовки тощо).
- Система повинна дозволяти додавати зображення.
- Користувачі повинні мати можливість видаляти пости.
- Система повинна дозволяти категоризувати пости за допомогою тегів.
- Відвідувачі повинні мати можливість переглядати опубліковані пости.
- Система повинна відображати пости в хронологічному порядку або за популярністю.
- Система повинна дозволяти фільтрувати пости за категоріями.
- Система повинна дозволяти шукати пости за ключовими словами.
- Відвідувачі повинні мати можливість залишати коментарі до постів.
- Користувачі повинні мати можливість відповідати на коментарі.
- Система повинна дозволяти модерувати коментарі (видаляти, схвалювати).
- Адміністратори повинні мати можливість керувати користувачами (створювати, видаляти, блокувати).

- Адміністратори повинні мати можливість змінювати налаштування блогу (назва, опис, тема).
- Система повинна надавати статистику відвідуваності блогу.

Нефункціональні вимоги включають:

- Блог повинен завантажуватися швидко (менше 3 секунд).
- Блог повинен витримувати велику кількість відвідувачів одночасно.
- Блог повинен працювати стабільно без збоїв.
- Дані блогу повинні регулярно створюватися резервні копії.
- Інтерфейс блогу повинен бути інтуїтивно зрозумілим та простим у використанні.
- Блог повинен бути адаптований для різних пристроїв (комп'ютери, планшети, смартфони).

В таблиці 1.3.1 всі вимоги класифіковано за пріоритетом, впливом на користувача та складністю реалізації.

Таблиця 1.3.1 – Класифікація вимог

№	Назва вимоги	Пріоритет	Вплив на користувача	Складність реалізації
1	Публікація та керування постами	Високий	Високий	Висока
2	Перегляд постів	Високий	Високий	Середня
3	Коментарі	Середній	Середній	Висока
4	Адміністрування	Високий	Низький	Низька
5	Продуктивність	Високий	Високий	Високий
6	Безпека	Високий	Високий	Середній
7	Надійність	Високий	Середній	Високий
8	Зручність використання	Середній	Високий	Середній

Одною з вимог, яка не є обов’язковою для реалізації є додавання штучного інтелекту для підрахунку аналітики. У роботах [5] та [6] підкреслюється важливість точності апаратного і програмного забезпечення для систем взаємодії "людина-машина". Ці підходи можуть бути адаптовані для інтеграції ШІ в блог з

метою автоматичного збору і обробки аналітичних даних, наприклад, аналізу поведінки користувачів.

1.4 Вибір інструментів для розробки

Одним з ключових етапів розробки будь-якого програмного забезпечення, особливо при використанні мікросервісної архітектури, є ретельне проєктування кожного сервісу. Це включає в себе не тільки визначення функціональності кожного мікросервісу, але й вибір найбільш підходящих інструментів для його реалізації. Правильний вибір технологій забезпечить ефективність, масштабованість та надійність кожного сервісу, а також спростить розробку та підтримку системи в цілому.

Враховуючи обраний стек технологій (React на клієнті, NestJS на сервері та MongoDB як база даних), для розробки блогу "Цифровий світ" будуть використані наступні інструменти:

NestJS фреймворк для створення кожного мікросервісу. Він побудований на мові TypeScript і забезпечує модульну структуру [7].

MongoDB вибрано, як базу даних через її гнучку схему зберігання даних, яка ідеально підходить для динамічних структур об’єктів, таких як користувачі, пости та коментарі. Вона забезпечує високу продуктивність навіть при великому обсягу даних [8].

Mongoose використовується для спрощення взаємодії з базою даних. Дозволяє визначати схеми та моделі даних, що підвищує надійність коду та зменшує ризик помилок [9].

React вибрано для реалізації клієнтської частини додатку, через його компонентний підхід [10].

Redux використовується для керування станом блогу на клієнтській стороні. Він дає можливість створити централізоване сховище стану, що полегшує управлінням даними між компонентами.

TypeScript основна мова програмування для проєкту [11].

RabbitMQ вибрано брокером повідомлень для забезпечення надійної та асинхронної взаємодії між мікросервісами.

Вищеперераховані інструменти забезпечують реалізацію масштабованої архітектури для блогу «Цифровий світ». Такий підхід гарантує зручність розширення системи в майбутньому.

1.5 Аналіз конкурентів

В українському інфопросторі є багато блогів, які публікують статті про передові технології та новини в світі. Вони виступають прямими конкурентами проєкту «Цифровий світ». Ці блоги залучають велику аудиторію за рахунок високоякісного контенту, своєчасного висвітлення актуальних тем і експертних оглядів [12].

Перша конкурентна платформа Dou.ua – одна з найпопулярніших IT-платформ в Україні. На рисунку 1.4.1 зображено логотип даної платформи.

Рисунок 1.4.1 – Логотип Dou.ua

Dou спеціалізується на публікації: новин в IT-галузі, аналітичних матеріалів, інтерв’ю з експертами, огляд компаній та ринкових трендів. Користувачі даної платформи активно обговорюють кар’єрні можливості та створюють рейтинг IT-

компаній. Їхній контент приваблює початківців та досвідчених спеціалістів. Dou має величезний вплив на українську IT-спільноту.

Другою конкурентною платформою є ITC це платформа спеціалізується на комп'ютерній та цифровій техніці, а також на інших популярний темах. На рисунку 1.4.2 зображено логотип ITC.

Рисунок 1.4.2 – Логотип ITC

Для прикладу користувачу доступні такі категорії: технології, IT-бізнес, крипто, наука та космос, пристрої, софт, military tech, авто, ігри, кіно, а також спецпроєкти в розвитку яких бере участь дана платформа. Платформа надає ґрунтовний контент з оглядом девайсів, породами та аналітикою в IT-бізнесі. Публікування інформації з різних сфер робить її корисної для широкої аудиторії.

Наступним конкурентом є ресурс, що націлений на новини стартапів, підприємств та технологій – AIN. На рисунку 1.4.3 зображено логотип даної платформи.

Рисунок 1.4.3 – Логотип AIN

Даний конкурент висвітлює бізнес-аналітику в IT-сфері, публікує інтерв'ю із засновниками стартапів, надає поради з інвестування, а також надає поради з маркетингу та управління IT-проєктами. Даному сайту надають перевагу

підприємці та інвестори, а також ІТ-фахівці, які цікавляться розвитком власного бізнесу або участю у стартапах.

Останнім конкурентом є сайт Dev, що орієнтується на новинах з ІТ-світу, інноваціях та стартапах. На рисунку 1.4.4 зображено логотип даної платформи.

Рисунок 1.4.4 – Логотип Dev

Dev публікує аналітичні матеріали, інтерв'ю з відомими постатями в галузі інформаційних технологій, статті про кібербезпеку, штучний інтелект та цифрові пристрої. Даний проєкт цікавий для розробників та фахівців, які хочуть бути в курсі трендів в ІТ-сфері.

При наявності таких серйозних конкурентів, «Цифровий світ» має знайти своє місце на ринку публікуючи якісний та унікальний контент. Наприклад, швидка публікація нових трендів та прогнози для майбутніх технологій від найкращих розробників України та світу. Це забезпечить інтерес до блогу досвідчених спеціалістів та студентів.

2. ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Проектування відношень між акторами та прецедентами

З системою блогу «Цифровий світ» будуть взаємодіяти три типи акторів: користувач, відвідувач та адміністратор. Основна роль акторів полягає в запусканні виконання сценаріїв роботи системи.

На рисунку 2.1.1 зображено діаграму варіантів використання для користувача. На діаграмах ВВ для позначення актора використовується форма людини. За допомогою ліній позначається зв'язок. Суцільна лінія означає, що користувач бере участь в дії. На даній діаграмі це означає, що «користувач» може залишати коментарі та працювати з постом. Лінія з незафарбованими трикутником позначають генералізацію, або цей зв'язок можна назвати успадкування, як в об'єктно орієнтованому програмуванні. Це означає, що прецедент «Створити пост» є нащадком прецеденту «Робота з постом» і він має всі властивості свого «батька». Штрихпунтира лінія з написом «include» показує, що це додаткова дія, без якої основний прецедент не обійдеться, тобто при виконанні «Створити пост» також виконаються прецеденти, такі як: «Додати фото», «Додати заголовок», «Додати теги» й «Додати статтю».

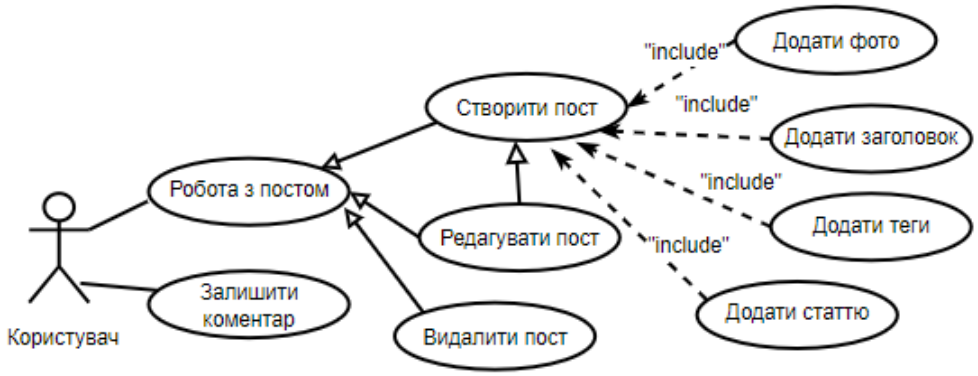

Рисунок 2.1.1 – Діаграма ВВ для користувача

«Користувач» – зареєстрований в системі користувач з доступом до основних функцій системи.

На рисунку 2.1.2 зображено діаграму варіантів використання для відвідувача блогу. Штрихпунтирна лінія з позначенням «extend» показує зв’язок при якому основний прецедент «Фільтрація постів» за бажанням користувача може розширитися двома додатковими функціями «Фільтрація за категоріями» та «Фільтрація за тегами».

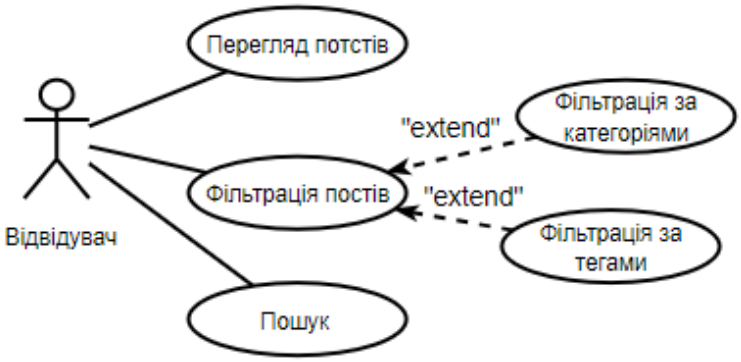

Рисунок 2.1.2 – Діаграма ВВ для відвідувача

Відвідувач – не зареєстрований користувач, який має обмежений доступ.

На рисунку 2.1.3 зображено діаграму варіантів використання для адміністратора.

Рисунок 2.1.3 – Діаграма ВВ для адміністратора

Адміністратор – це користувач, який керує системою та має не обмежений доступ до всіх функцій блогу.

Для детального розуміння значення кожного прецеденту в таблиці 2.1.1 наведено пояснення всіх прецедентів.

Таблиця 2.1.1 – Основні прецеденти блогу

№	Найменування	Актор	Формування
1	Створення посту	Користувач	Створення нового поста з текстом та зображенням
2	Редагування посту	Користувач	Зміна вмісту існуючих постів
3	Публікація	Користувач	Публікувати створеного поста
4	Видалення	Користувач	Видалення своїх постів
5	Додавання тегів	Користувач	Додавання тегу для категоризації постів
6	Коментування	Користувач, відвідувач	Коментування постів
7	Перегляд посту	Користувач, відвідувач	Перегляд опублікованих статей
8	Фільтрування	Користувач, відвідувач	Фільтрування постів за тегами
9	Пошук	Користувач, відвідувач	Пошук постів за ключовими словами
10	Керування користувачами	Адміністратор	Видалення користувачів системи
11	Статистика	Адміністратор	Перегляд статистики відвідування
12	Модерація	Адміністратор	Редагування вмісту статей

Наведені прецеденти разом із діаграмами формують цілісну картину функціонування блогу «Цифровий світ».

2.2 Декомпозиція системи на мікросервіси

При декомпозиції блогу «Цифровий світ» на окремі мікросервіси використано підхід розбивання системи за функціональними можливостями. Такий метод дозволив визначити незалежні компоненти, які виконують конкретні

завдання. Основні функціональні можливості блогу, які можна виділити окремим сервісом включають:

- Аутентифікацію та авторизацію користувачів;
- Створення, редагування та зберігання публікацій;
- Коментування постів;
- Пошук постів за ключовими словами;
- Збір та аналіз даних про активність користувачів.

На основі даного функціоналу систему розділено на такі міркосервіси: авторизація та керування користувачами, коментування, публікації, пошук та фільтрація, аналітика. На рисунку 2.2.1 зображено діаграму мікросервісів системи.

