**ДИПЛОМНА РОБОТА**

**ДР.122.042.007.ПЗ**

**НОВАКОВСЬКИЙ АНДРІЙ ОЛЕКСАНДРОВИЧ**

**2025**

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЖИТОМИРСЬКИЙ АГРОТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ

ВІДДІЛЕННЯ «Інженерна інфраструктура та комп'ютерні науки»

ЦИКЛОВА КОМІСІЯ «Інженерна інфраструктура та комп'ютерні науки»

**ПОЯСНЮВАЛЬНА ЗАПИСКА**

до дипломної роботи освітнього ступеня «фаховий молодший бакалавр»

за спеціальністю 122 Комп’ютерні науки

(освітня програма 122 «Комп’ютерні науки»)

на тему:

**«Розробка веб-сайту блогу з інтеграцією  
SEO оптимізації та аналітики»**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Студента | IV | | курсу | | | П-42 | групи | | | |
| Новаковського Андрія Олександровича | | | | | | | | | |
| (ПІБ) | | | | | | | | | |
| Керівник | | Габрійчук Наталя Ігорівна | | | | | | | |
| Рецензент | | Можаровський Сергій Володимирович | | | | | | | |
|  | | | | | | | | | |
| Національна шкала | | | | |  | | | | |
| Кількість балів | | | |  | | | Оцінка: ECTS | |  |
|  | | | |  | | |  | |  |
| Члени комісії | | | |  | | | | Ісаєв А.М. | |
|  | | | | (підпис) | | | | (прізвище та ініціали) | |
|  | | | |  | | | | Габрійчук Н.І. | |
|  | | | | (підпис) | | | | (прізвище та ініціали) | |
|  | | | |  | | | | Устименко Л.М. | |
|  | | | | (підпис) | | | | (прізвище та ініціали)  Устименко Я.І. | |
|  | | | | (підпис) | | | | (прізвище та ініціали) | |
|  | | | |

Житомир – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЖИТОМИРСЬКИЙ АГРОТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ

ВІДДІЛЕННЯ «Інженерна інфраструктура та комп'ютерні науки»

ЦИКЛОВА КОМІСІЯ «Інженерна інфраструктура та комп'ютерні науки»

**«ЗАТВЕРДЖУЮ»**

Голова циклової комісії «Інженерна

інфраструктура та комп'ютерні науки»

Діана ПАЛІЙ

«06» листопада 2024 р.

**ЗАВДАННЯ**

**на дипломну роботу**

Здобувач вищої освіти: **НОВАКОВСЬКИЙ Андрій Олександрович**

Керівник роботи: **гАБРІЙЧУК Наталя Ігорівна**

Тема роботи: **«Розробка веб-сайту блогу з інтеграцією SEO оптимізації та аналітики»**, затверджена наказом закладу вищої освіти від «03» грудня 2024 р., №496н.

Вихідні дані для роботи: Створити повнофункціональний веб-сайт блогу з інтегрованою системою управління контентом, SEO оптимізацією та веб-аналітикою, який забезпечує ефективне створення, редагування та просування контенту, а також надає зручні інструменти для взаємодії між авторами та читачами.

|  |  |  |  |
| --- | --- | --- | --- |
| Розділ | Консультант | Завдання  видав | Завдання прийняв |
| 1 | Наталя ГАБРІЙЧУК | 09.12.2024 | 19.02.2025 |
| 2 | Наталя ГАБРІЙЧУК | 19.02.2025 | 02.04.2025 |
| 3 | Наталя ГАБРІЙЧУК | 02.04.2025 | 26.05.2025 |

Консультанти з дипломної роботи із зазначенням розділів, що їх стосується:

**Календарний план**

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Етап роботи | Термін виконання | Примітка |
| 1 | Формулювання мети та цілей роботи, визначення завдань розробки | 9 грудня 2024 | Виконано |
| 2 | Планування розробки | 20 січня 2025 | Виконано |
| 3 | Проєктування користувацької частини сайту | 19 лютого 2025 | Виконано |
| 4 | Проєктування дизайну проекту | 5 березня 2025 | Виконано |
| 5 | Розробка алгоритмів | 20 березня 2025 | Виконано |
| 6 | Перевірка роботи алгоритмів | 27 березня 2025 | Виконано |
| 7 | Проектування функцій проекту | 2 квітня 2025 | Виконано |
| 8 | Налагодження взаємодії між елементами | 14 квітня 2025 | Виконано |
| 9 | Розробка дизайну для різних пристроїв | 25 квітня 2025 | Виконано |
| 10 | Тестування всіх функцій сайту | 15 травня 2025 | Виконано |
| 11 | Систематизація результатів розробки, написання та оформлення ПЗ | 26 травня 2025 | Виконано |
| 12 | Попередній захист роботи | 13 червня 2025 | Виконано |

Здобувач фахової передвищої освіти\_\_\_\_\_\_\_\_\_\_ Андрій НОВАКОВСЬКИЙ

Керівник \_\_\_\_\_\_\_\_\_\_ Наталя ГАБРІЙЧУК

РЕФЕРАТ

Записка: 68 стoр., 17 рис., 10 джерел.

ВЕБ-САЙТ, БЛОГ, SEO ОПТИМІЗАЦІЯ, АНАЛІТИКА, REACT, EXPRESS.JS, MONGODB, TAILWIND CSS, КОНТЕНТ-МЕНЕДЖМЕНТ

Об'єкт дослідження — процеси створення та управління контентом веб-блогу, інтеграція SEO оптимізації та веб-аналітики у сучасних веб-додатках.

Мета роботи — створення повнофункціонального веб-сайту блогу з інтегрованою системою управління контентом, SEO оптимізацією та аналітикою для ефективного управління та просування контенту.

Методи дослідження — методи аналізу сучасних технологій веб-розробки, дослідження підходів до SEO оптимізації, методи проектування клієнт-серверної архітектури.

Результати — проведено аналіз існуючих блог-платформ та SEO інструментів, здійснено вибір технологічного стеку. Виконано програмну реалізацію веб-сайту блогу з функціоналом створення та публікації статей, системою аутентифікації користувачів, інтеграцією PostHog та SEO оптимізацією з динамічним генеруванням мета-тегів.

ABSTRACT

The bachelor work consists of the source code of a blog website with integrated SEO optimization and analytics, along with documentation. The documentation has 68 pages, contains 17 figures and 10 sources.

The goal is to implement a full-featured blog website with content management system, SEO optimization and integrated analytics for effective content management and promotion.

The work compares similar blog platforms and SEO tools, reveals typical functions of modern content management systems. The selected technology stack includes React, Express.js, MongoDB, and Tailwind CSS.

The system includes article editor, user dashboard, administrative panels, PostHog integration and SEO optimization. The system was tested for performance and user experience optimization.

KEY WORDS: BLOG WEBSITE, SEO OPTIMIZATION, WEB ANALYTICS, CONTENT MANAGEMENT SYSTEM, REACT, EXPRESS.JS.

**ЗМІСТ**

[ВСТУП 6](#_Toc200016001)

[РОЗДІЛ 1. АНАЛІЗ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ВЕБ-САЙТА ДЛЯ БЛОГУ 8](#_Toc200016002)

[1.1. Постановка основної задачі проєкту 8](#_Toc200016003)

[1.2. Вибір технологій для розробки веб-сайту 10](#_Toc200016004)

[Висновки до розділу 1 12](#_Toc200016005)

[РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБ-САЙТУ ДЛЯ БЛОГУ 14](#_Toc200016006)

[2.1. Аналіз вимог та потреб блогу 14](#_Toc200016007)

[2.2. Функціональні вимоги 16](#_Toc200016008)

[2.3. Нефункціональні вимоги 18](#_Toc200016009)

[2.4. Дослідження конкурентів 20](#_Toc200016010)

[2.5. Визначення вимог до користувацького інтерфейсу 23](#_Toc200016011)

[2.6. Дизайн інтерфейсу користувача 25](#_Toc200016012)

[Висновки до розділу 2 27](#_Toc200016013)

[РОЗДІЛ 3. РОЗРОБКА ВЕБ-САЙТУ ДЛЯ БЛОГУ 29](#_Toc200016014)

[3.1. Структура проєкту 29](#_Toc200016015)

[3.2. Структура веб-сайту 31](#_Toc200016016)

[3.3. Технічна реалізація веб-сайту 39](#_Toc200016017)

[Висновки до розділу 3 64](#_Toc200016018)

[ВИСНОВКИ 66](#_Toc200016019)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 68](#_Toc200016020)

[ДОДАТКИ 69](#_Toc200016021)

# ВСТУП

У сучасну цифрову епоху блоги стали потужним інструментом комунікації, що дозволяє людям та організаціям ділитися знаннями, досвідом та ідеями з широкою аудиторією. Створення контенту та його поширення через блогові платформи набуло значної популярності як серед авторів, так і серед компаній, які прагнуть встановити експертність у своїй галузі та залучити цільову аудиторію. Однак простого створення якісного контенту недостатньо для досягнення успіху в конкурентному середовищі інтернету.

Ключовими факторами успішності сучасного блогу є його видимість у пошукових системах та здатність залучати і утримувати читачів. Саме тому інтеграція SEO оптимізації та аналітичних інструментів стає критично важливою для будь-якої блогової платформи, що претендує на ефективність та зростання аудиторії.

Мета даного дипломного проєкту полягає у розробці повнофункціонального веб-сайту блогу з інтеграцією SEO оптимізації та аналітичних систем. Проєкт спрямований на створення платформи, яка забезпечить не лише зручне створення та публікацію контенту, але й максимізує його охоплення через оптимізацію для пошукових систем та надасть детальну аналітику для прийняття обґрунтованих рішень щодо контент-стратегії.

Основні завдання проєкту включають проектування та реалізацію інтуїтивного користувацького інтерфейсу для авторів та читачів, впровадження комплексних SEO рішень для покращення позицій у пошукових результатах, інтеграцію аналітичних інструментів для моніторингу ефективності контенту та поведінки користувачів, а також забезпечення оптимальної продуктивності та безпеки платформи.

У процесі реалізації проєкту використовуються сучасні технології веб-розробки, що забезпечують високу продуктивність та масштабованість. Фронтенд частина розробляється з використанням React – потужної JavaScript бібліотеки для створення інтерактивних користувацьких інтерфейсів, що дозволяє створити динамічний та відзивчивий досвід для користувачів. Для стилізації застосовується Tailwind CSS – утилітарний CSS фреймворк, який забезпечує швидку та гнучку розробку адаптивного дизайну.

Серверна частина реалізується на основі Express.js – швидкого та мінімалістичного веб-фреймворку для Node.js, який забезпечує надійну основу для API та серверної логіки. Для роботи з базою даних використовується Mongoose – елегантна бібліотека для моделювання об'єктів MongoDB, що спрощує взаємодію з даними та забезпечує валідацію схем.

Особлива увага приділяється технічним аспектам SEO оптимізації, включаючи серверний рендеринг для покращення індексації пошуковими системами, оптимізацію швидкості завантаження сторінок, структурування даних та метатегів. Інтеграція аналітичних інструментів дозволить отримувати цінні інсайти про поведінку користувачів, популярність контенту та ефективність SEO стратегій.