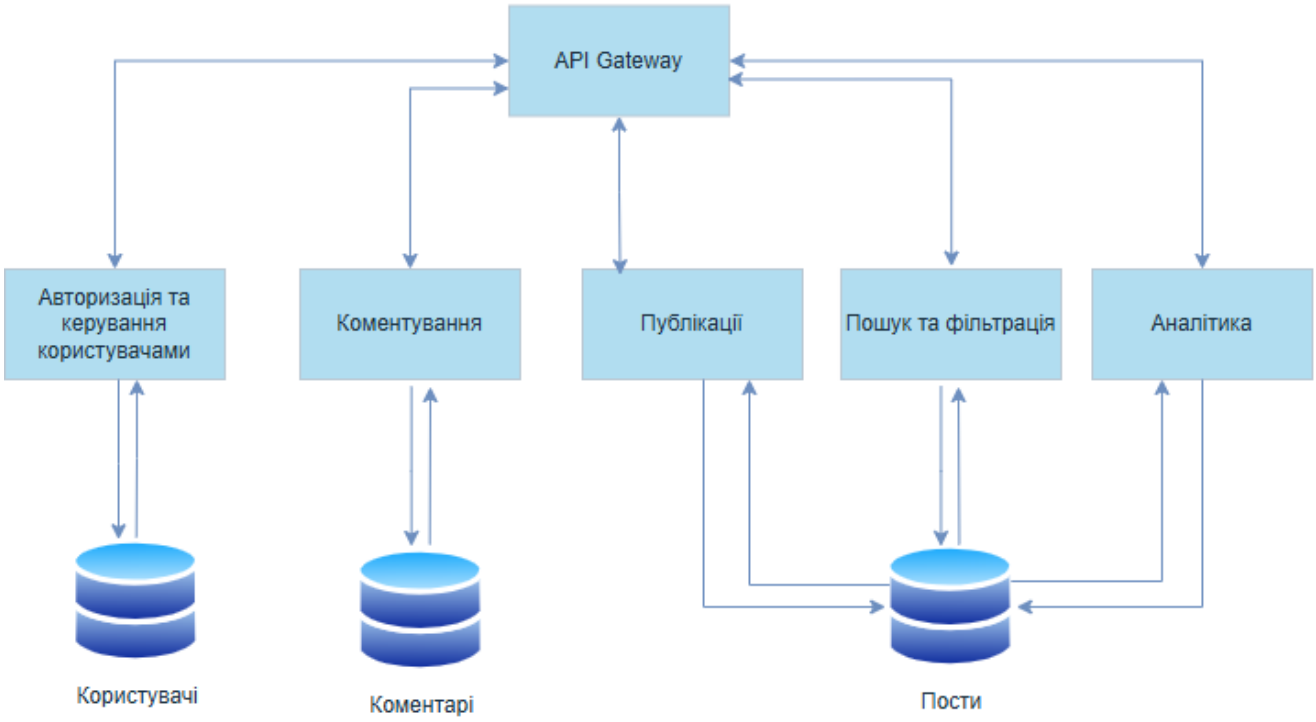

Рисунок 2.2.1 – Діаграма мікросервісів блогу

Мікросервіс «Авторизація та керування користувачами» реалізовує таку функціональність:

- Перевірка логіну та паролю під час авторизації;
- Надання прав редагування контенту в залежності від ролі в системі;
- Створення, редагування, видалення профілів користувачів блогу;

- Зберігання даних користувачів блогу.

Мікросервіс «Коментування» відповідає за таку функціональність:

- Додавання, редагування та видалення коментарів під постами;
- Відображення коментарів в хронологічному порядку;

Мікросервіс «Публікації» відповідає за такі функції, як:

- Публікація, редагування та видалення постів;
- Добавлення списку тегів, які асоціюються з контентом у статті;
- Прикріплення головного фото, яке відображає тему статті.

Мікросервіс «Пошук та фільтрація» відповідає за:

- Пошук статей за ключовими словами, які збігаються з заголовками постів;
- Фільтрацію за популярністю, за хронологією або за тегами.

Мікросервіс «Аналітика» відповідає за аналізування активності користувачів: підрахування вподобань, коментарів, складання рейтингів статей. На основі цих підрахунків створює звіти для покращення роботи блогу.

2.3 Визначення класів системи

Під час роботи з фреймворком NestJS, класи системи поділяються на Service та Controller і відіграють свою роль у мікросервісі. Класи з приставкою «Service» відповідають за бізнес-логіку, а класи з приставкою «Controller» відповідають за обробку запитів від клієнта.

Клас UserService відповідає за логіку управління користувачами, авторизацію, реєстрацію та оновлення профілю. Даний клас реалізований у мікросервісу «Авторизація та керування користувачами». На рисунку 2.3.1 зображено клас UserService.

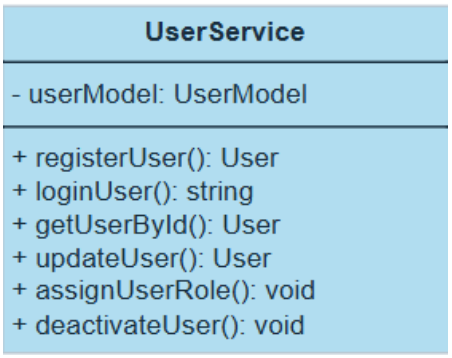

Рисунок 2.3.1 – Клас UserService

Даний клас має один приватний атрибут userModel, який використовується для доступу до бази даних користувачів. Також клас має шість публічних методів, роль кожного з методів описана у таблиці 2.3.1.

Таблиця 2.3.1 – Методи класу UserService

№	Назва методу	Призначення
1	registerUser	Створює нового користувача на основі вхідних даних
2	loginUser	Авторизовує користувача
3	getUserById	Повертає дані про користувача за його ідентифікатором
4	updateUser	Змінює дані користувача
5	assignUserRole	Призначає роль користувача в системі
6	deactivateUser	Деактивує користувача

Клас UserController призначений для обробки запитів, які надходять до мікросервісу «Авторизація та керування користувачами». Під час обробки запиту цей клас передає його класу UserService, а потім повертає відповідь. На рисунку 2.3.2 зображено методу класу UserController.

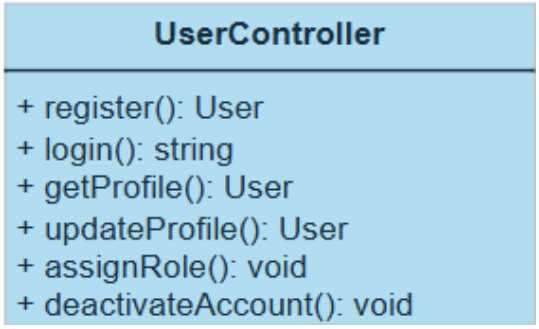

Рисунок 2.3.2 – Клас UserController

Клас UserController містить шість методів, які обробляють запити до мікросервісу та повертають результати.

1. Метод register обробляє запит на реєстрацію користувача та звертається до методу registerUser для виконання бізнес-логіки.
2. Метод login обробляє запит на авторизацію користувача та звертається до методу loginUser.
3. Метод getProfile обробляє запит на отримання даних про користувача за його ідентифікатором та звертається до методу getUserById для виконання запиту.
4. Метод updateProfile обробляє запит на редагування інформації про користувача та для виконання передає його методу updateUser.
5. Метод assignRole обробляє запит на присвоєння певної ролі користувачу та звертається до методу assignUserRole.
6. Метод deactivateAccount обробляє запит на деактивацію користувача та звертається до методу deactivateUser для виконання бізнес-логіки.

Клас PostService відповідає за управління постами блогу. Даний клас реалізований у мікросервісі «Публікації». На рисунку 2.3.3 зображено клас PostService та його методи.

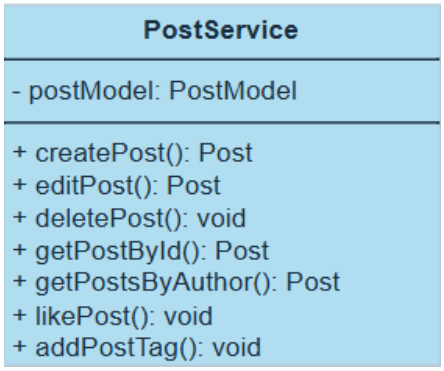

Рисунок 2.3.3 – Клас PostService

Даний клас містить один приватний атрибут postModel, який використовується для доступу до бази даних і постів та сім методів функціонал яких описано в таблиці 2.3.2.

Таблиця 2.3.2 – Методи класу PostService

№	Назва методу	Призначення
1	createPost	Створення статті
2	editPost	Редагування статті
3	deletePost	Видалення статті
4	getPostById	Отримання статті за індентифікаторм
5	getPostsByAuthor	Отримання всіх статей, які належать даному автору
6	likePost	Вподобання статті
7	addPostTag	Добавлення тегу до списку тегів статті

Клас PostController призначений для обробки запитів, які надходять до мікросервісу «Публікації». Даний клас в залежності від запиту звертається до методів класу PostService для виконання завдань, які передаються в запиті.

На рисунку 2.3.4 зображено клас PostController.

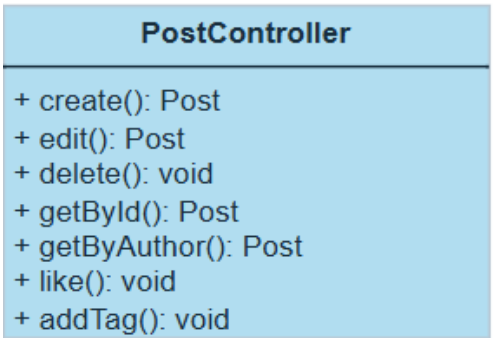

Рисунок 2.3.4 – Клас PostController

Даний клас містить сім методів, які обробляють запити до мікросервісу «Публікації» та повертають відповіді на них.

1. Метод create обробляє запит на створення статті та звертається до методу createPost для виконання цього завдання.
2. Метод edit обробляє запит на редагування вмісту статті та звертається до методу editPost.
3. Метод delete обробляє запит на видалення статті та звертається до методу deletePost
4. Метод getById обробляє запит на отримання статті за ідентифікатором та звертається до методу getPostById для виконання завдання

5. Метод getByAuthor обробляє запит на отримання списку постів, які написав вибраний автор та звертається до методу getPostsByAuthor.

6. Метод like обробляє запит на вподобання статті та звертається до likePost для збільшення кількості вподобань для статті.

7. Метод addTag обробляє запит на добавлення тегу до списку тегів статті та для виконання цієї логіки звертається до методу addPostTag.

Клас CommentService відповідає за бізнес-логіку, яка пов’язана публікацією коментарів під постом. Даний клас реалізований у мікросервісі «Коментарі». На рисунку 2.3.5 зображено класи CommentService.

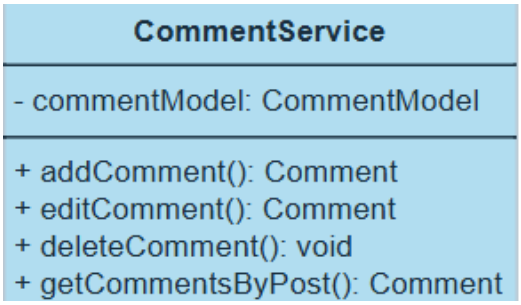

Рисунок 2.3.5 – Клас CommentService

Даний клас містить один приватний атрибут commentModel, який використовується для доступу до бази даних коментарів та зберігання списку об’єктів типу CommentModel. Також є чотири публічних методи. В таблиці 2.3.3 описано функціонал цих методів.

Таблиця 2.3.3 – Методи класу CommentService

№	Назва методу	Призначення
1	addComment	Публікація коментаря
2	editComment	Редагування коментаря
3	deleteComment	Видалення коментаря
4	getCommentsByPost	Отримання всіх коментарів, які опубліковані під даним постом

Клас CommentController призначений для обробки запитів, які надходять до мікросервісу «Коментування» та звертаються до методів класу CommentService для

виконання цих запитів. На рисунку 2.3.6 зображено клас CommentController та його методи.

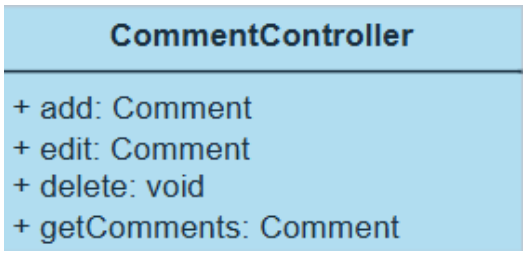

Рисунок 2.3.6 – Клас CommentController

Даний клас містить чотири методи, які обробляють запити та повертають відповіді на них.

1. Метод add обробляє запит на публікування коментаря під статтею та звертається до методу addComment.
2. Метод edit обробляє запит на редагування тексту коментаря та звертається до методу editComment.
3. Метод delete обробляє запит на видалення коментаря під постом та звертається до методу deleteComment.
4. Метод getComments обробляє запит на отримання всіх коментарів, які опубліковані під певним постом та звертається до методу deleteComments для виконання цього.

Клас SearchIndex відповідає за виконання пошуку та фільтрації постів. Даний клас реалізований у мікросервісі «Пошук та фільтрація». На рисунку 2.3.7 зображено клас SearchIndex та його методи.

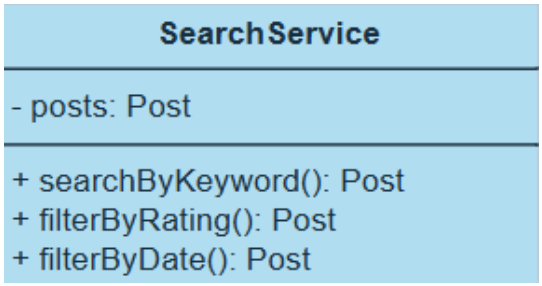

Рисунок 2.3.7 – Клас SearchService

Даний клас містить приватний атрибут posts, який призначений для зберігання списку об’єктів типу Post та три методи функціонал яких описано у таблиці 2.3.4.

Таблиця 2.3.4 – Методи класу SearchService

№	Назва методу	Призначення
1	searchByKeyword	Шукає пости за ключовим словом в заголовках або контенті
2	filterByRating	Фільтрує список постів за популярністю
3	filterByDate	Фільтрує список постів за їх хронологією

Клас SearchController обробляє запити, які надходять до мікросервісу «Пошук та фільтрація» та повертає результати оброблення цих запитів. Для виконання запитів даний клас звертається до методів класу SerchService. На рисунку 2.3.8 зображено клас SearchController та його методи.

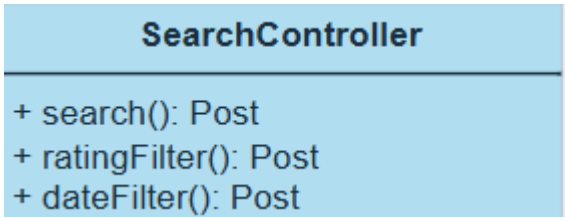

Рисунок 2.3.8 – Клас SearchController

Даний клас містить три методи, які обробляють запити та повертають результати.

1. Метод search обробляє запит на пошук статті за ключовим словом в заголовку або контенті та звертається до методу searchByKeyword для виконання запиту.

2. Метод ratingFilter обробляє запит на фільтрацію постів за популярністю та звертається до методу filterByRating.

3. Метод dateFilter обробляє запит на фільтрацію постів в хронологічному порядку та звертається до методу filterByDate.

Останнім класом є AnalyticsService, який призначений для збирання статистики активності користувачів блогу, для подальшого покращення роботи

системи. Даний клас розроблений в мікросервісі «Аналітика» та виконує завдання, які передає клас контролер, який обробляє запити до цього сервісу. На рисунку 2.3.9 зображено клас AnalyticsService.