Таким чином, даний проєкт спрямований на створення комплексного рішення для блогінгу, яке поєднує сучасні технології веб-розробки з потужними інструментами цифрового маркетингу, що дозволить авторам максимізувати вплив їхнього контенту та ефективно розвивати свою онлайн видимість, або як ще кажуть «присутність».

# РОЗДІЛ 1. АНАЛІЗ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ВЕБ-САЙТА ДЛЯ БЛОГУ

**1.1. Постановка основної задачі проєкту**

Основною задачею даного проєкту є розробка повнофункціонального веб-сайту блогу з інтеграцією SEO оптимізації та аналітичних інструментів, який забезпечить ефективну платформу для створення, публікації та просування контенту з максимальним охопленням цільової аудиторії через пошукові системи.

Проєкт передбачає створення сучасної блогової платформи, яка поєднає інтуїтивний користувацький досвід з потужними технічними рішеннями для SEO оптимізації та детальної аналітики. Система повинна забезпечити авторам зручні інструменти для управління контентом, а читачам – комфортний досвід споживання інформації на будь-яких пристроях.

Для досягнення поставленої мети необхідно вирішити наступні ключові завдання:

1. Аналіз функціональних вимог та потреб користувачів. Проведення детального дослідження потреб різних категорій користувачів системи, включаючи авторів контенту, адміністраторів та читачів блогу. Визначення основних функціональних можливостей, які повинні бути реалізовані для забезпечення ефективної роботи з контентом та його споживання.

2. Проектування архітектури системи. Розробка комплексної архітектури веб-додатку, яка включатиме клієнтську частину на базі React, серверну частину з використанням Express.js та базу даних MongoDB з Mongoose ODM. Планування структури API, компонентної архітектури фронтенду та схеми даних для оптимального зберігання контенту блогу.

3. Розробка стратегії SEO оптимізації. Створення комплексного підходу до пошукової оптимізації, який включатиме технічні аспекти SEO, такі як серверний рендеринг, оптимізація швидкості завантаження, структурування даних Schema.org, налаштування метатегів та OpenGraph розмітки. Планування системи внутрішньої перелінковки та оптимізації URL структури.

4. Інтеграція аналітичних інструментів. Впровадження систем веб-аналітики для моніторингу поведінки користувачів, аналізу ефективності контенту та відстеження ключових показників продуктивності. Розробка дашборду для авторів з детальною статистикою переглядів, джерел трафіку та взаємодії з контентом.

5. Створення адаптивного користувацького інтерфейсу. Розробка сучасного та інтуїтивного дизайну з використанням Tailwind CSS, який забезпечить оптимальний досвід користування на всіх типах пристроїв. Реалізація системи управління контентом з розширеним текстовим редактором, можливістю додавання мультимедійного контенту та попереднього перегляду публікацій.

6. Забезпечення продуктивності та безпеки. Оптимізація швидкості роботи додатку через ефективне кешування, мінімізацію та стиснення ресурсів. Впровадження заходів безпеки для захисту від основних веб-загроз, включаючи валідацію даних, автентифікацію та авторизацію користувачів.

7. Тестування та налагодження системи. Проведення комплексного тестування функціональності блогу, включаючи перевірку коректності роботи SEO оптимізації, точності аналітичних даних та адаптивності інтерфейсу на різних пристроях та у різних браузерах. Виявлення та усунення технічних недоліків для забезпечення стабільної роботи платформи.

Реалізація даного проєкту дозволить створити конкурентоспроможну блогову платформу, яка не лише забезпечить зручне створення та споживання контенту, але й максимізує його видимість у пошукових системах та надасть авторам необхідні інструменти для аналізу ефективності їхньої контент-стратегії.

## 1.2. Вибір технологій для розробки веб-сайту

Вибір технологій для розробки веб-сайту блогу з інтеграцією SEO оптимізації та аналітики є критично важливим етапом проєкту, оскільки від цього залежить продуктивність системи, її масштабованість, зручність підтримки та здатність ефективно індексуватися пошуковими системами. Під час вибору технологічного стеку було враховано специфіку завдань блогової платформи, вимоги до SEO оптимізації, необхідність інтеграції аналітичних інструментів та сучасні тенденції у веб-розробці.

1. React.js використовується як основна бібліотека для створення користувацького інтерфейсу. React забезпечує ефективне управління станом додатку через компонентну архітектуру, що дозволяє створювати динамічні та інтерактивні елементи інтерфейсу з високою продуктивністю. Віртуальний DOM React оптимізує процес оновлення сторінки, мінімізуючи кількість операцій з реальним DOM та покращуючи загальну швидкість роботи додатку. Для блогової платформи React дозволяє реалізувати складні інтерфейси управління контентом, динамічне завантаження статей та інтерактивні елементи без перезавантаження сторінки.

2. Express.js обрано як серверний фреймворк для Node.js завдяки його мінімалістичному підходу та гнучкості. Express забезпечує швидку розробку RESTful API, що є необхідним для управління контентом блогу, автентифікації користувачів та інтеграції з зовнішніми сервісами аналітики. Фреймворк дозволяє легко налаштувати маршрутизацію, middleware для обробки запитів та серверний рендеринг для покращення SEO показників. Завдяки великій екосистемі пакетів npm, Express забезпечує простоту інтеграції з різноманітними інструментами та сервісами.

3. MongoDB з Mongoose ODM використовується для зберігання та управління даними блогу. MongoDB як документо-орієнтована база даних ідеально підходить для зберігання контенту блогу завдяки гнучкій схемі, що дозволяє легко адаптуватися до змін структури статей та метаданих. Mongoose забезпечує елегантне моделювання об'єктів, валідацію схем та зручні методи для роботи з даними. Це особливо важливо для блогової платформи, де структура контенту може варіюватися, а необхідність швидкого пошуку та фільтрації статей є критичною.

4. Tailwind CSS обрано як CSS фреймворк для стилізації інтерфейсу завдяки його утилітарному підходу. Tailwind дозволяє швидко створювати адаптивний дизайн без написання власного CSS коду, що прискорює процес розробки та забезпечує консистентність дизайну. Фреймворк включає вбудовані класи для створення responisve дизайну, що є критично важливим для блогу, оскільки значна частина трафіку надходить з мобільних пристроїв.

5. Next.js інтегрується як React фреймворк для забезпечення серверного рендерингу та статичної генерації сторінок. Це критично важливо для SEO оптимізації блогу, оскільки серверний рендеринг забезпечує краще індексування контенту пошуковими роботами. Next.js також надає вбудовані оптимізації для зображень, автоматичне розділення коду та підтримку Progressive Web App функціональності, що покращує загальну продуктивність платформи.

6. JWT (JSON Web Tokens) використовується для автентифікації та авторизації користувачів. JWT забезпечує безпечний обмін інформацією між клієнтом та сервером без необхідності зберігання сесій на сервері, що спрощує масштабування додатку. Для блогової платформи це дозволяє реалізувати систему ролей для авторів, редакторів та адміністраторів з різними рівнями доступу до функціональності.

7. PostHog інтегрується для забезпечення аналітики. PostHog надає детальну інформацію про поведінку користувачів, популярність контенту та джерела трафіку.

Обраний технологічний стек забезпечує створення сучасної, швидкої та SEO-оптимізованої блогової платформи, яка може ефективно масштабуватися та адаптуватися до змінних вимог контент-маркетингу. Поєднання React екосистеми з потужними серверними рішеннями та спеціалізованими інструментами для SEO створює оптимальну основу для досягнення поставлених цілей проєкту.

## Висновки до розділу 1

Розробка веб-сайту блогу з інтеграцією SEO оптимізації та аналітики є складним технічним завданням, що вимагає ретельного планування архітектури системи та обґрунтованого вибору технологічного стеку. Проведений у даному розділі аналіз дозволив сформувати чітке уявлення про основні завдання проєкту та визначити оптимальні технологічні рішення для їх реалізації.

Постановка основної задачі проєкту виявила необхідність створення комплексної системи, яка поєднує функціональність сучасної блогової платформи з потужними інструментами пошукової оптимізації та детальної аналітики. Ключовими аспектами системи визначено забезпечення зручного управління контентом для авторів, оптимального користувацького досвіду для читачів та максимальної видимості у пошукових системах. Структурований підхід до постановки завдань дозволив виділити сім основних напрямків роботи, від аналізу функціональних вимог до комплексного тестування системи.

Аналіз технологічних рішень продемонстрував переваги використання сучасного JavaScript стеку для розробки блогової платформи. Вибір React як основної бібліотеки для фронтенду обґрунтовується необхідністю створення динамічного та інтерактивного інтерфейсу з оптимальною продуктивністю. Express.js забезпечує гнучкість серверної частини та простоту інтеграції з різноманітними сервісами, що є критично важливим для реалізації SEO функціональності та підключення аналітичних інструментів.

Особлива увага приділена вибору MongoDB з Mongoose ODM для управління даними, що забезпечує необхідну гнучкість схеми для зберігання різноманітного контенту блогу та ефективний пошук інформації. Використання Tailwind CSS дозволяє швидко створювати адаптивний дизайн, що є невід'ємною вимогою сучасних веб-додатків, особливо з урахуванням зростаючої частки мобільного трафіку.

Інтеграція Next.js як React фреймворку для серверного рендерингу визначена як ключовий елемент SEO стратегії, оскільки забезпечує краще індексування контенту пошуковими системами. Поєднання з JWT для автентифікації створює безпечну та масштабовану систему управління користувачами з різними рівнями доступу.

Інтеграція PostHog забезпечить моніторинг ефективності блогу, що в свою чергу дозволить приймати обґрунтовані рішення щодо розвитку платформи на основі реальних даних про поведінку користувачів та позиції у пошукових результатах.

Обраний технологічний стек характеризується високою сумісністю компонентів, що забезпечує стабільну роботу системи та спрощує процес розробки. Використання сучасних стандартів та найкращих практик веб-розробки гарантує створення масштабованого рішення, здатного адаптуватися до зростаючих вимог та змін у сфері цифрового маркетингу.

Таким чином, проведений аналіз створив міцну теоретичну основу для практичної реалізації проєкту. Чітко сформульовані завдання та обґрунтований вибір технологій забезпечують оптимальні передумови для створення ефективної блогової платформи, яка відповідатиме сучасним стандартам веб-розробки та вимогам пошукової оптимізації.

# РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБ-САЙТУ ДЛЯ БЛОГУ

## 2.1. Аналіз вимог та потреб блогу

Аналіз вимог та потреб блогу представляє фундаментальний етап розробки, який визначає архітектуру, функціональність та стратегічні цілі веб-платформи. Сучасний блог повинен забезпечувати не лише публікацію контенту, але й ефективну взаємодію з аудиторією, оптимізацію для пошукових систем та аналітику користувацької поведінки.

Основні функціональні вимоги до веб-сайту блогу включають створення системи управління контентом з можливістю публікації, редагування та видалення статей. Система повинна підтримувати категоризацію матеріалів та систему тегів для ефективної організації контенту. Важливою вимогою є реалізація системи коментарів з модерацією, що дозволить забезпечити якісну взаємодію між автором та читачами.

Пошукова функціональність має забезпечувати швидкий доступ до релевантного контенту як для користувачів, так і для пошукових роботів. Система повинна підтримувати адаптивний дизайн для коректного відображення на різних пристроях та забезпечувати високу швидкість завантаження сторінок.