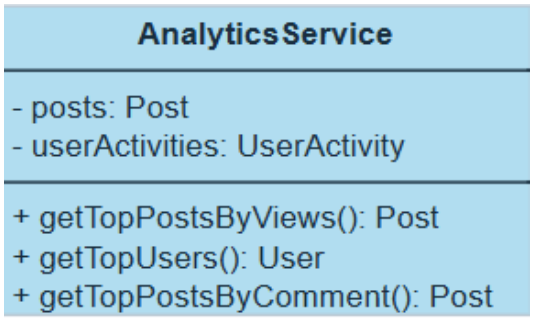

Рисунок 2.3.9 – Клас AnalyticsService

Даний клас містить два приватні атрибути. Перший posts для збереження списку елементів, які представляють собою інформацію про кожен пост – для підрахунку статистики. Другий атрибут userActivities для зберігання інфромації про активність користувачів. Також цей клас містить три публічні методи. Їх призначення описано в таблиці 2.3.5.

Таблиця 2.3.5 – Методи класу AnalyticsService

№	Назва методу	Призначення
1	getTopPostsByViews	Повертає найпопулярніші пости за кількістю переглядів
2	getTopUsers	Визначає найактивніших користувачів за кількістю їхніх дій
3	getTopPostsByComment	Повертає найбільш коментовані пости

Клас AnalyticsController обробляє запити, які надходять до мікросервісу «Аналітика» та передають завдання цих запитів відповідним методам класу AnalyticsService для виконання, а потім повертають результати. На рисунку 2.3.10 зображено клас AnalyticsController з його методами.

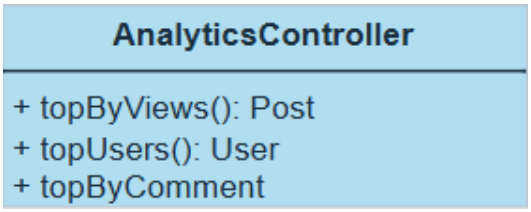

Рисунок 2.3.10 – Клас AnalyticsController

Методи даного класу обробляють запити, які приходять до мікросервісу:

1. Метод topByViews обробляє запит на отримання списку найпопулярніших постів за відвідуванням та звертається до методу getTopPostsByViews для виконання.

2. Метод topUsers обробляє запит на отримання списку найактивніших користувачів блогу та звертається до методу getTopUsers.

3. Метод topByComment обробляє запит на отримання найбільш обговорюваних постів та звертається до методу getTopPostsByComment для отримання результату.

На рисунку 2.3.11 зображено діаграму класів блогу «Цифровий світ».

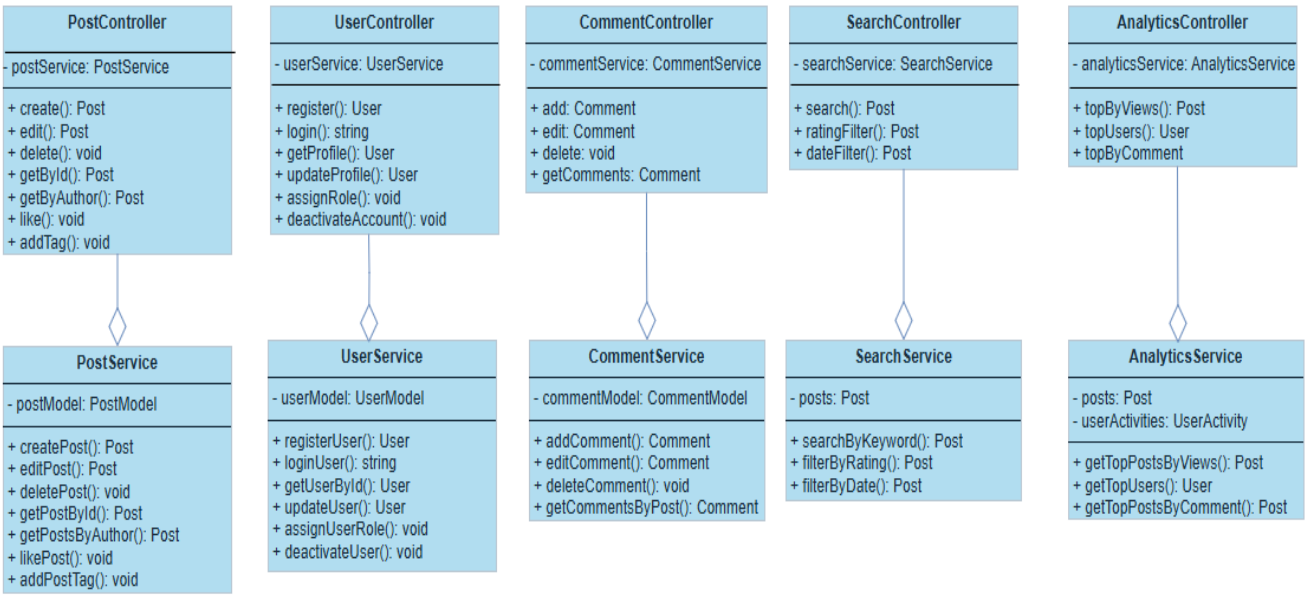

Рисунок 2.3.11 – Діаграма класів

Для реалізації кожного мікросервіса створено два класи: контролер та сервіс. Разом вся система складається з десяти класів, які відображають функціонал блогу «Цифровий світ».

2.4 Опис роботи системи

В даному пункті розглянуто основні сценарії використання системи блогу «Цифровий світ». Для кожного сценарію буде наведено опис процесу та діаграми послідовності, які ілюструють взаємодію між компонентами системи.

Першим головним сценарієм є процес створення поста. Основний потік:

1. Користувач заповнює форму для створення поста;
2. Після заповнення форми натискає кнопку «Зберегти»;
3. Запит надсилається на сервер до PostController з мікросервісу «Публікація»;
4. PostController передає запит до PostService;
5. PostService виконує метод createPost, зберігає пост у базі даних.
6. Після успішного створення поста повертається відповідь користувачу про успішне створення.

На рисунку 2.4.1 зображено діаграму послідовностей для створення нової статті.

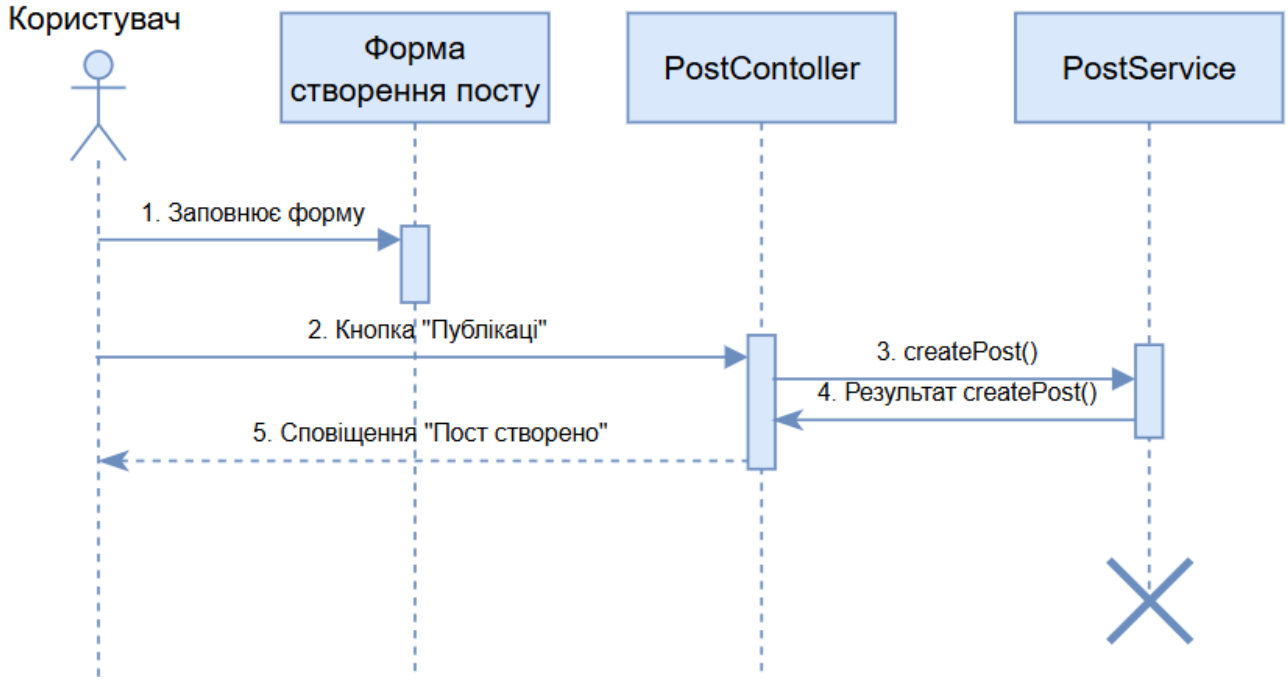

Рисунок 2.4.1 – Діаграма послідовності публікації статті

Наступним головним сценарієм є коментування статті. Основний потік:

1. Користувач залишає коментар під постом;
2. Запит на додавання коментаря надсилається до CommentController з мікросервісу «Коментування»;
3. CommentController передає запит до CommentService;
4. CommetnService виконує метод addComment та зберігає коментар у базі даних.
5. Користувачу відображається його коментар.

На рисунку 2.4.2 зображено діаграму послідовності для сценарію коментування посту.

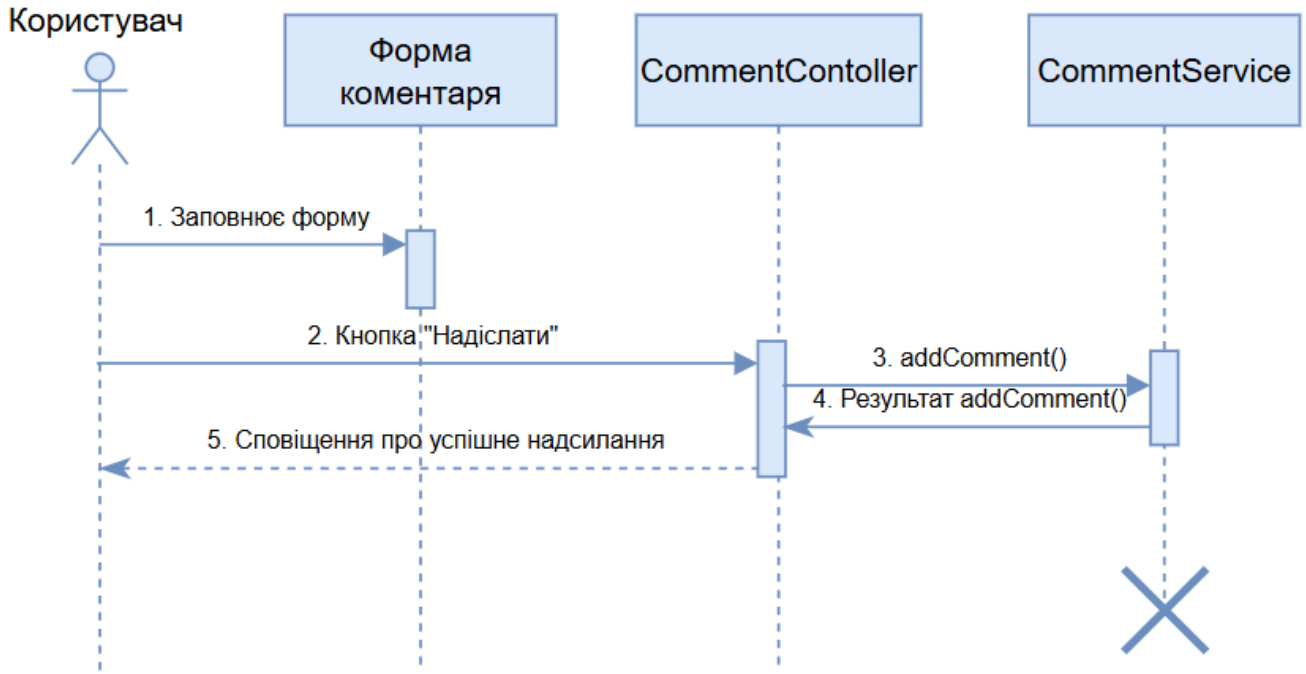

Рисунок 2.4.2 – Діаграма послідовності для коментування

Одним з головних сценаріїв, які найчастіше виконуються при користуванні блогом – пошук постів. Основний потік:

1. Користувач вводить ключове слово у пошуковий рядок;
2. Після того, як користувач перестає натискати на клавіші запит надсилається до SearchController з мікросервісу «Пошук та фільтрація»;
3. SearchController передає запит до SearchService;
4. SearchService виконує метод searchByKeyword, знаходить відповідні пости;
5. Після успішного виконання пошуку повертаються знайдені пости.

На рисунку 2.4.3 зображено діаграму послідовностей пошуку статті.

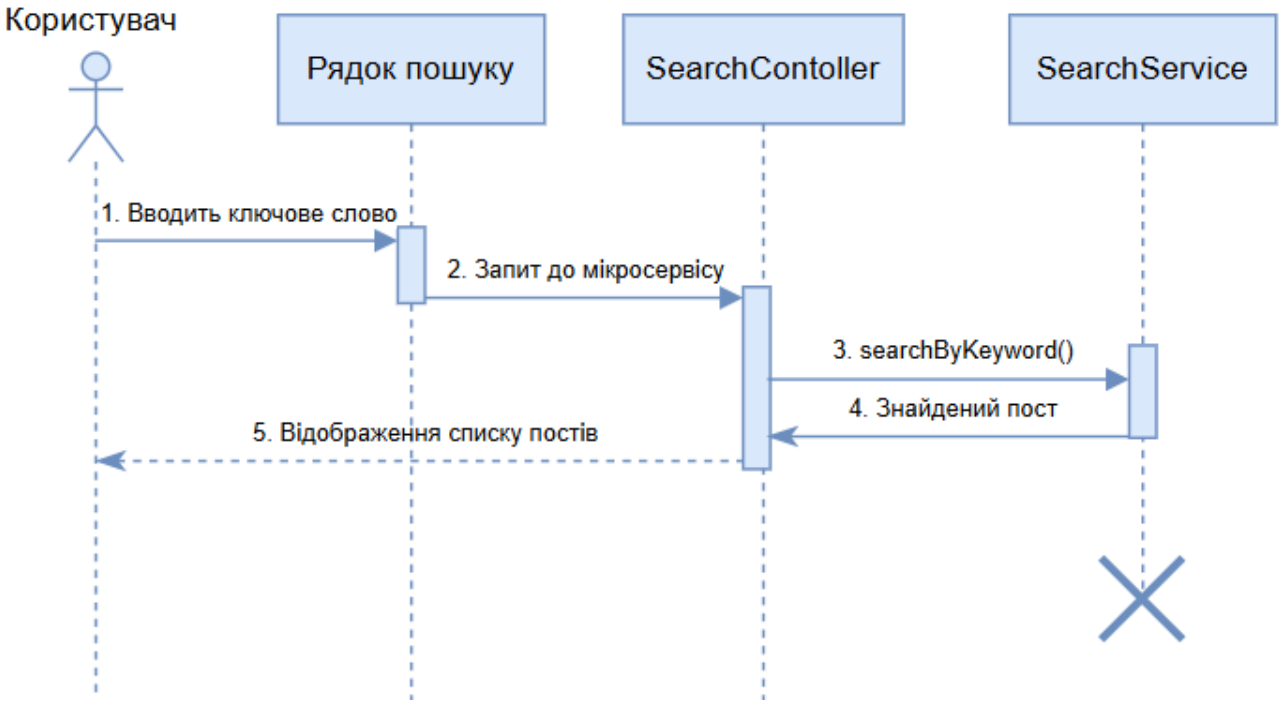

Рисунок 2.4.3 – Діаграма послідовності пошуку статті

Останнім головним сценарієм, який найчастіше виконується в кожній системі це авторизація користувача. Основний потік:

1. Користувач вводить логін та пароль у формі авторизації;
2. Після натискання кнопки «Увійти» запит надсилається до мікросервісу «Авторизація та керування корисутувачами», який обробляє UserController;
3. UserController передає запит до UserService;
4. UserService виконує метод loginUser, перевіряє дані користувача та авторизує його;
5. Після успішної авторизації користувачу відкривається доступ до більшої кількості функцій.

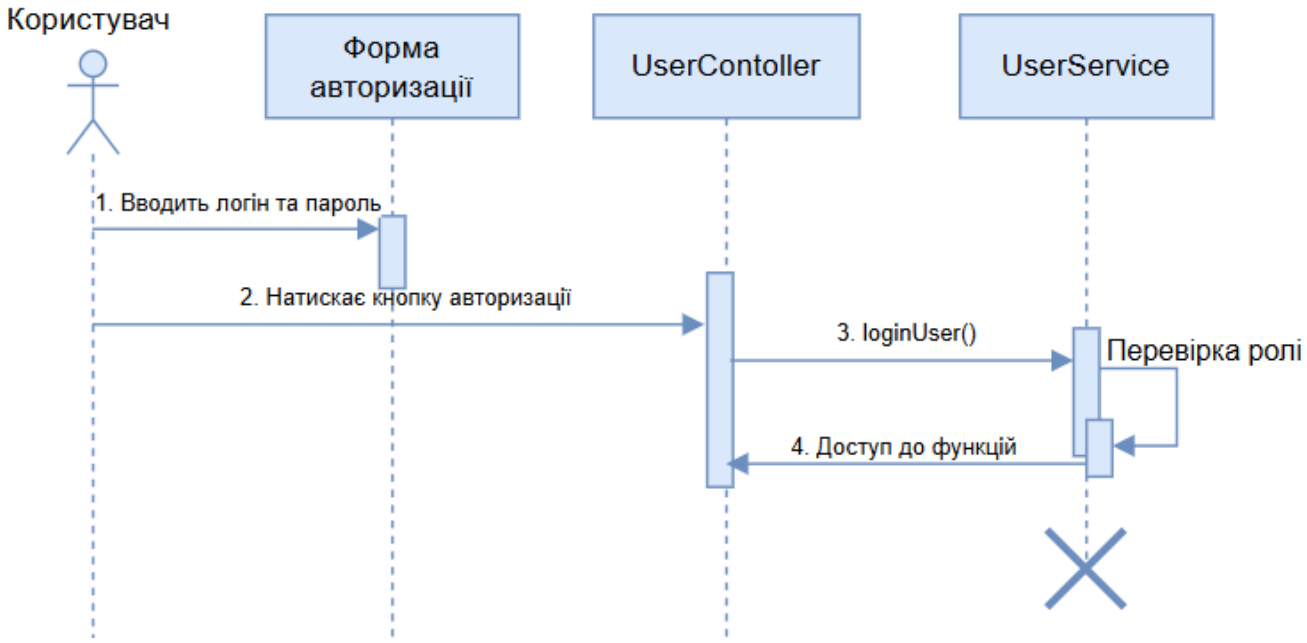

Рисунок 2.4.4 – Діаграма послідовності для авторизації

Розглянуто основні сценарії використання блогу "Цифровий світ". Зокрема, було описано процеси публікації статті, коментування, авторизації та пошуку. За допомогою діаграм послідовності показано принцип роботи системи.

3. КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Розробка мікросервісів блогу

Для розробки мікросервісів для блогу «Цифровий світ» використано фреймворк NestJS. Він є одним з найпопулярніших фреймворків для Node.js, бо використовує TypeScript та принципи об’єктно-орієнтованого програмування. NestJS дає можливість легко створювати та підтримувати масштабовані додатки, використовуючи свою модульну архітектуру, що робить його ідеальним варіантом для написання додатку з мікросервісною архітектурою.

Мікросервісна архітектура передбачає розподіл функціональності додатку на окремі сервіси, кожен з яких виконує конкретну задачу та може бути розгорнутим окремо. В даному розділі описано реалізацію тільки основних методів системи.

Першим мікросервісом було розроблено «Авторизацію та керування користувачми». Основні методи класу UserController (рис. 3.1.1).

```
@Controller()
export class UserController {
  constructor(private readonly userService: UserService) {}

  @MessagePattern('login')
  async login(data: LoginDto) {
    return this.userService.loginUser(data);
  }

  @MessagePattern('getProfile')
  async getProfile( id: string) {
    return this.userService.getUserById(id);
  }

  @MessagePattern('assignRole')
  async assignRole( data: { id: string; role: string }) {
    const { id, role } = data;
    return this.userService.assignUserRole(id, role);
  }
}
```

Рисунок 3.1.1 – Методи класу UserController

Даний код демонструє реалізацію контролера UserController використовуючи фреймворк NestJS. Він відповідає за обробку запитів, які надходять через RabbitMQ, які пов’язані з користувачами та передає запити до сервісу для виконання бізнес-логіки. Контролер надає кінцеві точки для взаємодії з користувачами, такі як: авторизація, отримання інформації про профіль та призначення ролі.

За допомогою «@Contoller()» клас стає класом контролером. Анотація «@MessagePattern» дозволяє контролеру обробляти повідомлення, що надходять через RabbitMQ. Кожен метод вказує на певний шаблон повідомлення, наприклад, MessagePattern(‘login’) буде обробляти запити, які в своєму тілі будуть містить ключ «pattern» зі значенням login.

В класі UserService методи реалізовують бізнес-логіку за результатами якої до них звертаються методи контролера. На рисунку 3.1.2 зображено код методу loginUser.

```
async loginUser(data: LoginDto): Promise<any> {
    const { username, password } = data;
    const user = await this.userModel.findOne({ username }).exec();

    if (!user || user.password !== password) {
      throw new UnauthorizedException('Invalid credentials');
    }
    const payload = { username: user.username, sub: user._id };
    const accessToken = this.jwtService.sign(payload);

    return { message: 'Login successful', user, accessToken };
  }
```

Рисунок 3.1.2 – Метод loginUser

В асинхронному методі loginUser спочатку за допомогою деструктуризації витягується логін та пароль, який був переданий. Після цього відбувається пошук користувача в базі даних та результат пошуку записується в змінну user. Після проходить перевірка, якщо користувача з даним ім’ям не знайдено, або введений пароль не відповідає з паролем збереженим в базі даних, то повертається помилка з повідомленням «Invalid credentials», якщо перевірка проходить успішно, то

генерується jwt-токен та повертається об’єкт user, в якому зберігається інформація про користувача, токен та повідомлення «Login successful».

На рисунку 3.1.3 зображено код методів getUserById та assignUserRole.

```
async getUserById(id: string): Promise<User> {
  const user = await this.userModel.findById(id).exec();
  if (!user) {
    throw new NotFoundException('User not found');
  }
  return user;
}

async assignUserRole(id: string, role: string): Promise<User> {
  const user = await this.userModel.findById(id).exec();
  if (!user) {
    throw new NotFoundException('User not found');
  }

  user.role = role;
  return await user.save();
}
```

Рисунок 3.1.3 – Методи класу UserService

Метод getUserById отримує змінну id, яка має тип string. В змінну user записується результат асинхронного пошуку користувача в базі даних. Після цього відбувається перевірка, якщо користувач з даним ідентифікатором не знайдено, то повертається помилка з повідомленням «User not found», при успішному результаті повертається об’єкт user, який містить інформацію про користувача.

Метод assignUserRole відповідає за призначення ролі користувача системи. Для цього він отримує ідентифікатор користувача та назву ролі, яка передається в змінні role. Після цього відбувається пошук користувача, якщо користувач з даним ідентифікатором знайдений, то йому призначається роль та зміни зберігаються в базі даних.

Наступним було розроблено мікросервіс «Публікації», який відповідає за роботу з постами. Було створено контролер PostController, який обробляє запити брокера повідомлень з чергою «posts» та звертається до методів класу PostService,

які реалізовують бізнес-логіку. На рисунку 3.1.4 зображено код методів createPost та likePost.

```
async createPost(dataPost: CreatePostDto): Promise<Post> {
    const createdPost = new this.postModel(dataPost);
    return createdPost.save();
  }

  async likePost(id: string): Promise<Post> {
    const post = await this.postModel.findById(id).exec();
    if (!post) {
      throw new NotFoundException('Post not found');
    }

    post.likes += 1;
    return post.save();
  }
```

Рисунок 3.1.4 – Створення та вподобання поста

Метод createPost отримує типізовані дані про статтю, яка буде опублікована. Вони містять: фото, заголовок, основний текс та список тегів. В об’єкт createdPost записується ця інформація а після цього цей об’єкт зберігається в базі даних.

Метод likePost відповідає за вподобання статті. Для цього даний метод отримує ідентифікатор статті. Після цього відбувається пошук статті в базі даних та результат записується в змінну post. Після цього відбувається перевірка, якщо змінна post не містить інформації про знайдену статтю, то повертається помилка, в іншому випадку в об’єкті post поле likes інкрементується на одиницю і зміни зберігаються з в базі даних. На рисунку 3.1.5 зображено код методу editPost.

```
async editPost(dataPost: EditPostDto): Promise<Post> {
    const {id, title, content, imageUrl, tagList} = dataPost;
    const existingPost = await this.postModel.findById(id).exec();
    if (!existingPost) {
      throw new NotFoundException('Post not found');
    }

    existingPost.title = title;
    existingPost.content = content ;
    existingPost.imageUrl = imageUrl;
    existingPost.tagList = tagList;

    return existingPost.save();
  }
```

Рисунок 3.1.5 – Редагування статті

Метод editPost приймає об’єкт, який містить ідентифікатор статті та всі поля, які мають бути модифіковані. В змінну existingPost записується результат пошуку статті в базі даних за ідентифікатором. Після цього відбувається перевірка, якщо стаття була знайдена, то поля перезаписуються та зберігаються зміни.

Наступним було розроблено мікросервіс «Коментування». Даний мікросервіс обробляє запити брокера повідомлень з чергою «comment». Методи класу CommentService реалізовують роботу з коментарями. На рисунку 3.1.6 зображено код основних методів даного класу.

```
async addComment(dataComment: AddCommentDto): Promise<Comment> {
    const { postId, userId, content } = dataComment;
    const newComment = new this.commentModel({ postId, userId, content });
    return newComment.save();
  }

  async getCommentsByPost(postId: string): Promise<Comment[]> {
    const comments = await this.commentModel.find({ postId }).exec();
    if (!comments || comments.length === 0) {
      throw new NotFoundException('No comments found for this post');
    }
    return comments;
  }
```

Рисунок 3.1.6 – Основні методи CommentService

Метод addComment приймає об’єкт, який зберігає інформацію про коментар. Після цього за допомогою деструктуризації з цього об’єкта витягують такі змінні: postId – ідентифікатор поста; userId – ідентифікатор користувача, та content – текст коментаря. Після цього створюється об’єкт для збереження коментаря та зберігається в базі даних коментарів.

Метод getCommetnsByPost відповідає за повернення списку коментарів, які були опубліковані під певним постом. Отже, даний метод приймає ідентифікатор статті, після цього виконується пошук в базі даних коментарі, які належать даному посту та результат присвоюється змінній comments. Далі проводиться перевірка, якщо коментарів для даної статті не знайдено, то повертається помилка з

повідомленням «No comments found for this post», в іншому випадку повертається масив коментарів.

Після цього було розроблено мікросервіс «Пошук та фільтрування». Створено клас контролер, який обробляє запити брокера повідомлень з чергою «search». Методи класу контролера обробляють запити за такими адресами: пошук за ключовим словом – «searchKeyword»; фільтрація за рейтингом – «searchRating» фільтрація за хронологією – «searchDate».

Бізнес-логіку даних запитів виконують методи класу SearchService вони зображені на рисунку 3.1.7.

```
async searchByKeyword(keyword: string): Promise<Post[]> {
    return this.postModel.find({
      $or: [
        { title: { $regex: keyword, $options: 'i' } },
        { content: { $regex: keyword, $options: 'i' } }
      ]
    }).exec();
  }

  async filterByRating(minRating: number): Promise<Post[]> {
    return this.postModel.find({
      rating: { $gte: minRating }
    }).sort({ rating: -1 }).exec();
  }

  async filterByDate(order: 'asc' | 'desc'): Promise<Post[]> {
    return this.postModel.find().sort({ createdAt: order === 'asc' ? 1 : -1 }).exec();
  }
```

Рисунок 3.1.7 – Методи пошуку та фільтрації

Метод searchByKeyword отримує ключове слово для пошуку. Після цього використовує його для пошуку в базі даних за допомогою функції find. Для того, пошук відбувався в заголовках та в контентів самої статті використовується оператор «$or» та вписується дві умови. Регулярний вираз «$regrex» шукає збіги в полях title та content. Опція «i» дає вказівку не звертати уваги на регістр. Знайдені статті повертаються даним методом до контролера, який відправляє їх клієнту.

Метод filterByDate здійснює фільтрацію постів за датою їх створення та дозволяє повернути у хронологічному або зворотному порядку. Даний метод приймає «asc» або «desc». Після цього знаходить всі пости з бази даних і за

допомогою функції sort сортує за полем createdAt згідно вхідної умови. Якщо order дорівнює asc, то сортування відбувається від найстаріших до найновіших, якщо дорівнює desc, тоді від найновіших до найстаріших.

Таким чином, архітектура системи побудована на мікросервісній архітектурі з використанням фреймворка NestJS, демонструє високу надійність, модульність та ефективність. Всі розроблені мікросервіси інтегруються у єдину систему, забезпечуючи її стабільну роботу та відповідність сучасним стандартам розробки веб-додатків.