Інтеграція SEO-оптимізації вимагає реалізації автоматичної генерації мета-тегів, створення XML-карт сайту та забезпечення семантичної розмітки контенту. Система повинна підтримувати налаштування ЧПУ (людино-зрозумілих URL) та автоматичну оптимізацію зображень для покращення показників завантаження. Технічна архітектура на базі React забезпечить створення динамічного користувацького інтерфейсу з серверним рендерингом для покращення SEO-показників. Express.js як серверна частина надасть гнучкість у створенні API, а Mongoose забезпечить ефективну роботу з базою даних MongoDB.

Система аналітики повинна відстежувати ключові метрики користувацької активності, включаючи час перебування на сторінці, показники відмов та конверсії. Інтеграція з PostHog та власними інструментами аналітики дозволить отримувати детальну інформацію про поведінку аудиторії та ефективність контент-стратегії.

Основну цільову аудиторію блогу складають активні інтернет-користувачі, зацікавлені в отриманні якісного та актуального контенту у визначеній тематичній ніші. Читачі очікують швидкого доступу до інформації, зручної навігації та можливості взаємодії через коментарі та соціальні мережі. Автори контенту потребують інтуїтивно зрозумілої системи управління, що дозволить ефективно створювати, редагувати та публікувати матеріали без глибоких технічних знань. Адміністратори сайту мають отримати комплексні інструменти для модерації контенту, управління користувачами та аналізу ефективності платформи.

Веб-сайт повинен забезпечувати стабільну роботу при одночасному доступі великої кількості користувачів та швидке завантаження сторінок незалежно від обсягу контенту. Система безпеки має включати захист від основних типів атак, шифрування даних та регулярне резервне копіювання. Масштабованість архітектури дозволить розширювати функціональність платформи відповідно до зростання аудиторії та зміни потреб бізнесу. Модульна структура коду забезпечить легкість подальшого розвитку та інтеграції нових можливостей.

## 2.2. Функціональні вимоги

Для забезпечення ефективного функціонування блог-платформи та задоволення потреб різних груп користувачів було визначено наступні функціональні вимоги системи.

Головна сторінка блогу повинна представляти огляд найновіших публікацій з можливістю швидкого переходу до повного тексту статей. Сторінка має містити навігаційне меню для доступу до основних розділів сайту, пошукову форму та блок з популярними або рекомендованими матеріалами. Важливим елементом є відображення категорій контенту та хмари тегів для зручної навігації по тематичних напрямках.

Сторінка окремої статті має забезпечувати комфортне читання з оптимізованим форматуванням тексту, підтримкою мультимедійного контенту та соціальних функцій. Система повинна автоматично генерувати зміст для довгих матеріалів, відображати дату публікації, інформацію про автора та час читання. Інтеграція кнопок соціальних мереж дозволить читачам легко ділитися цікавим контентом.

Система коментарів має забезпечувати можливість залишати відгуки авторизованими користувачами. Функціональність включає можливість відповідати на коментарі, створюючи древовидну структуру обговорення, та систему оцінювання коментарів іншими користувачами.

Адміністративна панель повинна надавати повний контроль над контентом сайту через інтуїтивно зрозумілий інтерфейс. Система управління статтями включає створення, редагування та видалення публікацій з можливістю збереження чернеток та планування публікації на майбутнє. Функціональність управління категоріями та тегами дозволить ефективно організовувати контентну структуру.

Пошукова система має забезпечувати швидкий та релевантний пошук по всьому контенту сайту з підтримкою фільтрації за категоріями, датами та авторами. Система повинна підтримувати пошук за ключовими словами в заголовках, тексті статей та метаданих з автоматичними підказками та виправленням помилок.

Система аналітики має відстежувати ключові метрики відвідуваності, поведінки користувачів та ефективності контенту. Функціональність включає генерацію звітів про найпопулярніші статті, джерела трафіку, час перебування на сторінках та конверсійні показники. Інтеграція з зовнішніми аналітичними сервісами дозволить отримувати розширену статистику та insights.

SEO-модуль повинен автоматично оптимізувати контент для пошукових систем через генерацію мета-тегів, створення семантичної розмітки та забезпечення технічної оптимізації. Система має підтримувати налаштування індивідуальних SEO-параметрів для кожної сторінки та автоматичну генерацію XML sitemap з регулярним оновленням.

Система підписки на розсилку дозволить читачам отримувати сповіщення про нові публікації через електронну пошту. Функціональність включає управління списками підписників, створення персоналізованих розсилок та аналітику ефективності email-маркетингу.

Мобільна адаптивність забезпечить коректне відображення та повну функціональність сайту на пристроях з різними розмірами екрану. Система має оптимізувати швидкість завантаження на мобільних пристроях та забезпечувати зручну навігацію через touch-інтерфейс.

## 2.3. Нефункціональні вимоги

Окрім вимог функціональних, треба ще і врахувати й нефункціональні, вони також визначають якість роботи сайту. Нефункціональні вимоги визначають якісні характеристики веб-сайту блогу та встановлюють критерії, які забезпечують ефективну роботу системи, задоволення потреб користувачів та відповідність сучасним стандартам веб-розробки. Ці вимоги є критично важливими для досягнення успіху блогової платформи, оскільки впливають на користувацький досвід, позиції у пошукових системах та загальну конкурентоспроможність ресурсу.

Продуктивність та швидкість завантаження становлять фундаментальну вимогу для сучасного блогу. Платформа повинна забезпечувати швидкість завантаження основних сторінок не більше двох секунд при стандартному широкосмуговому підключенні. Час першого контентного відображення має не перевищувати однієї секунди, що критично важливо для утримання користувачів та покращення SEO показників. Система повинна ефективно обробляти статичні ресурси через CDN, застосовувати компресію зображень та мінімізацію CSS і JavaScript файлів для досягнення оптимальних показників Core Web Vitals.

Адаптивність та кросплатформна сумісність забезпечують коректне відображення контенту на всіх типах пристроїв та у різних браузерах. Дизайн платформи повинен адаптуватися до екранів від мобільних телефонів з діагоналлю 320 пікселів до широкоформатних моніторів з роздільною здатністю понад 1920 пікселів. Інтерфейс має підтримувати як портретну, так і альбомну орієнтацію на мобільних пристроях, забезпечуючи однаково зручне використання у всіх конфігураціях. Сумісність з основними браузерами включає підтримку останніх версій Chrome, Firefox, Safari та Edge.

Безпека даних та захист інформації представляють критично важливий аспект роботи блогової платформи. Система повинна використовувати HTTPS протокол для всіх з'єднань, забезпечуючи шифрування передачі даних між клієнтом та сервером. Автентифікація користувачів здійснюється через захищені JWT токени з обмеженим терміном дії та можливістю оновлення. Всі користувацькі дані зберігаються з використанням сучасних методів хешування паролів та валідації вхідних даних для запобігання XSS та SQL ін'єкцій атакам.

Масштабованість архітектури забезпечує здатність системи справлятися зі зростаючим навантаженням та розширенням функціональності. База даних повинна ефективно обробляти до десяти тисяч одночасних запитів без значного зниження продуктивності. Архітектура системи підтримує горизонтальне масштабування серверних ресурсів та розподіл навантаження між декількома інстансами додатку. Код організований у модульній структурі, що дозволяє легко додавати нові функціональні компоненти без впливу на існуючий функціонал.

SEO оптимізація та індексування інтегровані на рівні архітектури системи. Платформа генерує семантичну HTML розмітку з правильним використанням заголовків, метатегів та структурованих даних Schema.org. Серверний рендеринг забезпечує повноцінне індексування контенту пошуковими роботами, а система sitemap автоматично оновлюється при додаванні нового контенту.

Надійність та відмовостійкість гарантують стабільну роботу платформи навіть при несприятливих умовах. Система включає механізми автоматичного відновлення після збоїв, регулярне резервне копіювання даних та моніторинг основних показників продуктивності. Час безвідмовної роботи повинен становити не менше 99.5% протягом місяця з урахуванням планових технічних робіт.

Юзабіліті та користувацький досвід визначають інтуїтивність та зручність використання платформи. Інтерфейс повинен дозволяти новим користувачам швидко орієнтуватися у функціональності без додаткового навчання. Час виконання основних операцій, таких як створення нової публікації або пошук контенту, не повинен перевищувати п'яти кліків від головної сторінки.

Дотримання цих нефункціональних вимог забезпечує створення професійної блогової платформи, яка відповідає сучасним стандартам веб-розробки та забезпечує оптимальний досвід для всіх категорій користувачів.

## 2.4. Дослідження конкурентів

Аналіз конкурентного середовища є важливим етапом розробки блогової платформи, оскільки дозволяє визначити найкращі практики галузі, виявити недоліки існуючих рішень та сформувати унікальну пропозицію власного продукту. У ході дослідження було проаналізовано провідні блогові платформи, які займають домінуючі позиції на ринку та представляють різні підходи до організації контенту та взаємодії з користувачами.

Medium представляє собою одну з найпопулярніших сучасних блогових платформ, яка зосереджена на якісному контенті та мінімалістичному дизайні. Платформа забезпечує чистий та зручний інтерфейс для читання, що сприяє фокусуванню уваги на змісті публікацій. Medium використовує алгоритмічну систему рекомендацій для персоналізації контенту, що дозволяє користувачам знаходити релевантні матеріали відповідно до їхніх інтересів.

Однією з ключових особливостей Medium є система монетизації через підписку Medium Partner Program, яка дозволяє авторам отримувати винагороду за якісний контент. Платформа також надає базову аналітику для авторів, включаючи статистику переглядів, читання та взаємодії з публікаціями. Проте Medium має обмеження щодо можливостей SEO оптимізації для індивідуальних авторів, оскільки більшість налаштувань контролюється самою платформою.

WordPress.com як хостингова версія популярної системи управління контентом представляє більш традиційний підхід до ведення блогу. Платформа надає широкі можливості кастомізації дизайну через систему тем та плагінів, що дозволяє створювати унікальні за виглядом блоги. WordPress забезпечує комплексні інструменти SEO оптимізації, включаючи налаштування метатегів, XML карт сайту та структурованих даних.

Система аналітики WordPress включає детальну статистику відвідувань, популярних публікацій та джерел трафіку. Платформа також підтримує інтеграцію з аналітичними сервісами для отримання більш глибоких інсайтів. Недоліком WordPress.com є складність інтерфейсу для початкових користувачів та обмеження функціональності у безкоштовних планах.

Ghost позиціонується як платформа для професійних видавців та зосереджується на швидкості роботи та SEO оптимізації. Платформа використовує сучасний технологічний стек з Node.js, що забезпечує високу продуктивність та швидкість завантаження сторінок. Ghost надає вбудовані інструменти для email маркетингу та створення платних підписок, що робить його привабливим для комерційних проєктів.

Система аналітики Ghost включає детальні звіти про ефективність контенту, поведінку підписників та фінансові показники для платних підписок. Платформа забезпечує автоматичну генерацію AMP сторінок та оптимізацію для соціальних мереж, що покращує видимість контенту. Однак Ghost має обмежені можливості кастомізації порівняно з WordPress та вимагає технічних знань для повного використання потенціалу.

Substack спеціалізується на newsletter форматі контенту та монетизації через платні підписки. Платформа автоматично генерує як веб версії публікацій, так і email розсилки, що забезпечує максимальне охоплення аудиторії. Substack надає простий та інтуїтивний інтерфейс для авторів, що дозволяє зосередитися на створенні контенту без технічних складнощів.

Аналітика Substack зосереджена на метриках підписників, включаючи відкриття email листів, кліки та конверсію в платні підписки. Платформа забезпечує інтеграцію з соціальними мережами та можливості для кросс-промоушену між авторами. Обмеженням Substack є фокус на newsletter форматі, що може не підходити для всіх типів контенту та авторів.

Порівняльний аналіз виявив, що кожна платформа має свої сильні та слабкі сторони. Medium забезпечує найкращий досвід читання та систему виявлення контенту, але обмежує контроль авторів над SEO. WordPress надає максимальну гнучкість та SEO можливості, але вимагає більше технічних знань. Ghost оптимізований для продуктивності та професійного видавництва, тоді як Substack ідеально підходить для newsletter контенту.

Спільними недоліками існуючих рішень є недостатня інтеграція аналітичних інструментів, обмежені можливості персоналізації SEO налаштувань для окремих публікацій та відсутність комплексного підходу до оптимізації контенту для різних каналів розповсюдження. Більшість платформ також не надають достатньо детальної аналітики поведінки користувачів для оптимізації контент стратегії.

На основі проведеного аналізу було визначено ключові можливості для створення конкурентоспроможної блогової платформи. Розроблюване рішення повинно поєднувати простоту використання Medium з гнучкістю WordPress, продуктивність Ghost та комплексний підхід до аналітики, який перевершує можливості існуючих платформ. Особлива увага має бути приділена інтеграції передових SEO інструментів та створенню системи аналітики, яка надає практичні рекомендації для покращення ефективності контенту.

## 2.5. Визначення вимог до користувацького інтерфейсу

На основі проведеного аналізу цільової аудиторії та функціональних потреб блог-платформи було визначено комплекс вимог до користувацького інтерфейсу, що забезпечать оптимальний користувацький досвід та ефективну взаємодію з системою.

Першочерговою вимогою є забезпечення простоти та інтуїтивності інтерфейсу, що дозволить користувачам різного рівня технічної підготовки ефективно взаємодіяти з платформою. Дизайн повинен слідувати загальноприйнятим принципам веб-юзабіліті та використовувати знайомі користувачам патерни взаємодії. Це включає логічне розміщення елементів управління, зрозумілі піктограми та консистентність у використанні кольорів та типографіки по всьому сайту.

Навігаційна структура має забезпечувати швидкий доступ до основних розділів блогу через чітко структуроване головне меню, хлібні крихти та контекстні посилання. Користувачі повинні мати можливість легко орієнтуватися в контентній структурі сайту та швидко знаходити потрібну інформацію за допомогою пошукової системи з автодоповненням та фільтрами. Бокова панель з категоріями, архівом публікацій та популярними матеріалами додатково підвищить зручність навігації.

Візуальний дизайн інтерфейсу повинен відображати сучасні тенденції веб-дизайну з використанням чистого та мінімалістичного підходу, що не відволікає увагу від основного контенту. Колірна схема має бути збалансованою та професійною, з достатнім контрастом для забезпечення читабельності тексту. Типографіка повинна підтримувати комфортне читання на різних пристроях з оптимальним розміром шрифту та міжрядковими інтервалами.

Адаптивність інтерфейсу забезпечить коректне відображення та повну функціональність на пристроях з різними розмірами екрану. Мобільна версія сайту має оптимізовану навігацію з hamburger-меню, touch-friendly елементами управління та швидким завантаженням сторінок. Планшетна версія повинна ефективно використовувати доступний простір екрану для відображення контенту та навігаційних елементів.

Швидкість відгуку інтерфейсу є критично важливою для забезпечення позитивного користувацького досвіду. Всі інтерактивні елементи повинні миттєво реагувати на дії користувача з відповідним візуальним зворотним зв'язком. Форми мають включати валідацію в реальному часі з інформативними повідомленнями про помилки. Завантаження контенту повинно супроводжуватися індикаторами прогресу або skeleton-анімаціями.

Система зворотного зв'язку повинна інформувати користувачів про результати їхніх дій через toast-повідомлення, модальні вікна або статусні індикатори. Помилки мають супроводжуватися зрозумілими поясненнями та рекомендаціями щодо їх усунення.

Персоналізація інтерфейсу дозволить користувачам налаштовувати відображення контенту відповідно до своїх уподобань, включаючи вибір теми оформлення, розмір шрифту та щільність відображення інформації.

Таким чином, визначені вимоги до користувацького інтерфейсу створюють основу для розробки сучасної та функціональної блог-платформи, що забезпечить високий рівень користувацького досвіду та ефективність взаємодії з системою. Наступним етапом є детальне проектування архітектури системи та вибір оптимальних технологічних рішень для реалізації поставлених цілей.

## 2.6. Дизайн інтерфейсу користувача

Дизайн інтерфейсу користувача представляє критично важливий аспект розробки блог-платформи, оскільки він безпосередньо впливає на користувацький досвід, залученість аудиторії та загальну ефективність системи. Якісний дизайн інтерфейсу забезпечує не лише естетичну привабливість, але й функціональність, доступність та зручність використання для всіх категорій користувачів.

Основоположним принципом дизайну блог-платформи є забезпечення максимальної зручності використання через створення інтуїтивно зрозумілого інтерфейсу. Кожен елемент системи повинен мати очевидне призначення та розташування, що дозволить користувачам ефективно взаємодіяти з контентом без необхідності детального вивчення функціональності. Навігаційні елементи розміщуються у передбачуваних місцях, форми мають зрозумілі підписи та підказки, а інтерактивні компоненти забезпечують чіткий візуальний зворотний зв'язок на дії користувача.

Консистентність дизайну забезпечується через створення єдиної системи візуальних стандартів, що застосовуються по всьому сайту. Це включає використання узгодженої колірної палітри, типографічної ієрархії, стилів кнопок та форм, а також принципів компонування елементів на сторінці. Системний підхід до дизайну дозволяє користувачам швидко адаптуватися до інтерфейсу та ефективно орієнтуватися в різних розділах платформи.

Візуальна привабливість досягається через баланс між естетикою та функціональністю, використовуючи сучасні принципи веб-дизайну. Чиста та мінімалістична композиція забезпечує фокус на контенті, водночас підтримуючи професійний та привабливий зовнішній вигляд. Ретельно підібрана типографіка забезпечує відмінну читабельність на всіх пристроях, а продумана колірна схема створює гармонійну атмосферу та підтримує брендову ідентичність.

Адаптивний дизайн реалізується через застосування гнучких сіток та медіа-запитів, що забезпечують оптимальне відображення контенту на пристроях з різними характеристиками екрану. Мобільна версія отримує спеціально оптимізовані елементи управління, адаптовані для сенсорної взаємодії, включаючи збільшені області дотику та спрощену навігацію.

Система візуальної ієрархії організовує інформацію за ступенем важливості через використання розміру, кольору, контрасту та позиціонування елементів. Основний контент отримує найбільшу візуальну вагу, навігаційні елементи розміщуються у другорядних зонах, а допоміжна інформація представляється у менш помітних областях інтерфейсу.

Інтерактивність підвищується через застосування сучасних анімацій та переходів, що роблять взаємодію з інтерфейсом більш природною та приємною. Ці елементи використовуються помірно та цілеспрямовано, підкреслюючи важливі дії та забезпечуючи візуальну континуальність при переході між станами інтерфейсу.

Доступність забезпечується через дотримання стандартів веб-доступності, включаючи достатній колірний контраст, альтернативні способи навігації, семантичну розмітку та підтримку допоміжних технологій. Це гарантує можливість ефективного використання платформи користувачами з різними потребами та обмеженнями.

Результатом застосування цих принципів стає створення цілісного дизайн-рішення, що оптимально поєднує естетичні та функціональні аспекти, забезпечуючи високий рівень користувацького задоволення та ефективність досягнення цілей як читачів, так і авторів контенту.

## Висновки до розділу 2

Проектування блог-платформи з інтеграцією SEO-оптимізації та аналітики представляє технологічний процес, що вимагає ретельного аналізу вимог, стратегічного планування архітектури та продуманого підходу до дизайну користувацького інтерфейсу. Проведений аналіз потреб сучасного блогу дозволив визначити ключові функціональні та нефункціональні вимоги, що формують основу для створення ефективної контент-платформи.

Систематизація функціональних вимог виявила необхідність реалізації комплексної системи управління контентом, яка поєднує інтуїтивний інтерфейс для авторів з потужними можливостями для читачів. Особлива увага приділена інтеграції SEO-інструментів та аналітичних систем, що забезпечить високу видимість контенту в пошукових системах та надасть детальну інформацію про поведінку аудиторії.

Визначення цільової аудиторії та аналіз її потреб дозволили сформувати чітке розуміння користувацьких сценаріїв та пріоритетів функціональності. Читачі потребують швидкого доступу до якісного контенту та зручних інструментів взаємодії, тоді як автори та адміністратори очікують потужних можливостей управління та аналітики.

Технічні вимоги до платформи охоплюють забезпечення високої продуктивності, масштабованості та безпеки системи. Використання сучасного технологічного стеку на базі React, Express та MongoDB створює міцну основу для реалізації всіх поставлених цілей та забезпечує можливості для майбутнього розширення функціональності.

Розробка вимог до користувацького інтерфейсу та принципів дизайну встановила стандарти для створення сучасного, доступного та ефективного веб-додатку. Акцент на простоті використання, консистентності та адаптивності гарантує високий рівень користувацького досвіду на всіх типах пристроїв.

Особливої уваги заслуговує інтеграція SEO-оптимізації як невід'ємної частини архітектури системи, що забезпечить органічне зростання трафіку та покращення позицій у пошукових результатах. Поєднання з комплексною аналітикою створює потужний інструмент для розуміння ефективності контент-стратегії та оптимізації користувацького досвіду.

Результатом проектувального етапу стало формування цілісного бачення блог-платформи, яка поєднує сучасні технологічні рішення з глибоким розумінням потреб користувачів. Визначені вимоги та принципи створюють надійну основу для подальших етапів розробки та забезпечують створення конкурентоспроможного продукту, здатного ефективно вирішувати задачі контент-маркетингу та взаємодії з аудиторією в сучасному цифровому середовищі.

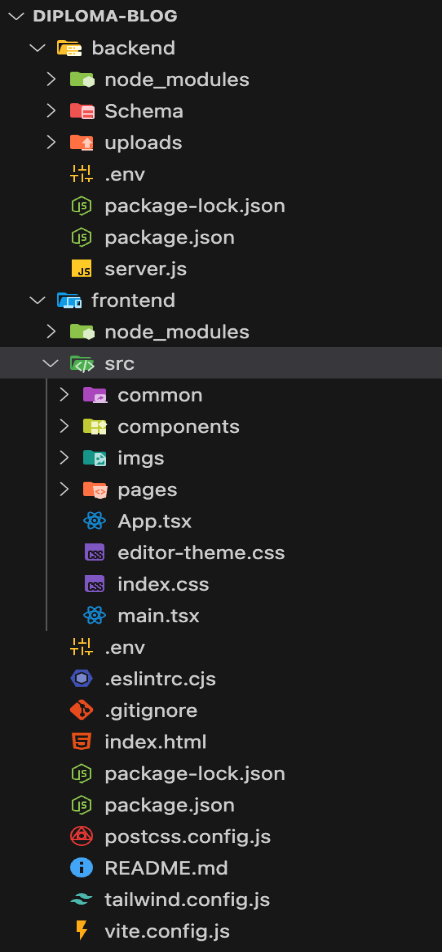
# РОЗДІЛ 3. РОЗРОБКА ВЕБ-САЙТУ ДЛЯ БЛОГУ

## 3.1. Структура проєкту

Структура проєкту є важливою частиною розробки веб-сайту, яка визначає організацію коду, розподіл файлів та папок, що забезпечує зручність розробки, підтримки та розширення функціоналу в майбутньому. Правильна архітектура проєкту сприяє ефективній командній роботі, полегшує процес налагодження та підтримки коду, а також забезпечує масштабованість додатку.