3.2 Розробка клієнтської частини блогу

3.2.1 Компонентний підхід

Бібліотека React – один з найкращих виборів для створення клієнтської частини додатку. Найбільшою перевагою даної бібліотеки є використання компонентів для створення інтерфейсу. Вони дозволяють повторно використовувати код, який відображає певні частини додатку. Завдяки цьому React дозволяє спростити та пришвидшити процес розробки.

Компонентний підхід базується на принципі модульності. Інтерфейс додатку розбивається на малі, незалежні компоненти. Кожен компонент виконує свою функцію.

Для створення блогу «Цифровий світ» створено цілу ієрархію компонентів. З другорядних можна виділити компонент Header, Logo, About, ContactInfo. Основні компоненти зображені на рисунку 3.2.1.1.

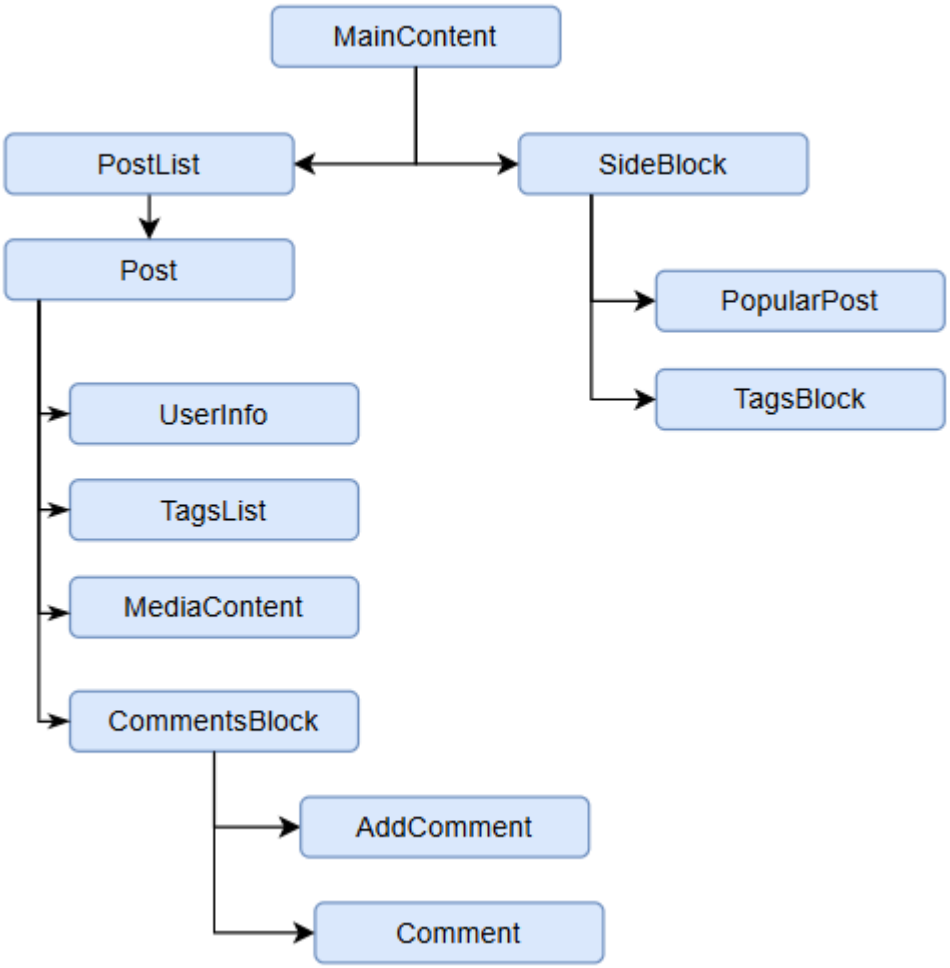

Рисунок 3.2.1.1 – Основні компоненти

Дані компоненти мають таку відповідальність:

1. PostList – відображення списку постів блогу;
2. Post – відображення посту;
3. UserInfo – відображення інформації про автора посту;
4. TagsList – відображення списку тегів;
5. MediaContent – показ медіа вмісту;
6. CommentsBlock – відображення списку коментарів пов’язаних з постом;
7. AddCommetn – містить форму для створення коментаря для посту;
8. Comment – відображає один коментар: автор, текст, дата створення;
9. SideBlock – бокова панель з популярними тегами та постами;
10. PopularPosts – відображає список найпопулярніших статей;
11. TagsBlock – список найпопулярніших тегів в блозі.

Всі компоненти це функції, які можна поділити на дві частини. В першій вони зберігають змінні, хуки, допоміжні функції для виконання бізнес-логіки. Друга частина це те, що функції повертають, а саме JSX-розмітка. JSX це синтаксис, який дозволяє використовувати html розмітку та JavaScript одночасно. Саме в JSX розмітці використовують змінні та результати функцій компонента для відображення динамічного інтерфейсу. На рисунку 3.2.1.2 зображено код компонента CommentsBlock.

```
export const CommentsBlock = ({ items, children }) => {
  return (
    <SideBlock title="Коментарі">
      <List>
        {items.map((obj, index) => (
          <React.Fragment key={index}>
            <ListItem alignItems="flex-start">
              <ListItemAvatar>
                {<Avatar alt={obj.user.fullName} src={`${pro
cess.env.REACT_APP_API_URL}${obj.user.avatarUrl}`} />}
              </ListItemAvatar>
              {
                <ListItemText
                  primary={obj.user.fullName}
                  secondary={obj.text}
                />
              }
            </ListItem>
            <Divider variant="inset" component="li" />
          </React.Fragment>
        ))}
      </List>
      {children}
    </SideBlock>
```

Рисунок 3.2.1.2 – Компонент CommentsBlock

Також за принципом створення компонентів, створено сторінки для додатку. Блог «Цифровий світ» має такі сторінки: створення посту, авторизація, реєстрація, сторінка повної статті, домашня сторінка.

3.2.2 Маршрутизація блогу

Маршрутизація – важлива частина клієнтської частини блогу. Вона надає можливість користувачам переходити з сторінки на сторінку без перезавантаження вебсторінки. Для реалізації маршрутизації у веб додатку «Цифровий світ» використано бібліотеку React Router, яка забезпечує просту організацію для Single Page Application.

Для налаштування маршрутизації блогу використано компонент «Routes». Кожен маршрут визначає URL адресу та компонент, який має відображатися за цим шляхом. На рисунку 3.2.2.1 зображено маршрути, які створено для блогу.

```
<Routes>
        <Route path="/" element={<Home />} />
        <Route path="/posts/:id" element={<FullPost />} />
        <Route path="/tags/:name" element={<TagPosts />} />
        <Route path="/posts/:id/edit" element={<AddPost />} />
        <Route path="/add-post" element={<AddPost />} />
        <Route path="/login" element={<Login />} />
        <Route path="/register" element={<Registration />} />
      </Routes>
```

Рисунок 3.2.2.1 – Налаштування маршрутизації

Створено сім маршрутів, які відповідають за відображення таких сторінок:

1. Перший маршрут «/» використовує компонент Home для відображення головної сторінки;
2. Другий маршрут «/posts/:id» використовується для перегляду окремого посту. Динамічна змінна «:id» представляє ідентифікатор статті;
3. Третій маршрут «/tags/:name» використовує компонент TagPosts для відображення списку постів, пов’язаних з вибираним тегом, який передається в змінній «:name»;

4. Четвертий маршрут «/post/:id/edit» використовується для редагування існуючого посту. Компонент AddPost виконує роль форми для створення та редагування;
5. П’ятий маршрут «/add-post» відображає форму для створення нового посту;
6. Шостий маршрут «/login» використовується для авторизації користувача. Компонент Login відображає форму авторизації;
7. Сьомий маршрут «/register» відображає форму реєстрації користувача в системі за допомогою компонента Registration.

Коли користувач вводить URL або взаємодіє з посилання в блозі, то React Router аналізує поточний шлях і визначає, який компонент потрібно відобразити.

Використання даної бібліотеки дозволило забезпечити зручне використання навігації блогу для кінцевого користувача.

3.3 Тестування блогу

Тестування це один з найважливіших етапів розробки програмного забезпечення. Він впливає на коректу робото розробленого функціоналу. В рамках блогу «Цифровий світ» тестування проводилося в мануальному режимі, що передбачає безпосередню перевірку роботи функціоналу розробником. Такий підхід дозволив симулювати поведінку реального користувача та перевірити коректність роботи всіх основних функцій блогу.

Ручне тестування включало перевірку:

1. Створення, редагування, видалення та відображення постів;
2. Додавання та відображення коментарів;
3. Відображення фільтрації постів за тегами;
4. Навігація між сторінками блогу;
5. Перевірка взаємодії клієнта з сервером через API.

При відкритті блогу користувача вітає головна сторінка (рис. 3.3.1). Дана сторінка відображається незалежно від того, чи користувач є авторизованим.

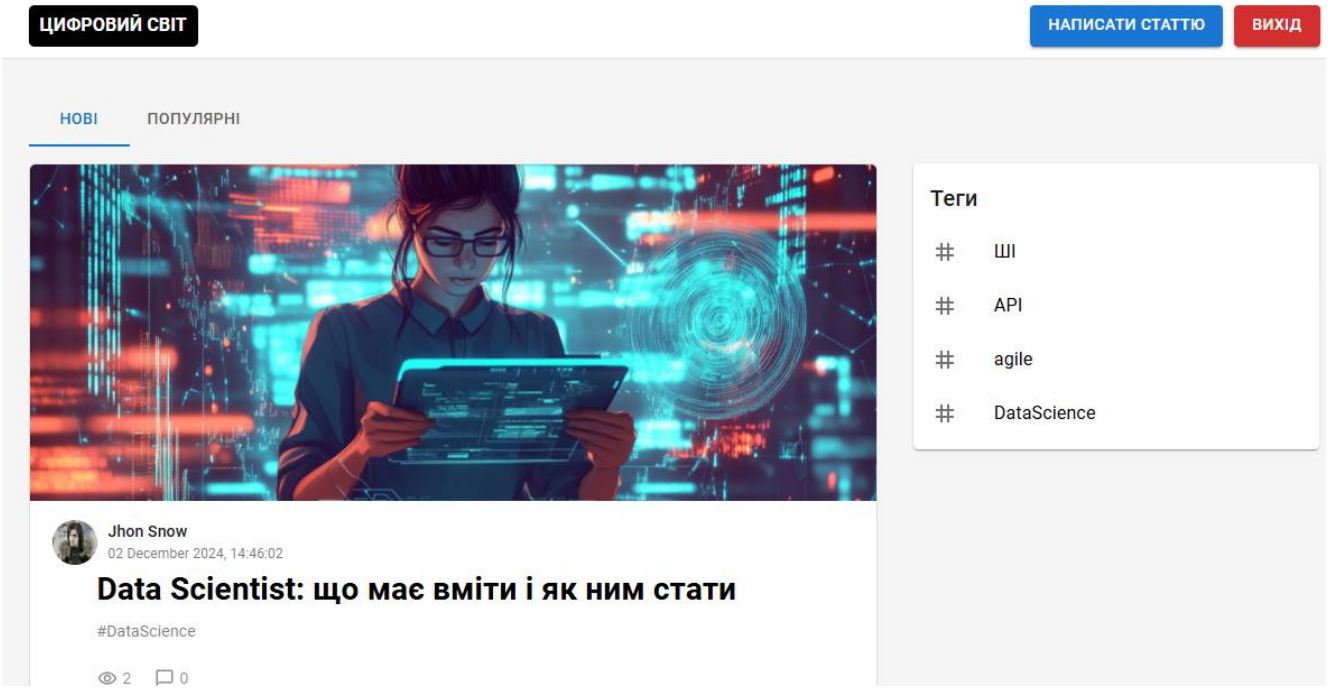

Рисунок 3.3.1 – Головна сторінка

У верхній частині головної сторінки розміщена «шапка» блогу. З лівої сторони відображається назва «Цифровий світ». З правої дві кнопки «Написати статтю» та «Вихід», якщо користувач не авторизований, то ці кнопки зміняться на «Вхід» та «Створити акаунт» (рис. 3.3.2).

Рисунок 3.3.2 – Кнопки без авторизації

Над списком постів розміщено дві кнопки «Нові» та «Популярні». З допомогою них можна змінювати відображення постів в хронологічній послідовності, або в за популярністю. За замовчуванням список відображається за першим критерієм. Текст вибраного способу підсвічується синім кольором та підкреслений синьої рискою для розуміння який спосіб сортування використовується.

Праворуч від списку постів відображається список чотирьох найпопулярніших тегів (рис. 3.3.3). Даний список надає список надає можливість категоризації статей за тегами, при натисканні на тег відображуються всі пости з даним тегом.

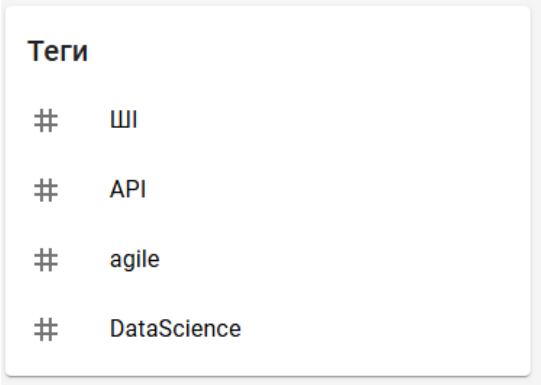

Рисунок 3.3.3 – Популярні теги

Основним контентом є пости. При перегляді постів з головної сторінки користувачу відображається: зображення, аватар автора, ім’я автора, дата публікації, заголовок, список тегів, кількість переглядів та кількість коментарів. На рисунку 3.3.4 зображено пост з списку.

Рисунок 3.3.4 – Структура інформації статті

Заголовок кожної статті – це посилання, яке перенаправляє на сторінку на якій відображається повна стаття. На рисунку 3.3.5 зображено вигляд повної статті.

Рисунок 3.3.5 – Повна стаття

Після головної інформації статті розміщено блок з коментарями. Тут можна побачити всі коментарі, які належать даній статті та залишити новий (рис. 3.3.6).