Структура проєкту для веб-сайту блогу побудована за принципом розділення на клієнтську та серверну частини, що відповідає архітектурі повнофункціонального веб-додатку (рис. 3.1.1):

* Структура серверної частини (Backend) містить серверну частину додатку, побудовану на технологіях Node.js та Express.js:
  + Папка Schema/: містить моделі даних для MongoDB з використанням Mongoose ODM, включаючи схеми для користувачів, статей блогу, коментарів та інших сутностей
  + Папка uploads/: призначена для зберігання завантажених файлів, таких як зображення для статей та аватари користувачів
  + Файл server.js: головний файл сервера, який містить конфігурацію Express-додатку, маршрутизацію API та підключення до бази даних
* Структура клієнтської частини (Frontend) містить клієнтську частину додатку, розроблену з використанням React та TypeScript:
  + Файл .env: містить змінні середовища для конфігурації додатку
  + Файл .eslintrc.cjs: налаштування ESLint для забезпечення якості коду
  + Файл package.json: містить метадані проєкту та залежності
  + Файл vite.config.js: конфігурація збирача Vite для розробки та побудови проєкту
  + Файл tailwind.config.js: налаштування Tailwind CSS фреймворку
  + Файл postcss.config.js: конфігурація PostCSS для обробки стилів
  + Файл main.tsx: головна точка входу React-додатку
  + Файл App.tsx: кореневий компонент додатку з основною логікою маршрутизації
  + Файл index.css: глобальні стилі додатку
  + Файл editor-theme.css: спеціалізовані стилі для текстового редактора статей
  + Папка components/: містить багаторазові React-компоненти, такі як навігація, форми, модальні вікна та інші елементи UI
  + Папка pages/: включає компоненти-сторінки додатку, такі як головна сторінка, сторінка статті, панель адміністрування тощо
  + Папка common/: зберігає загальні утиліти, хуки, константи та допоміжні функції
  + Папка imgs/: містить статичні зображення, логотипи та інші графічні ресурси

  
Рисунок 3.1.1

## 3.2. Структура веб-сайту

Веб-сайт блогу побудований за принципом односторінкового додатку (SPA) з використанням React Router для навігації між різними розділами. Архітектура сайту забезпечує інтуїтивну навігацію та зручний користувацький досвід. Система маршрутизації веб-сайту організована ієрархічно з використанням вкладених маршрутів.

Головна сторінка (/) - відображає останні публікації блогу, популярні статті та основну навігацію (рис. 3.2.1)

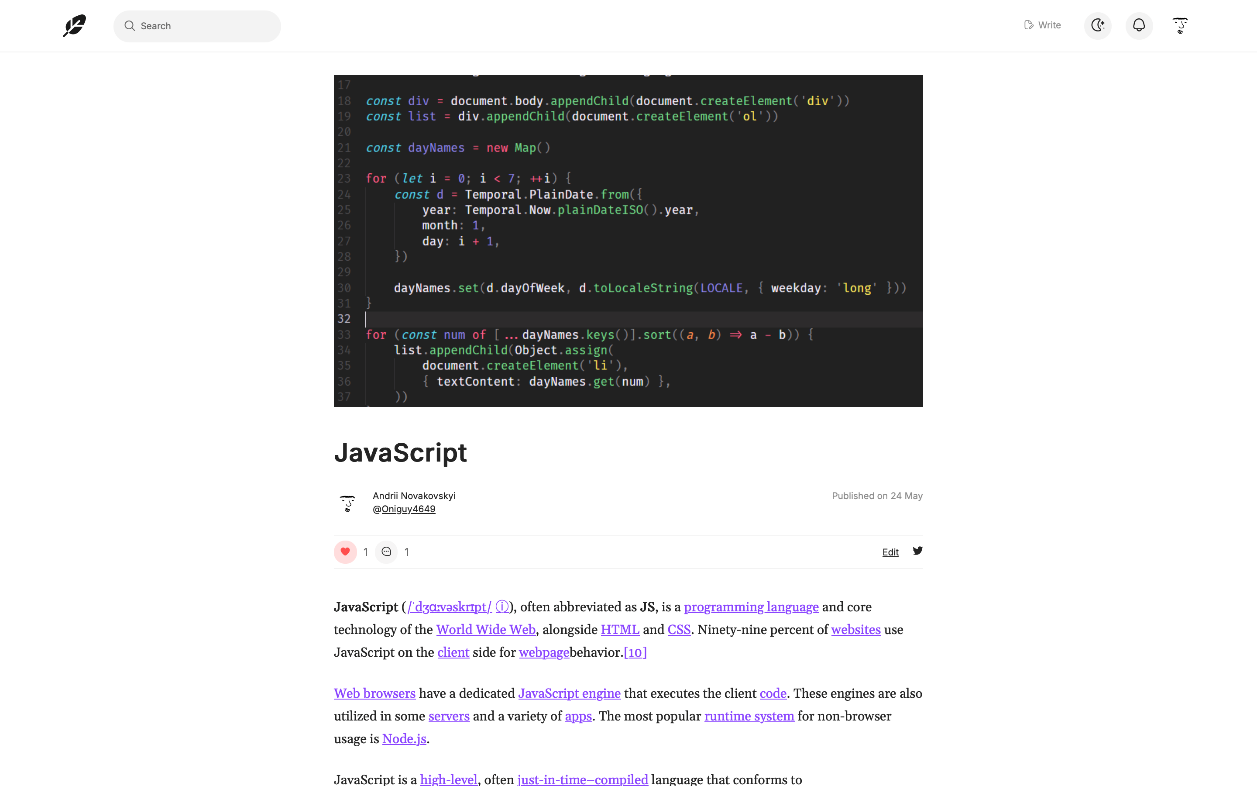


Рисунок 3.2.1

Сторінка статті (/blog/:blog\_id) - детальний перегляд окремої статті з можливістю коментування (рис. 3.2.2)

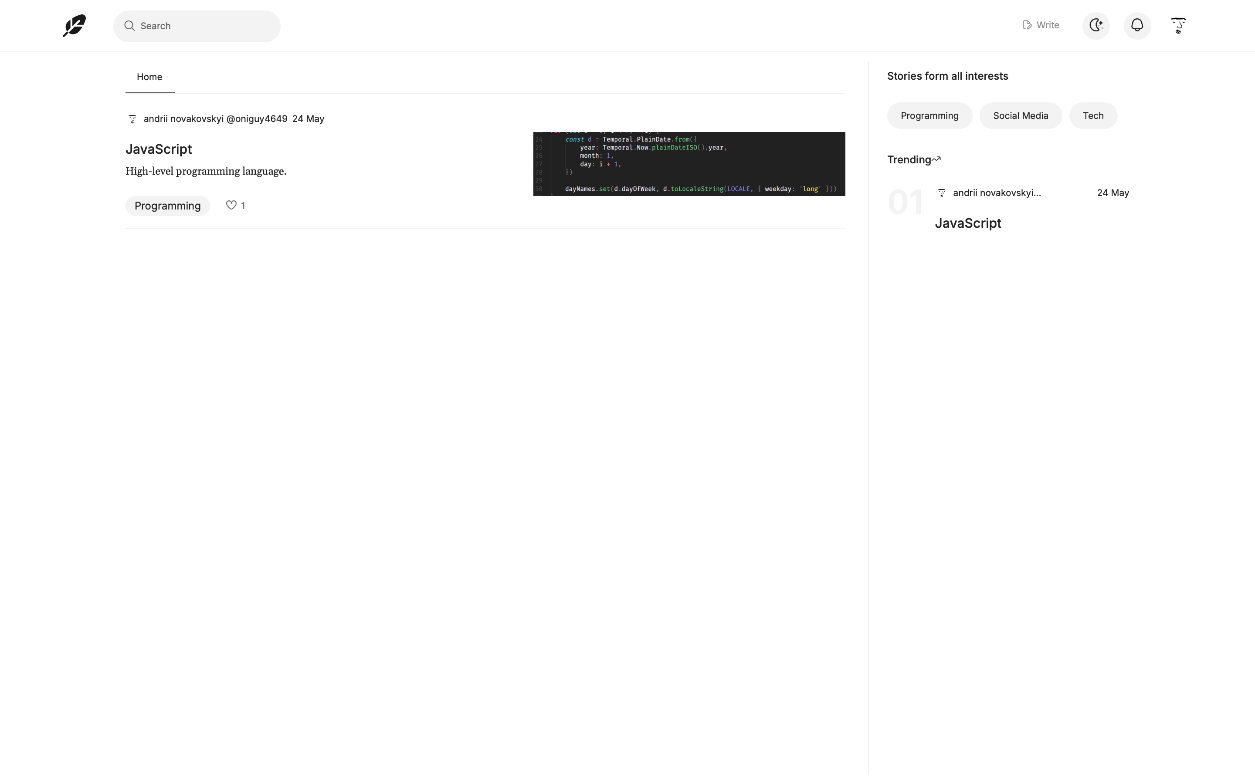


Рисунок 3.2.2

Сторінка пошуку (/search/:query) - результати пошуку статей за ключовими словами (рис. 3.2.3)

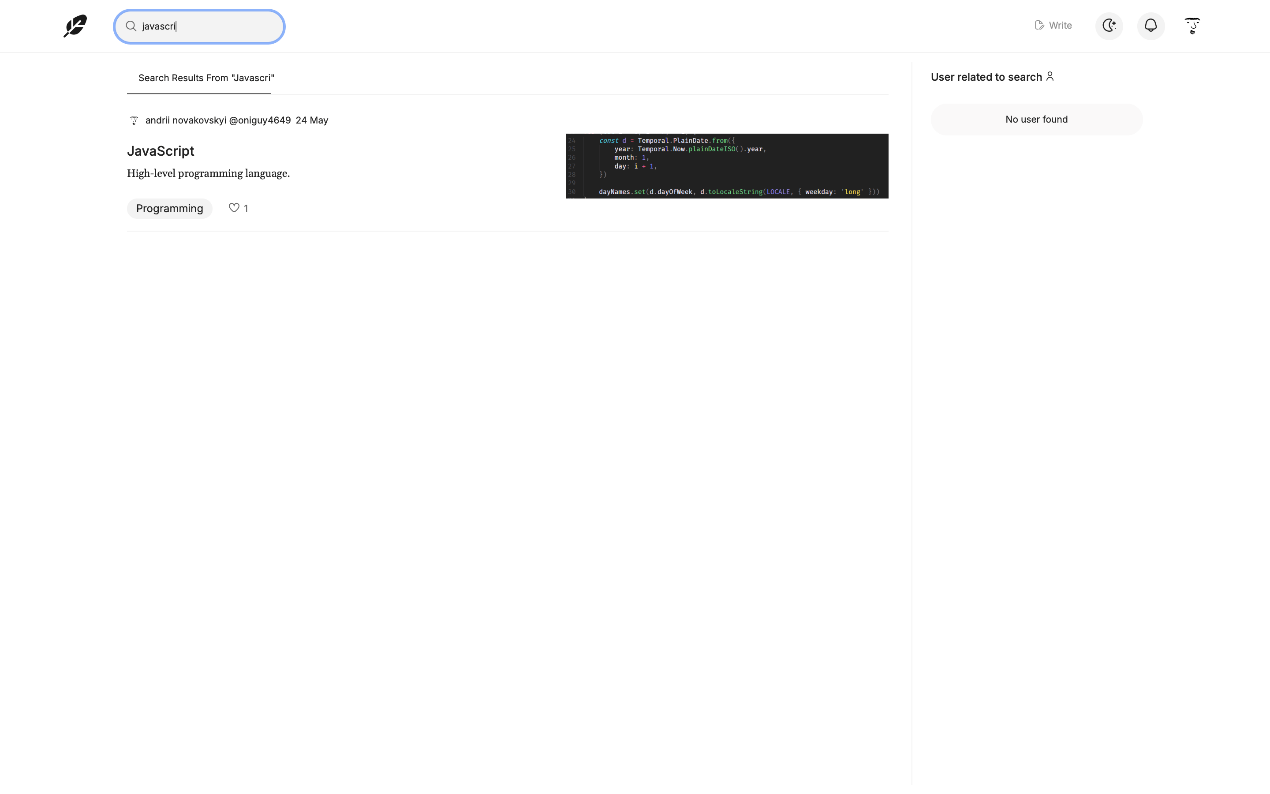


Рисунок 3.2.3

Сторінка профілю користувача (/user/:id) - публічний профіль автора з його статтями (рис. 3.2.4)

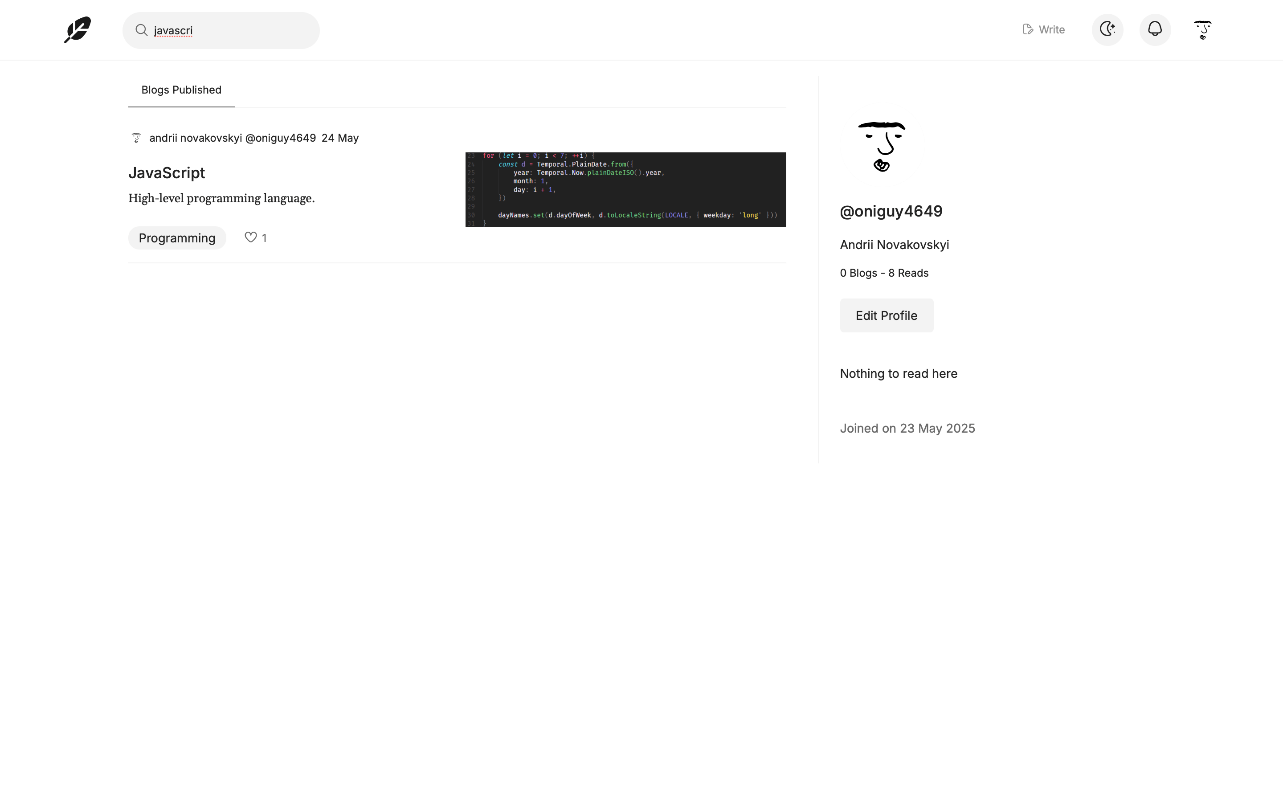


Рисунок 3.2.4

Сторінка входу (/signin) - форма авторизації користувачів (рис. 3.2.5)

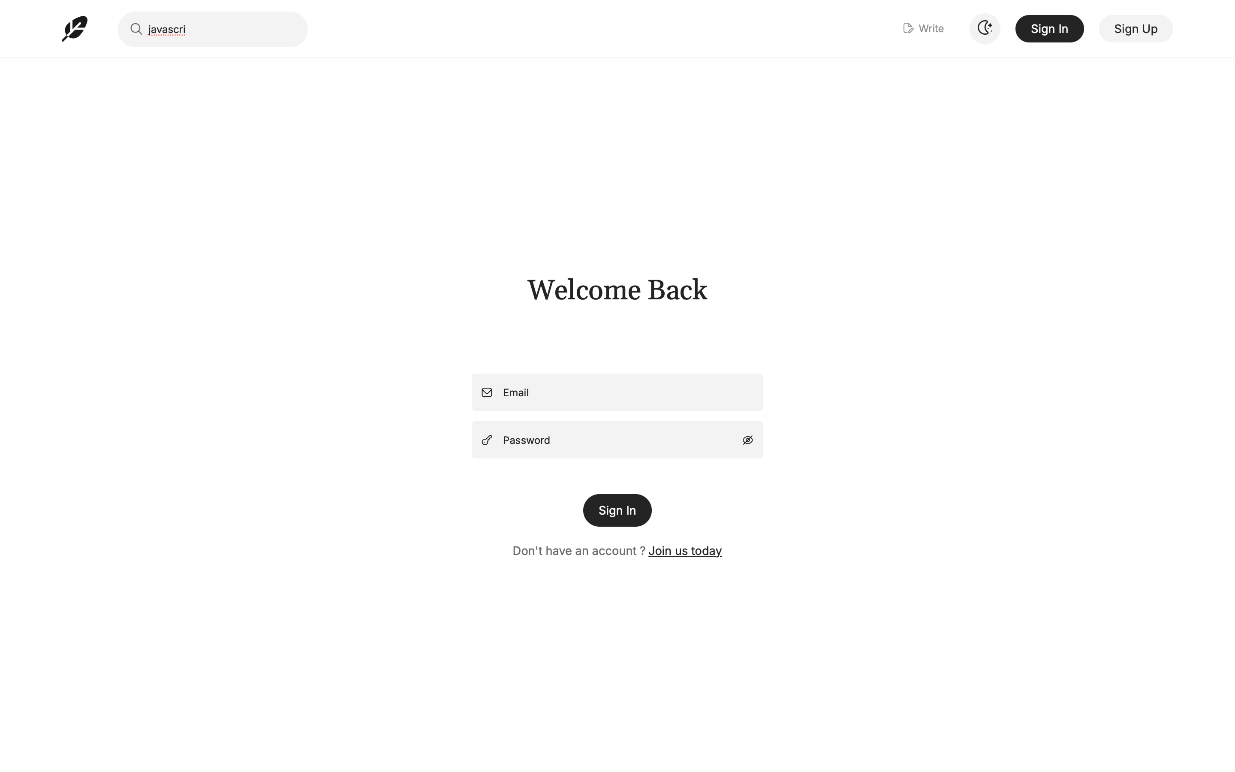


Рисунок 3.2.5

Сторінка реєстрації (/signup) - форма створення нового акаунту (рис. 3.2.6)

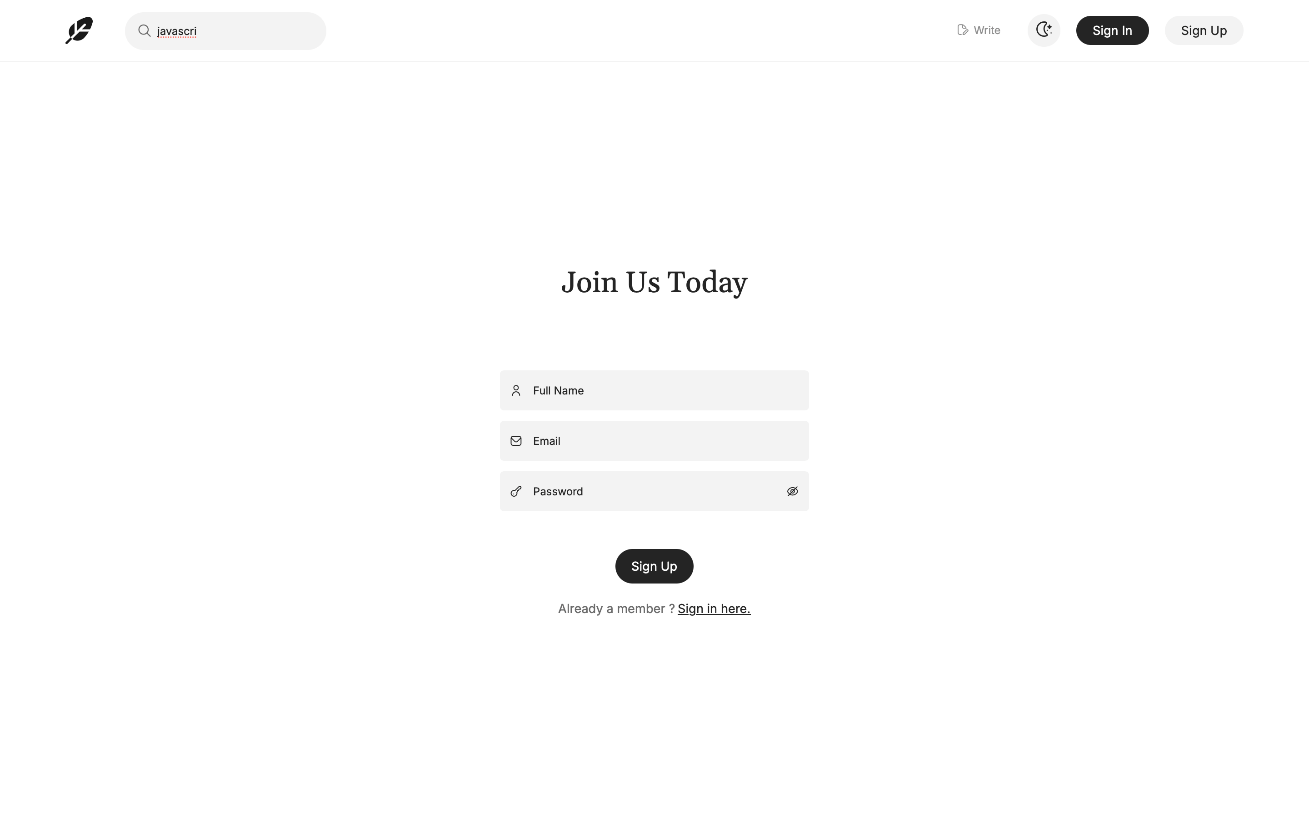


Рисунок 3.2.6

Панель управління (Dashboard), доступна лише авторизованим користувачам через маршрут /dashboard. Управління блогами (/dashboard/blogs) - перегляд, редагування та видалення власних статей (рис. 3.2.7)