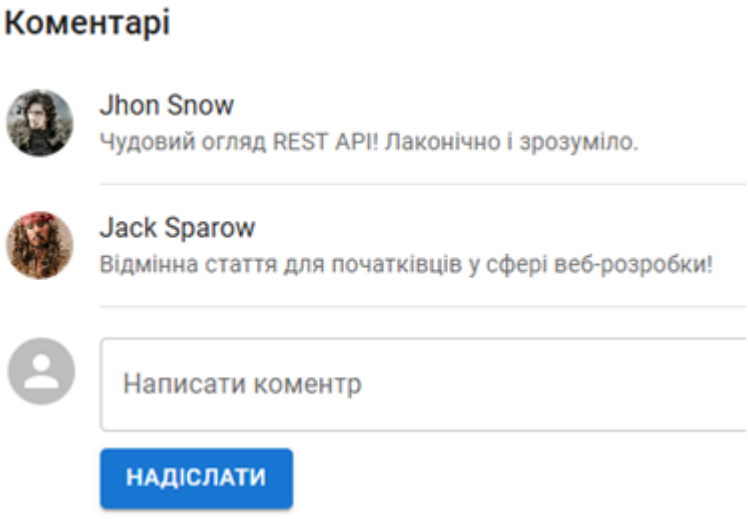

Рисунок 3.3.6 – Блок коментарів

Для створення нового посту потрібно авторизуватися та натиснути на кнопку «Написати статтю», яка є посиланням на сторінку з формою створення статті. На рисунку 3.3.7 зображено дану форму.

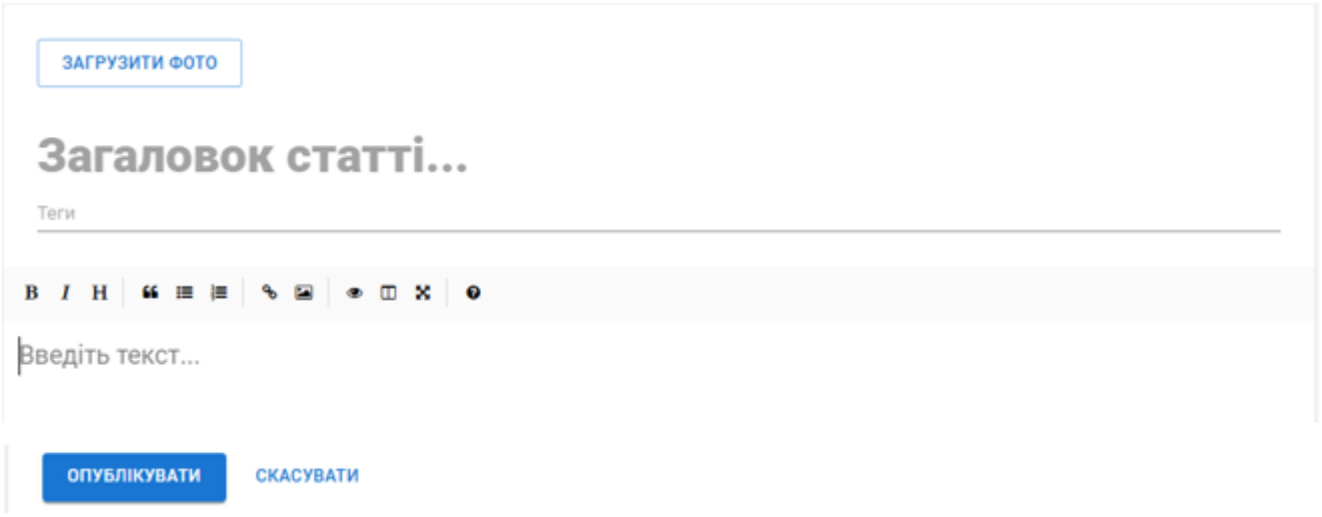

Рисунок 3.3.7 – Форма створення статті

Для публікації завантаження фото потрібно натиснути на кнопку «Загрузити фото» та вибрати потрібний файл. Після цього потрібно ввести назву статті, теги, які асоціюються з даною тематикою та ввести текст.

При написанні статті автор може стилізувати текст за допомогою розмітки Markdown використовуючи спеціальні кнопки, які розміщенні над полем вводу тексту (рис. 3.3.8).

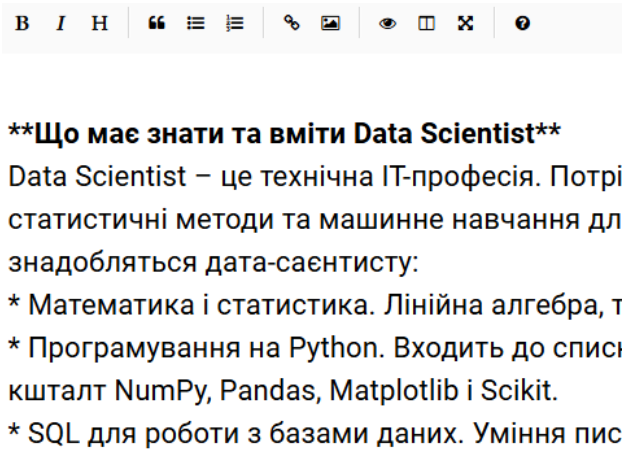

Рисунок 3.3.8 – Розмітка Markdown

При наведенні на статтю для автора з’являються кнопки для редагування та видалення статті (рис. 3.3.9).

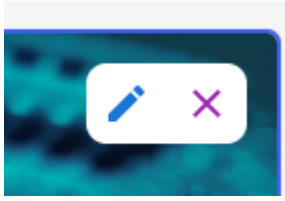

Рисунок 3.3.9 – Кнопки редагування та видалення

Для авторизації потрібно натиснути кнопку «Вхід» та заповнити форму авторизації: електрона адреса та пароль (рис. 3.3.10).

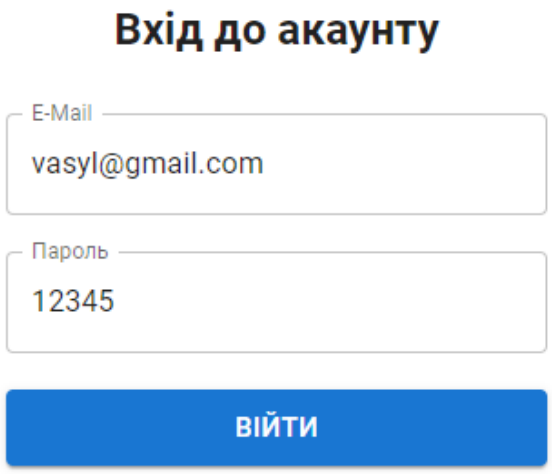

Рисунок 3.3.10 – Авторизація

Для реєстрації потрібно натиснути кнопку «Створити акаунт» та заповнити форму, яка складається з вибору фото, ім’я, електрона адреса, пароль. Поки всі поля не будуть заповненні, то кнопка «Створити обліковий запис» буде неактивною.

На рисунку 3.3.11 зображено сторінку реєстрації.

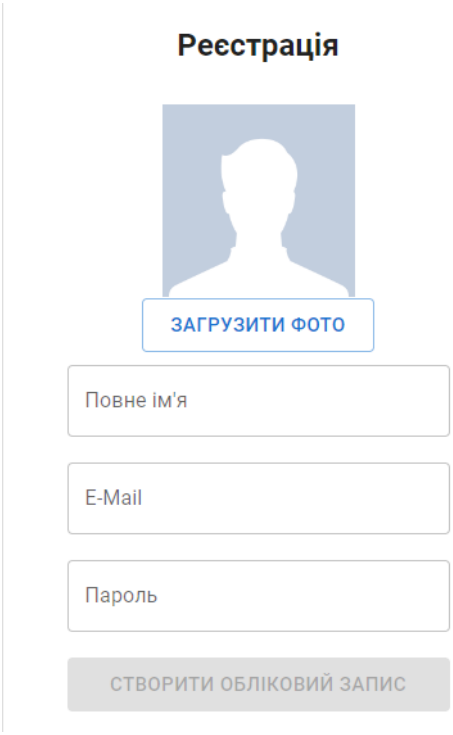

Рисунок 3.3.11 – Реєстрація

Після проведення тестування в мануальному режимі виявлено, що основні функції блогу «Цифровий світ» функціонують стабільно та відповідають вимогам, визначеним на етапі проєктування. Усі основні функції, такі як створення, редагування, видалення постів, робота з коментарями та тегами, а також маршрутизація між сторінками, працюють коректно. Інтеграція клієнта з сервером через API була успішно перевірена, що підтвердило правильність обробки запитів та відповіді сервера.

Під час тестування були виявлені та виправлені декілька незначних помилок, таких як некоректне відображення деяких елементів інтерфейсу та проблеми з маршрутизацією. Після внесення виправлень тестові сценарії були повторно виконані, і всі тести пройшли успішно.

Таким чином, можна зробити висновок, що клієнтська частина блогу готова до використання.

4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Охорона праці

В кваліфікаційній роботі магістра на тему “Аналіз та впровадження мікросервісної архітектури для створення масштабованого блогу “Цифровий світ””, здійснюється розробка блогу "Цифровий світ" на основі мікросервісної архітектури. Зокрема, вирішуються завдання: проєктування, створення та практична реалізація окремих мікросервісів для забезпечення функціональності блогу. Реалізація цих завдань вимагає використання різного роду обладнання — комп’ютерної техніки, серверів тощо. У зв’язку з цим постає низка важливих питань, пов’язаних із обов’язковим дотриманням вимог з охорони праці та правил техніки безпеки на робочому місці.

Для працівників, які будуть працювати над створенням та подальшим підтриманням блогу “Цифровий світ” потрібно організувати робочі місця з урахуванням вимог безпеки до роботи працівників з екранними пристроями [13]:

1. Робочі місця працівників з екранними пристроями мають бути спроектовані так і мати такі розміри, щоб працівники мали простір для зміни робочого положення та рухів.

2. Освітлення робочого місця працівника з екранними пристроями має створювати відповідний контраст між екраном і навколишнім середовищем (з урахуванням виду роботи) та відповідати вимогам ДСанПІН 3.3.2.007-98 [14].

3. Екранні пристрої не мають бути джерелом ризику для працівників.

4. Усе випромінювання, за винятком видимої частини електромагнітного спектра, має бути зведене до незначного рівня з погляду безпеки і охорони здоров’я працівників.

5. Символи на екранних пристроях мають бути чіткими, відповідного розміру. Між символами і рядками символів має бути належна відстань.

6. Зображення на екрані має бути стабільним, без миготінь або інших видів нестабільності.

7. Яскравість та контрастність символів має легко регулюватися працівником під час роботи з екранними пристроями, а також швидко адаптуватися до навколишніх умов.

8. Вибираючи екрани, слід надавати перевагу таким екранам, які легко та вільно повертаються і нахиляються відповідно до потреби працівника.

9. За необхідності може використовуватись окрема підставка або регульований стіл для розміщення екрана.

10. Екран не має відблискувати або відбивати світло, щоб не викликати дискомфорту у працівника під час роботи з екранними пристроями.

11. Вибираючи клавіатуру, слід надавати перевагу такій клавіатурі, яка відкидається і є автономною (відокремленою від екрана), щоб працівник міг вибрати зручну робочу позу й уникнути втоми рук (кисті і верхньої частини руки).

12. Під час розробки, вибору, замовлення та модифікації програмного забезпечення, а також під час розробки завдань, що передбачають використання устаткування з екранними пристроями, роботодавець має керуватися таким програмним забезпеченням, яке відповідає розв’язуваним завданням і є простим у використанні, а де необхідно - адаптованим до рівня знань і досвіду працівника.

З працівниками проводитимуться “Інструктаж з поведінки в разі виникнення аварійної ситуації” кожні півроку, або позапланове у разі виникнення потреби.

Тому, кожен співробітник знає:

1. Ознаки аварійної ситуації на робочому місці [15]:

- поява збоїв у роботі ПК, зникнення зображення на екрані монітора;
- коротке замикання, іскріння, появи запаху горіння, підвищене нагрівання корпусу, штепсельних рознімань, сполучних проводів, зниження або зникнення напруги в мережі.

2. В аварійній ситуації необхідно:

- роботу припинити, відключити від мережі комп’ютери та оргтехніку;
- при загорянні використовувати вуглекислотний або порошковий вогнегасники;

- вжити заходів по евакуації людей і наданню першої медичної допомоги постраждалим;
- доповісти про те, що трапилося, керівникові;
- при необхідності викликати швидку допомогу, пожежну команду.

3. Якщо виникла ситуація, що може призвести до аварії або нещасного випадку необхідно огородити небезпечну зону і не запускати в неї сторонніх осіб. Якщо є потерпілі надавати їм першу домедичну допомогу, при необхідності викликати швидку допомогу.

Дотримання цих вимог та проведення регулярних інструктажів дозволить забезпечити безпечні умови праці для працівників, залучених до створення та підтримки блогу "Цифровий світ". Таким чином, систематичний підхід до охорони праці не лише сприяє збереженню здоров’я персоналу, а й підвищує ефективність виконання поставлених завдань у процесі розробки.

4.2 Забезпечення безпеки життєдіяльності при роботі з ПК

Під час виконання кваліфікаційної роботи магістра було розроблено масштабований блог «Цифровий світ». Для реалізації використано мікросервісну архітектуру та розбито весь функціонал на автономні частини, які виконують конкретні задачі. При розробці блогу використовується персональний комп’ютер, тому потрібно знати, як забезпечити безпеку життєдіяльності при роботі з ним.

Безпека роботи з персональним комп’ютером забезпечується дотриманням певних правил та вимог, що включають правильну організацію робочого місця, врахування фізіологічних аспектів користувача, а також належне поводження з обладнанням. До роботи допускаються лише особи, які пройшли необхідне навчання, інструктажі та медичний огляд. У подальшому вони регулярно проходять повторні інструктажі та періодичні медичні огляди, що є важливим для запобігання професійним захворюванням.

Робота з комп’ютером пов’язана з впливом фізичних та психофізіологічних факторів, таких як тривала статична напруга, вплив електромагнітного випромінювання та можливість виникнення стресу. Основними складовими робочого місця є монітор, системний блок і клавіатура. Робоче місце повинно бути облаштоване з урахуванням ергономіки, забезпечуючи оптимальне розташування обладнання, щоб уникнути перенапруги зору, м’язів та суглобів [16]. Монітор слід встановлювати на відстані 400–700 мм від очей, з нахилом екрана, що забезпечує зручний кут огляду. Правильне розташування обладнання та використання антивідблискових засобів, таких як жалюзі чи спеціальні фільтри для моніторів, дозволяють уникнути відблисків та надмірної яскравості, що сприяє збереженню зору користувача.

Приміщення для роботи з комп’ютерами має бути обладнане системою вентиляції та кондиціювання для підтримання оптимальної температури й вологості. Рекомендовано уникати використання синтетичних матеріалів в одязі, що накопичують статичну електрику, а також забезпечувати регулярне прибирання пилу на обладнанні для зменшення електростатичних зарядів.

Важливим аспектом безпеки є правильна постава під час роботи. Зручна поза забезпечується налаштуванням висоти стільця, підставки для ніг та столу. Рекомендоване положення передбачає горизонтальне розташування стегон, вертикальне розташування верхніх частин рук, та кут згину ліктьового суглоба в межах 70–90°. Для уникнення перенапруги очей та м’язів слід дотримуватися перерв у роботі.

Дотримання техніки безпеки передбачає перевірку стану обладнання, його правильного підключення та налаштування параметрів роботи. Забороняється самостійно проводити ремонт чи втручатися в апаратуру без відповідної підготовки. Усі несправності повинні негайно повідомлятися керівнику. Також важливо регулярно очищувати поверхні обладнання відповідними засобами, уникаючи рідин, що можуть пошкодити апаратуру.

Наприкінці роботи слід завершити всі активні процеси, вимкнути пристрої, відключити їх від мережі живлення та організувати робоче місце. Рекомендується

виконувати вправи для зняття втоми та розслаблення, що зменшують негативний вплив статичних поз та навантажень.

Доречно зауважити, що при дослідженні питання про безпеку працівників при використанні персональних комп’ютерів використано навчальний посібник «Техноекологія та цивільна безпека. Частина «Цивільна безпека»» [17] та методичний посібник для здобувачів освітнього ступеня «магістр» «Безпека в надзвичайних ситуація» [18].

Таким чином, дотримання наведених вимог та рекомендацій забезпечує безпечну роботу з персональним комп’ютером, мінімізує ризики травм чи професійних захворювань, а також сприяє підвищенню продуктивності праці.

ВИСНОВКИ

В процесі виконання кваліфікаційної роботи магістра за темою "Аналіз та впровадження мікросервісної архітектури для створення масштабованого блогу Цифровий світ" було досягнуто цілей, які були поставлені на етапі аналізу предметної області. Результатом роботи стало створення масштабованого блогу на основі мікросервісної архітектури.

Проаналізовано мікросервісну архітектуру, її переваги над монолітним підходом. Також визначено функціональні та нефункціональну вимоги системи. Проведено аналіз головних українських конкурентів, які публікують контент пов’язаний з інформаційними технологіями.

На етапі проєктування спроектовано діаграми використання, які відображають доступний функціонал для відвідувача, авторизованого користувача та адміністратора. Спроектовано діаграму класів та описано всі атрибути та методи. Також складено діаграми послідовності, які описують основні сценарії роботи блогу.

За допомогою фреймворку NestJS розроблено незалежні сервіси: авторизації та управління користувачами, публікації, коментування, пошук та фільтрація, аналітика. Перевірено ключові сценарії взаємодії користувача з блогом: авторизацію, створення, редагування та видалення публікацій, додавання коментарів, роботу з тегами. Система може бути розширена за рахунок добавлення нових мікросервісів для рекомендацій, розширеної аналітики, чи інтеграції з іншими платформами.

У розділі охорона праці та безпека в надзвичайних ситуаціях кваліфікаційної роботи магістра зосереджено увагу на забезпеченні безпечних умов праці для працівників, які залучені до розробки та підтримки блогу "Цифровий світ", відповідно до вимог охорони праці та техніки безпеки. Організація робочих місць працівників з екранними пристроями передбачає дотримання стандартів щодо ергономіки, освітлення, якості зображення та адаптивності обладнання до

індивідуальних потреб співробітників. Виконання зазначених рекомендацій дозволяє створити комфортні та безпечні умови для роботи з комп’ютерами, що позитивно впливає на здоров’я працівників і підвищує їхню продуктивність.

Отже, виконана кваліфікаційна робота підтверджує актуальність використання мікросервісної архітектури для створення масштабованих веб-додатків. Результати роботи можуть бути використані як основа для подальшого розвитку блогу "Цифровий світ". У майбутньому для блогу "Цифровий світ" планується впровадження сучасних систем, заснованих на алгоритмах штучного інтелекту. Це дозволить значно підвищити ефективність аналізу даних. Зокрема, основний акцент буде зроблено на автоматизації процесів оцінки популярності постів, прогнозування змін трафіку та аналізу рівня взаємодії користувачів із контентом. Також штучний інтелект можна буде інтегрувати для автоматичного створення звітів, модерації коментарів, а в подальшому — навіть для персоналізації контенту для кожного користувача. Це значно покращить досвід взаємодії читачів із блогом і дозволить зберігати конкурентоспроможність на динамічному ринку інформаційних платформ.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Що таке мікросервісна архітектура: значення, складові, переваги [Електронний ресурс] – Режим доступу до ресурсу: https://wezom.com.ua/ua/blog/scho-take-mikroservisna-arhitektura-znachennya-skladovi-perevagi.
2. Про архітектуру додатків [Електронний ресурс] – Режим доступу до ресурсу: https://foxminded.ua/arkhitektura-zastosunku/.
3. Мікросервісна архітектура [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@IvanZmerzlyi/microservices-architecture-461687045b3d.
4. Мікросервісна архітектура: плюси та мінуси [Електронний ресурс] – Режим доступу до ресурсу: https://itedu.center/ua/blog/articles/microservices-architecture-advantages-and-disadventages/?srsltid=AfmBOooHCpe49Fk_HCB3JvFhNhnkFY6bdAUyCM34vRmpp_0QJNIEQSY0
5. Stefanyshyn, V. , Stefanyshyn, I. , Pastukh, O. , Yatsyshyn, V. , Yakymenko. Accuracy of software and hardware of computer systems for human-machine interaction, I. CEUR Workshop Proceedings, 2024, 3842, pp. 178–183.
6. Stefanyshyn, I. , Pastukh, O., Stefanyshyn, V. , Baran, I. , Boyko, I. Robustness of AI algorithms for neurocomputer interfaces based on software and hardware technologies CEUR Workshop Proceedings, 2024, 3742, pp. 137–149.
7. Documentation | NestJS [Електронний ресурс] – Режим доступу до ресурсу: https://docs.nestjs.com/.
8. MongoDB Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/docs/.
9. Getting Started [Електронний ресурс] – Режим доступу до ресурсу: https://mongoosejs.com/docs/.

10. React Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://react.dev/.

11. TypeScript Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.typescriptlang.org/docs/.

12. ТОП-11 цікавих українських сайтів та блогів про IT-галузь [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://cityhost.ua/uk/blog/top-cikavih-ukra-nskih-saytiv-ta-blogiv-pro-it.html.

13. Про затвердження Вимог щодо безпеки та захисту здоров\'я працівників під час роботи з екранними пристроями [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0508-18#Text.

14. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/v0007282-98#Text.

15. ІНСТРУКЦІЯ З ОХОРОНИ ПРАЦІ ДЛЯ ОФІСНИХ ПРАЦІВНИКІВ [Електронний ресурс] – Режим доступу до ресурсу: https://safety.org.ua/instruktsiya-z-ohorony-pratsi-dlya-ofisnyh-pratsivnykiv/.

16. ДСТУ EN 894-1:2017. Ергономічні вимоги до проектування робочих місць.

17. Навчальний посібник «ТЕХНОЕКОЛОГІЯ ТА ЦИВІЛЬНА БЕЗПЕКА. ЧАСТИНА «ЦИВІЛЬНА БЕЗПЕКА»» / автор-укладач В.С. Стручок– Тернопіль: ФОП Паляниця В. А., – 156 с. Отримано з http://elartu.tntu.edu.ua/handle/lib/39424

18. Методичний посібник для здобувачів освітнього ступеня «магістр» всіх спеціальностей денної та заочної (дистанційної) форм навчання «БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ» / В.С. Стручок –Тернопіль: ФОП Паляниця В. А., –156 с. Отримано з https://elartu.tntu.edu.ua/handle/lib/39196.

ДОДАТКИ

ДОДАТОК А Лістинг коду

Лістинг коду 1 – Схема користувача

```
export type UserDocument = User & Document;
@Schema()
export class User {
  @Prop({ required: true, unique: true })
  email: string;
  @Prop({ required: true })
  password: string;
  @Prop({ required: true })
  name: string;
  @Prop({ required: false })
  role: string;
}
export const UserSchema = SchemaFactory.createForClass(User);
```

Лістинг коду 2 – Контролер мікросервісу авторизації

```
@Controller()
export class AppController {
  constructor(private readonly appService: AppService){}

  @MessagePattern('register')
  async register(data: { email: string; password: string; name: string, role:
string }) {
    const { email, password, name, role } = data;
    return this.appService.register(email, password, name, role);
  }

  @MessagePattern('login')
  async login(data: {email: string; password: string}){
    const { email, password} = data;
    return this.appService.login(email, password);
  }

  @MessagePattern('getById')
  async getById(data: {id:string}){
    const {id} = data;
    return this.appService.getById(id);
  }
}
```

Лістинг коду 3 – Бізнес-логіка мікросервісу авторизації

```
@Injectable()
export class AppService {
  constructor(@InjectModel(User.name)          private          userModel:
Model<UserDocument>) {}

  async register(
    email: string,
    password: string,
    name: string,
    role: string,
  ): Promise<User> {
    // Перевірка, чи вже існує користувач
    const existingUser = await this.userModel.findOne({ email });
    if (existingUser) {
      throw new Error('User already exists');
    }

    // Створення нового користувача
    const newUser = new this.userModel({ email, password, name, role });
    console.log(newUser);
    return newUser.save();
  }

  async login(email: string, password: string): Promise<User> {
    const user = await this.userModel.findOne({ email });
    if (!user) {
      throw new Error('User not found!');
    }
    if (password === user.password) {
      console.log("login",user);
      return user;
    } else throw new Error('Wrong password or email!');
  }

  async getById(id: string): Promise<User> {
    const user = await this.userModel.findById(id);
    console.log('ID', id);
    if(user) {
      console.log(user);
      return user;
    }
      else throw new Error('User not found!')
  }
}
```

Лістинг коду 4 – Налаштування модуля мікросервісу авторизації

```
@Module({
 imports:
[MongooseModule.forRoot('mongodb://localhost:27017/auth_service'),
    MongooseModule.forFeature([{ name: User.name, schema: UserSchema }]),],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}
```

Лістинг коду 5 – Налаштування брокера повідомлень для мікросервісу авторизації

```
async function bootstrap() {
                const                app                =                await
NestFactory.createMicroservice<MicroserviceOptions>(AppModule, {
    transport: Transport.RMQ,
    options: {
      urls: ['amqp://localhost:5672'],
      queue: 'auth_queue',
      queueOptions: {
        durable: false,
      },
    },
  });
  await app.listen();
  console.log('Auth service is running');
}
bootstrap();
```

Лістинг коду 6 – Схема коментаря

```
@Schema()
export class Comment extends Document {
  @Prop({ required: true })
  postId: string;
  @Prop({ required: true })
  userId: string;
  @Prop({ required: true })
  text: string;
  @Prop({ default: Date.now })
  createdAt: Date;
}
export const CommentSchema = SchemaFactory.createForClass(Comment);
```

Лістинг коду 7 – Контролер мікросервісу коментування

```
@Controller()
```

```
export class CommentController {
  constructor(private readonly commentService: CommentService) {}

  @MessagePattern('create_comment')
  createComment(data: { postId: string; userId: string; text: string }) {
      return this.commentService.createComment(data.postId, data.userId,
data.text);
  }

  @MessagePattern('get_comments')
  getComments(data: { postId: string }) {
    return this.commentService.getCommentsByPost(data.postId);
  }

  @MessagePattern('update_comment')
  updateComment(data: { id: string; text: string }) {
    return this.commentService.updateComment(data.id, data.text);
  }
     }
```

Лістинг коду 8 – Бізнес-логіка мікросервісу коментування

```
@Injectable()
export class CommentService {
      constructor(@InjectModel(Comment.name) private commentModel:
Model<Comment>) {}
   async createComment(postId: string, userId: string, text: string):
Promise<Comment> {
    const newComment = new this.commentModel({ postId, userId, text });
    return newComment.save();
  }
  async getCommentsByPost(postId: string): Promise<Comment[]> {
    return this.commentModel.find({ postId }).exec();
  }
  async updateComment(id: string, text: string): Promise<Comment> {
     return this.commentModel.findByIdAndUpdate(id, { text }, { new: true
}).exec();
  }
}
```

Лістинг коду 9 – Модуль мікросервісу коментування

```
      @Module({
                                                              imports:
[MongooseModule.forRoot('mongodb://localhost:27017/comment_service'),
             MongooseModule.forFeature([{ name: Comment.name, schema:
CommentSchema }])
        ],
        controllers: [CommentController],
        providers: [CommentService],
```

```
      })
      export class AppModule {}
```

Лістинг коду 10 – Налаштування брокера повідомлень для мікросервісу коментування

```
async function bootstrap() {
                const                app                =                await
NestFactory.createMicroservice<MicroserviceOptions>(AppModule, {
    transport: Transport.RMQ,
    options: {
      urls: ['amqp://localhost:5672'],
      queue: 'comment_queue',
      queueOptions: {
        durable: false,
      },
    },
  });
  await app.listen();
  console.log('Comment service is running');
}
bootstrap();
```

Лістинг коду 11 – Схема типізації посту

```
@Schema()
export class Post extends Document {
  @Prop({ required: true })
  title: string;
  @Prop({ type: [String], default: [] })
  tags: string[];
  @Prop({ required: true })
  authorId: string;
  @Prop({ required: true })
  content: string;
  @Prop()
  photoUrl: string;
}

export const PostSchema = SchemaFactory.createForClass(Post);
```

Лістинг коду 12 – Контролер мікросервісу публікації

```
      @Controller('posts')
      export class PostController {
        constructor(private readonly postService: PostService) {}
        @MessagePattern('createPost')
         async createPost(data: { title: string; tags: string[]; authorId:
string; content: string; photoUrl: string }) {
           return this.postService.createPost(data);
```

```
    }

    @MessagePattern('updatePost')
    async updatePost(data: { id: string; title?: string; tags?: string[];
content?: string; photoUrl?: string }) {
      const { id, ...updateData } = data;
      return this.postService.updatePost(id, updateData);
    }
    @MessagePattern('deletePost')
    async deletePost(data: { id: string }) {
      return this.postService.deletePost(data.id);
    }
  }
```

Лістинг коду 13 – Бізнес-логіка мікросервісу публікації

```
@Injectable()
export class PostService {
  constructor(@InjectModel(Post.name) private postModel: Model<Post>) {}

  async createPost(data: Partial<Post>): Promise<Post> {
    return new this.postModel(data).save();
  }

  async updatePost(id: string, data: Partial<Post>): Promise<Post> {
    return this.postModel.findByIdAndUpdate(id, data, { new: true });
  }

  async deletePost(id: string): Promise<Post> {
    return this.postModel.findByIdAndDelete(id);
  }
}
```

Лістинг коду 14 – Модуль мікросервісу публікації

```
@Module({
                                                          imports:
[MongooseModule.forRoot('mongodb://localhost:27017/post_service'),
    MongooseModule.forFeature([{ name: Post.name, schema: PostSchema }]) ],
  controllers: [PostController],
  providers: [PostService],
})
export class AppModule {}
```