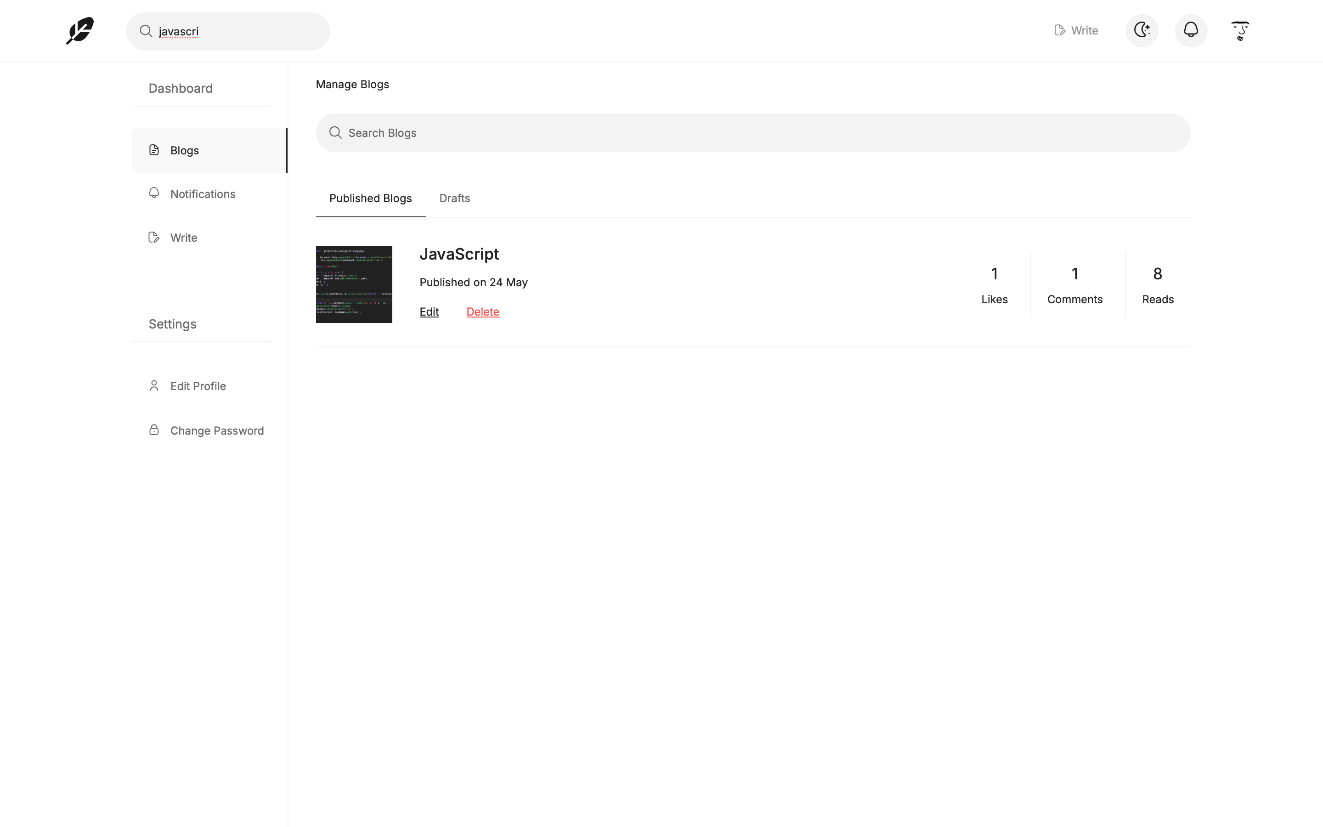


Рисунок 3.2.7

Сповіщення (/dashboard/notifications) - центр повідомлень про активність користувачів (рис. 3.2.8)

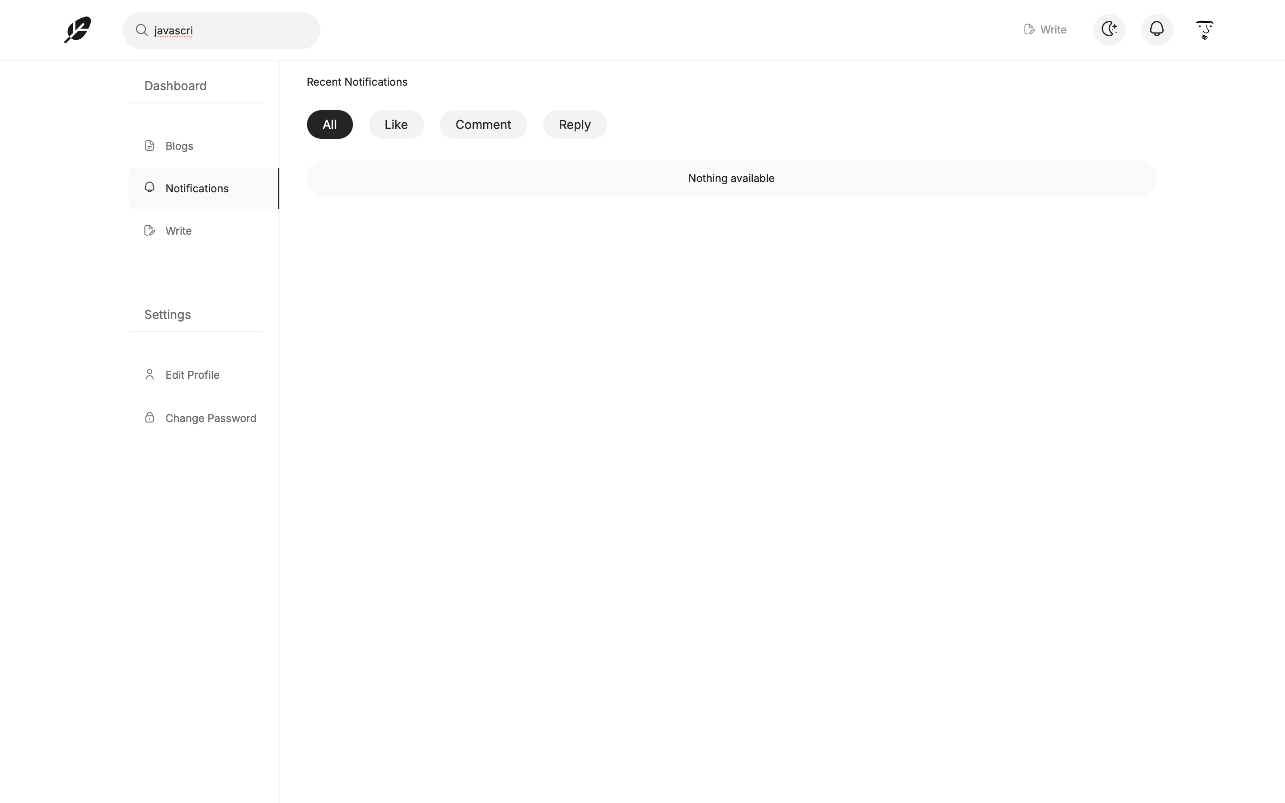


Рисунок 3.2.8

Редагування профілю (/settings/edit-profile) - зміна особистої інформації та аватара (рис. 3.2.9)

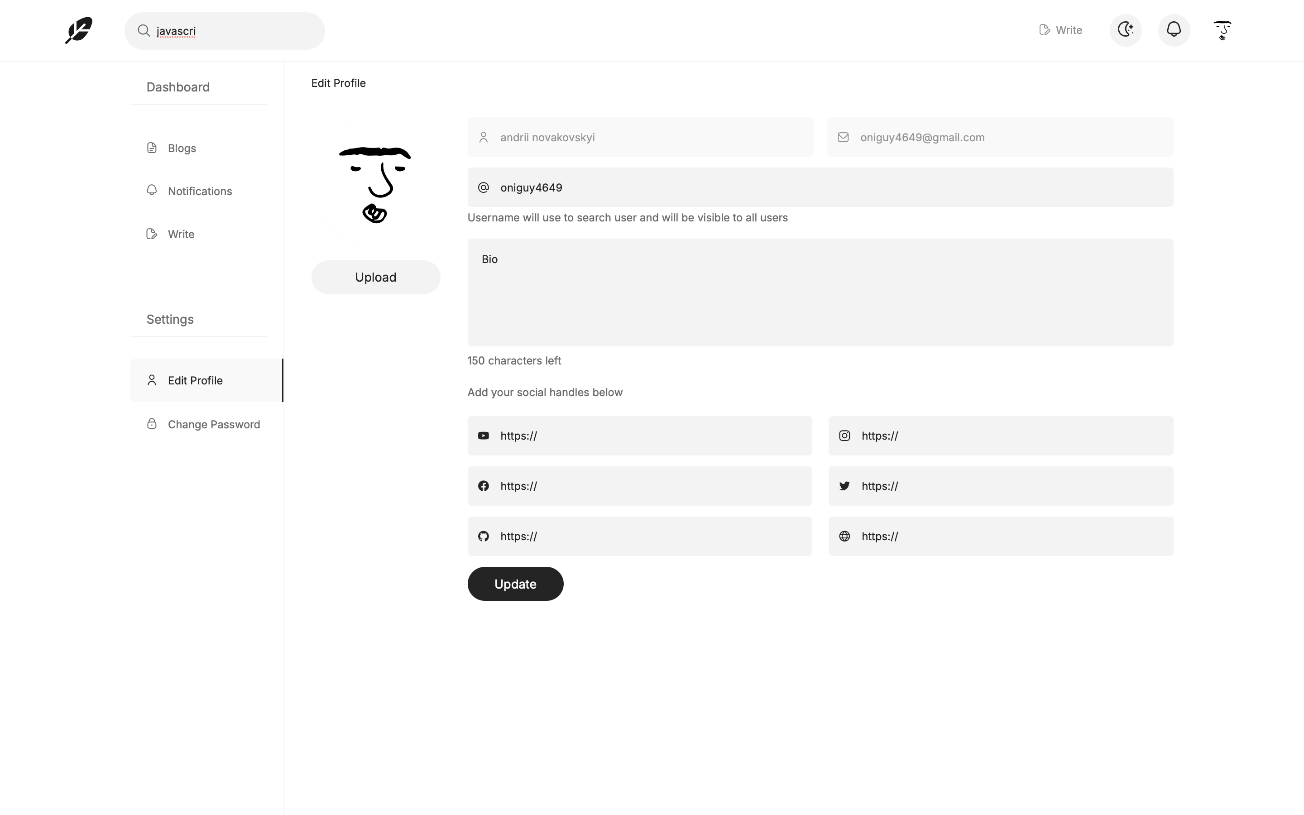


Рисунок 3.2.9

Зміна пароля (/settings/change-password) - форма оновлення пароля з валідацією (рис. 3.2.10)