Лістинг коду 15 – Налаштування брокера повідомлень для мікросервісу публікації

```
async function bootstrap() {
                const                app                =                await
NestFactory.createMicroservice<MicroserviceOptions>(AppModule, {
```

```
    transport: Transport.RMQ,
    options: {
      urls: ['amqp://localhost:5672'],
      queue: 'post_queue',
      queueOptions: {
        durable: false,
      },
    },
  });
  await app.listen();
  console.log('Post service is running');
}
bootstrap();
```

Лістинг коду 16 – Контролер мікросервісу пошуку та фільтрації

```
@Controller()
export class SearchController {
  constructor(private readonly searchService: SearchService) {}

  @MessagePattern({ pattern: 'search_posts' })
  async searchPosts(data: { keyword: string }) {
    return this.searchService.searchByKeyword(data.keyword);
  }

  @MessagePattern({ pattern: 'filter_posts' })
  async filterPosts(data: { tag: string }) {
    return this.searchService.filterByTag(data.tag);
  }
}
```

Лістинг коду 17 – Бізнес-логіка для мікросервісу пошуку та фільтрації

```
Injectable()
export class SearchService {
  constructor(@InjectModel('Post') private readonly postModel: Model<Post>)
{
  async searchByKeyword(keyword: string): Promise<Post[]> {
    return this.postModel
      .find({
        $or: [
          { title: { $regex: keyword, $options: 'i' } }, // Пошук у назві
          { text: { $regex: keyword, $options: 'i' } },  // Пошук у тексті
        ],
      })
      .exec();
  }
  async filterByTag(tag: string): Promise<Post[]> {
    return this.postModel
      .find({
```

```
        tags: tag, // Пошук за тегом
      })
      .exec();
  }
}
```

Лістинг коду 18 – Налаштування брокеру повідомлень для мікросервісу пошуку та фільтрації

```
async function bootstrap() {
                const                app                =                await
NestFactory.createMicroservice<MicroserviceOptions>(AppModule, {
    transport: Transport.RMQ,
    options: {
      urls: ['amqp://localhost:5672'], // URL RabbitMQ
      queue: 'search_queue', // Черга для мікросервісу
      queueOptions: {
        durable: false,
      },
    },
  });

  await app.listen();
  console.log('Search microservice is listening...');
}
bootstrap();
```

Лістинг коду 19 – Компонент AddComment

```
export const Index = ({ setData }) => {
  const userData = useSelector((state) => state.auth.data);
  const { id } = useParams();

  const [textComment, setTextComment] = useState('');

  const onSendComment = async () => {
    try {
      const { data } = await axios.patch(`/posts/${id}/comments`, {
        textComment,
      });
      setData((prevData) => ({
        ...prevData,
        comments: [...prevData.comments, data],
      }));
      setTextComment('');
    } catch (err) {
      console.warn(err);
      alert('Помилка при публікації коментара');
    }
```

```
  };

  return (
    <>
      <div className={styles.root}>
                <Avatar   classes={{   root:   styles.avatar   }}   src={
`${process.env.REACT_APP_API_URL}${userData?.avatarUrl}`  } />
        <div className={styles.form}>
          <TextField
            onChange={(e) => setTextComment(e.target.value)}
            label="Написати коментра"
            variant="outlined"
            maxRows={10}
            value={textComment}
            multiline
            fullWidth
          />
          <Button onClick={onSendComment} variant="contained">
            Надіслати
          </Button>
        </div>
      </div>
    </>
  );
};
```

Лістинг коду 20 – Компонент Post

```
export const Post =
({id,title,createdAt,imageUrl,user,viewsCount,commentsCount,tags,children,i
sFullPost,
  isLoading,
  isEditable,
}) => {
  const dispatch = useDispatch();

  if (isLoading) {
    return <PostSkeleton />;
  }

  const onClickRemove = () => {
    if (window.confirm('Ви впевнені, що хочете видалити статю?')) {
      dispatch(fetchDeletePost(id));
    }
  };

  return (
    <div className={clsx(styles.root, { [styles.rootFull]: isFullPost })}>
      {isEditable && (
        <div className={styles.editButtons}>
```

```
          <Link to={`/posts/${id}/edit`}>
            <IconButton color="primary">
              <EditIcon />
            </IconButton>
          </Link>
          <IconButton onClick={onClickRemove} color="secondary">
            <DeleteIcon />
          </IconButton>
        </div>
      )}
      {imageUrl && (
        <img
          className={clsx(styles.image, { [styles.imageFull]: isFullPost })}
          src={imageUrl}
          alt={title}
        />
      )}
      <div className={styles.wrapper}>
        <UserInfo {...user} additionalText={createdAt} />
        <div className={styles.indention}>
          <h2
             className={clsx(styles.title, { [styles.titleFull]: isFullPost
})}
          >
            {isFullPost ? title : <Link to={`/posts/${id}`}>{title}</Link>}
          </h2>
          <ul className={styles.tags}>
            {tags.map((name) => (
              <li key={name}>
                <Link to={`/tags/${name}`}>#{name}</Link>
              </li>
            ))}
          </ul>
          {children && <div className={styles.content}>{children}</div>}
          <ul className={styles.postDetails}>
            <li>
              <EyeIcon />
              <span>{viewsCount}</span>
            </li>
            <li>
              <CommentIcon />
              <span>{commentsCount}</span>
            </li>
          </ul>
        </div>
      </div>
    </div>
  );
};
```

Лістинг коду 21 – Слайс для авторизації

```
export const fetchAuth = createAsyncThunk(
  'auth/fetchAuth',
  async (params) => {
    const {data} = await axios.post('/auth/login', params);
    return data;
  }
);

export const fetchRegister = createAsyncThunk(
  'auth/fetchRegister',
  async (params) => {
    const {data} = await axios.post('/auth/register', params);
    return data;
  }
);

export const fetchAuthMe = createAsyncThunk('/auth/fetchAuthMe', async()=>{
  const {data} = await axios.get('/auth/me');
  return data;
});

const initialState = {
  data: null,
  status: 'loading',
};

const authSlice = createSlice({
  name: 'auth',
  initialState,
  reducers:{
    logout: (state)=>{
      state.data = null;
    }
  },
  extraReducers: {
    [fetchAuth.pending]: (state) => {
      state.status = 'loading';
      state.data = null;
    },
    [fetchAuth.fulfilled]: (state, action) => {
        state.status = 'loaded';
        state.data = action.payload;
    },
    [fetchAuth.rejected]: (state) => {
        state.status = 'error';
        state.data = null;
    },
    [fetchAuthMe.pending]: (state) => {
      state.status = 'loading';
```

```
      state.data = null;
    },
    [fetchAuthMe.fulfilled]: (state, action) => {
        state.status = 'loaded';
        state.data = action.payload;
    },
    [fetchAuthMe.rejected]: (state) => {
        state.status = 'error';
        state.data = null;
    },
    [fetchRegister.pending]: (state) => {
      state.status = 'loading';
      state.data = null;
    },
    [fetchRegister.fulfilled]: (state, action) => {
        state.status = 'loaded';
        state.data = action.payload;
    },
    [fetchRegister.rejected]: (state) => {
        state.status = 'error';
        state.data = null;
    },
  },
});

export const selectIsAuth = (state)=> Boolean(state.auth.data);

export const authReducer = authSlice.reducer;

export const {logout} = authSlice.actions;
```

Лістинг коду 22 – Слайс для постів

```
export const fetchPosts = createAsyncThunk('posts/fetchPosts', async
() => {
  const { data } = await axios.get('/posts');
  return data;
});

export const fetchTags = createAsyncThunk('posts/fetchTags', async ()
=> {
  const { data } = await axios.get('/tags');
  return data;
});

export const fetchDeletePost =
createAsyncThunk('/posts/fetchDeletePost', async(id)=>{
  axios.delete(`/posts/${id}`)
})
```

```
export const fetchPostByTag = createAsyncThunk('/posts/fetchPostByTag', async(tagName)=>{
  const {data} = await axios.get(`/tags/${tagName}`);
  return data;
})


const initialState = {
  posts: {
    items: [],
    sortType:0,
    status: 'loading',
  },
  tags: {
    items: [],
    status: 'loading',
  },
};

const postsSlice = createSlice({
  name: 'posts',
  initialState,
  reducers: {
    sortPosts: (state, action) =>{
      switch (action.payload) {
        case 'popular':
          state.posts.items.sort(byField('viewsCount'));
          state.posts.sortType=1;
          break;
          case 'new':
            state.posts.items.sort(byData())
          state.posts.sortType=0;
            break;
        default:
          break;
      }

    },
  },
  extraReducers: {
    // Отримання статей
    [fetchPosts.pending]: (state) => {
      state.posts.items = [];
      state.posts.status = 'loading';
    },
    [fetchPosts.fulfilled]: (state, action) => {
      state.posts.items = action.payload;
      state.posts.status = 'loaded';
      state.posts.items.sort(byData())
    },
```

```
        [fetchPosts.rejected]: (state) => {
          state.posts.items = [];
          state.posts.status = 'error';
        },
        // Отримання тегів
        [fetchTags.pending]: (state) => {
          state.tags.items = [];
          state.tags.status = 'loading';
        },
        [fetchTags.fulfilled]: (state, action) => {
          state.tags.items = action.payload;
          state.tags.status = 'loaded';
        },
        [fetchTags.rejected]: (state) => {
          state.tags.items = [];
          state.tags.status = 'error';
        },
        //Видалення статі
        [fetchDeletePost.pending]: (state, action)=>{
           state.posts.items = state.posts.items.filter(obj=>obj._id !==
action.meta.arg);
        },
        //Отримання статетй за тегом
        [fetchPostByTag.pending]: (state) => {
          state.posts.items = [];
          state.posts.status = 'loading';
        },
        [fetchPostByTag.fulfilled]: (state, action) => {
          state.posts.items = action.payload;
          state.posts.status = 'loaded';
          state.posts.items.sort(byData())
        },
        [fetchPostByTag.rejected]: (state) => {
          state.posts.items = [];
          state.posts.status = 'error';
        },
      },
    });

    export const postReducer = postsSlice.reducer;
    export const {sortPosts} = postsSlice.actions;
```

Лістинг коду 23 – Повна сторінка посту

```
export const FullPost = () => {
  const [data, setData] = useState();
  const [isLoading, setIsLoading] = useState(true);
  const { id } = useParams();
  useEffect( () => {
    axios
```

```
      .get(`/posts/${id}`)
      .then((res) => {
        setData(res.data);
        setIsLoading(false);
      })
      .catch((err) => {
        console.warn(err);
        alert('Помилка при отриманні статті');
      });

    // eslint-disable-next-line react-hooks/exhaustive-deps
  }, []);

  if (isLoading) {
    return <Post isLoading={isLoading} isFullPost />;
  }

  return (
    <>
      <Post
        id={data._id}
        title={data.title}
                                          imageUrl={data.imageUrl ? `${process.env.REACT_APP_API_URL}${data.imageUrl}` : ''}
        user={data.user}
        createdAt={data.createdAt}
        viewsCount={data.viewsCount}
        commentsCount={data.commentsCount}
        tags={data.tags}
        isFullPost
      >
        <ReactMarkdown children={data.text} />
      </Post>
      <CommentsBlock  items={data.comments}>
        <Index setData={setData} />
      </CommentsBlock>
    </>
  );
};
```

ДОДАТОК Б Тези конференції

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
імені ІВАНА ПУЛЮЯ
НАУКОВЕ ТОВАРИСТВО ім. ШЕВЧЕНКА

М А Т Е Р І А Л И

XII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

18-19 грудня 2024 року

ТЕРНОПІЛЬ
2024

УДК 004.41

В. Москалик; Ю. Стоянов, к.т.н. доц., старший викладач
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

РОЛЬ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ У ЗАБЕЗПЕЧЕННІ МАСШТАБОВАНОСТІ ВЕБ-ДОДАТКІВ

UDC 004.41

V. Moskalyk; Y. Stoyanov, Ph.D.

THE ROLE OF MICROSRVICES ARCHITECTURE IN ENSURING SCALABILITY OF WEB APPLICATIONS

Сучасні веб-додатки повинні швидко обробляти великі обсяги даних, забезпечувати високу продуктивність та надійність навіть за умови пікових навантажень. Мікросервісна архітектура часто стає ключовим варіантом для проектування великих та складних додатків.

Однією з ключових переваг мікросервісної архітектури є її здатність забезпечувати масштабованість окремих компонентів системи. На відміну від монолітних додатків, де масштабування можливе лише для всієї програми, у мікросервісах можна масштабувати лише ті сервіси, які цього потребують. Наприклад у веб-додатку із сервісами авторизації, публікації контенту та аналітики під час пікових навантажень можна збільшити потужності лише для сервісу аналітики [1].

Оскільки мікросервіси є автономними, відмова одного компонента не впливає на роботу інших. Це забезпечує високу надійність системи, бо ключові функції, такі як авторизація чи перегляд контенту, продовжують працювати навіть під час збоїв в інших мікросервісах додатку.

Ще важливою перевагою даної архітектури є можливість використовувати різні технології для реалізації окремих сервісів. Для сервісу аналітики можна використати мову програмування Python через її зручність у роботі з даними, тоді як для реалізації мікросервісів, які потребують високої продуктивності та стабільності, можуть використовуватись C# або Java. Така технологічна незалежність дозволяє оптимізувати кожен компонент системи під конкретні завдання. Також, завдяки автономності в системі можуть впроваджуватися інструменти, які набирають популярності з швидким розвитком інформаційних технологій.

Попри суттєві переваги мікросервісна архітектура має недоліки. Одним з яких є складніст тестування та налагодження. При великій кількості сервісів, коли виникає помилка в системі, т пошук причини вимагає велику кількість часу та потребує досвідчених спеціалістів [2].

Хоча цей підхід має свої виклики, його переваги, включно зі зменшенням ризику збоїв т економією ресурсів при масштабуванні, роблять мікросервіси перспективним вибором для веб додатків.

Література

1. Andersen G. The Role of Microservices Architecture in Web Development [Електронний ресурс] Grady Andersen. – 2024. – Режим доступу до ресурсу: https://moldstud.com/articles/p-the-role-of microservices-architecture-in-web-development.
2. Архітектура мікросервісів: Особливості, переваги, реальні приклади [Електронний ресурс]. 2024. – Режим доступу до ресурсу: https://www.hostzealot.com.ua/blog/about-solutions arxitektura-mikroservisiv-osoblivosti-perevagi-realni-prikladi.

ДОДАТОК В Диск із кваліфікаційною роботою