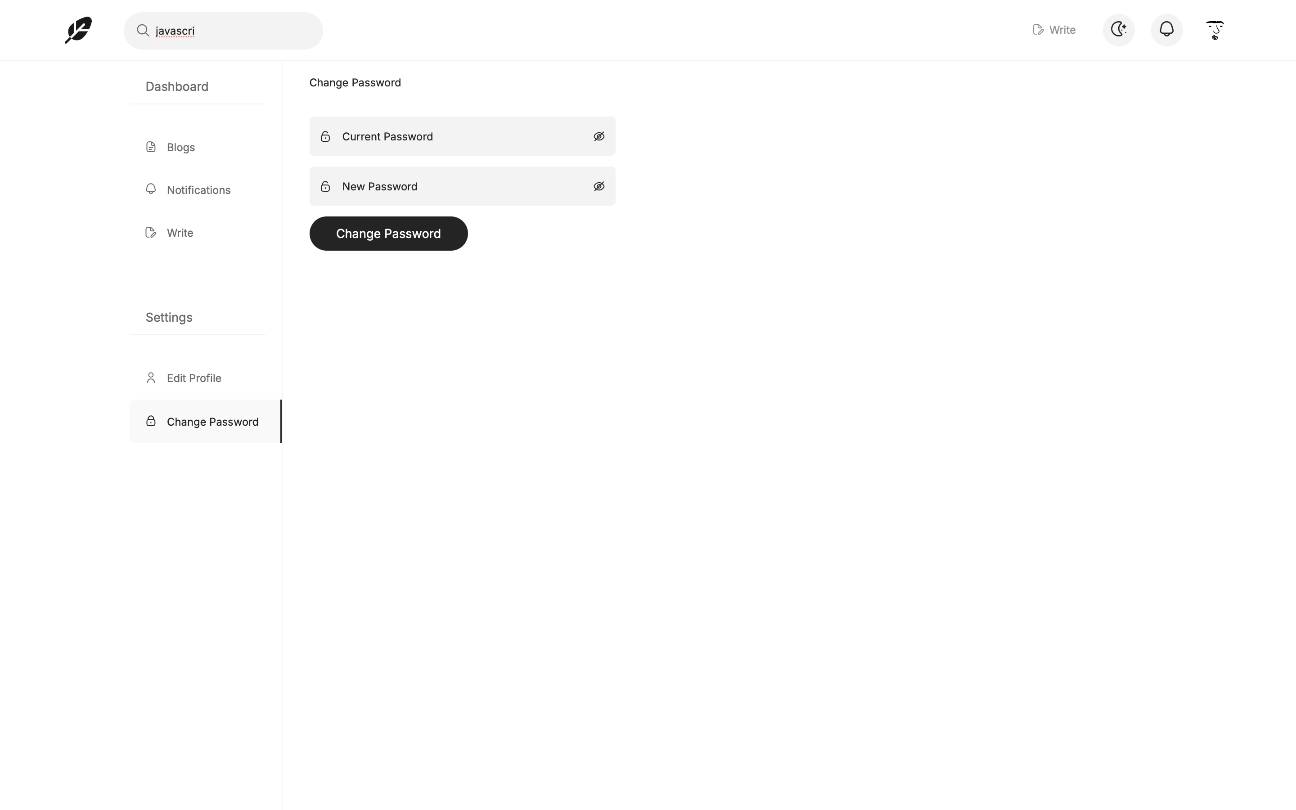


Рисунок 3.2.10

Створення нової статті (/editor) - повнофункціональний текстовий редактор імплементований за допомогою Editor.js (рис. 3.2.11)

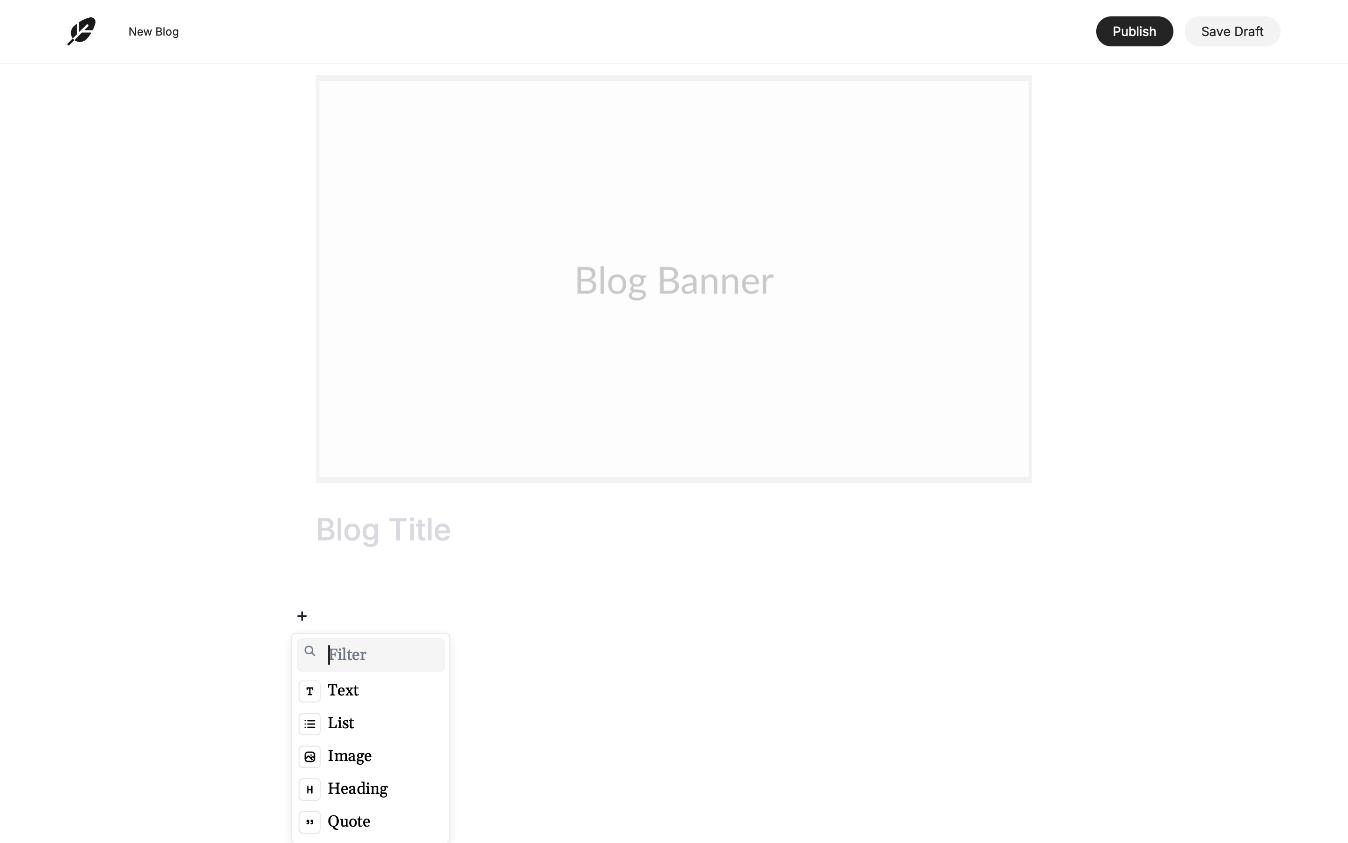


Рисунок 3.2.11

Редагування існуючої статті (/editor/:blog\_id) - режим редагування опублікованих статей (рис. 3.2.12)



Рисунок 3.2.12

Сторінка 404 (/\*) - відображається при переході на неіснуючі маршрути (рис. 3.2.13)

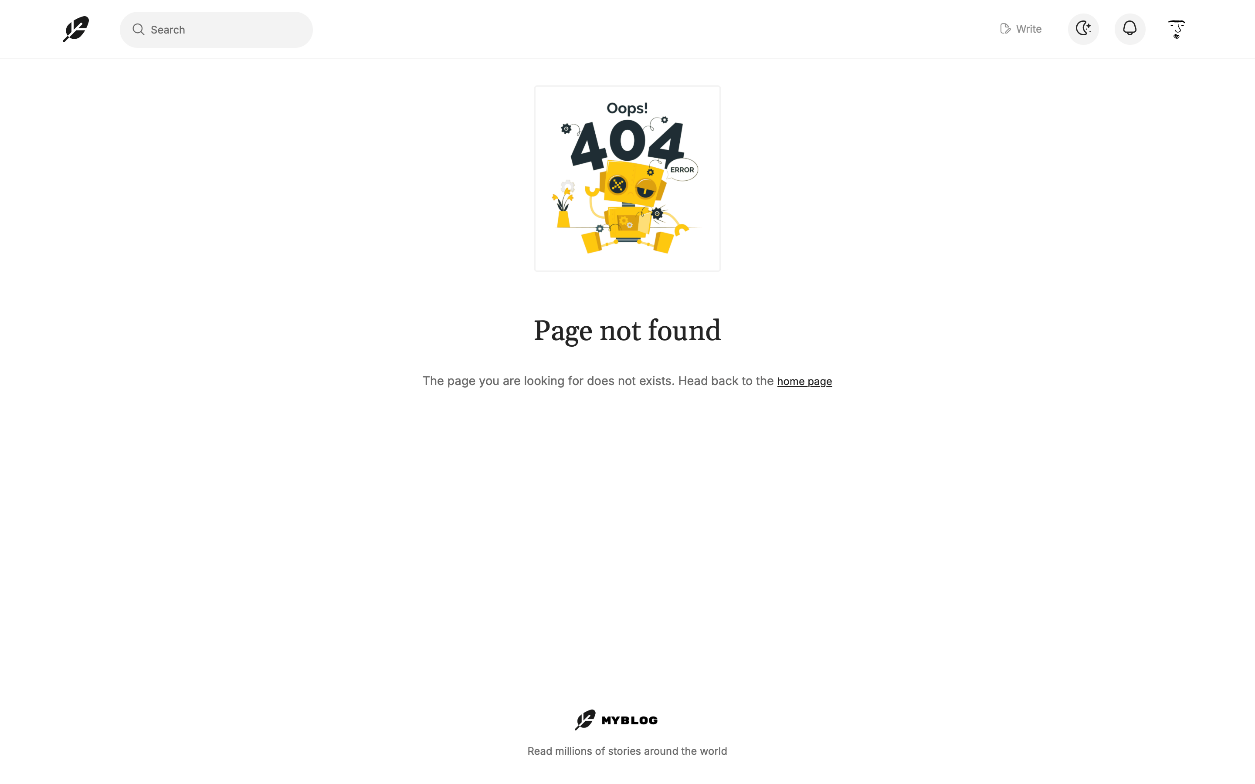


Рисунок 3.2.13

Веб-сайт використовує модульну архітектуру з розділенням на основні компоненти.

Navbar - головна навігаційна панель, присутня на всіх публічних сторінках (рис. 3.2.14)



Рисунок 3.2.14

SideNav - бічна навігація для панелі управління та налаштувань (рис. 3.2.15)

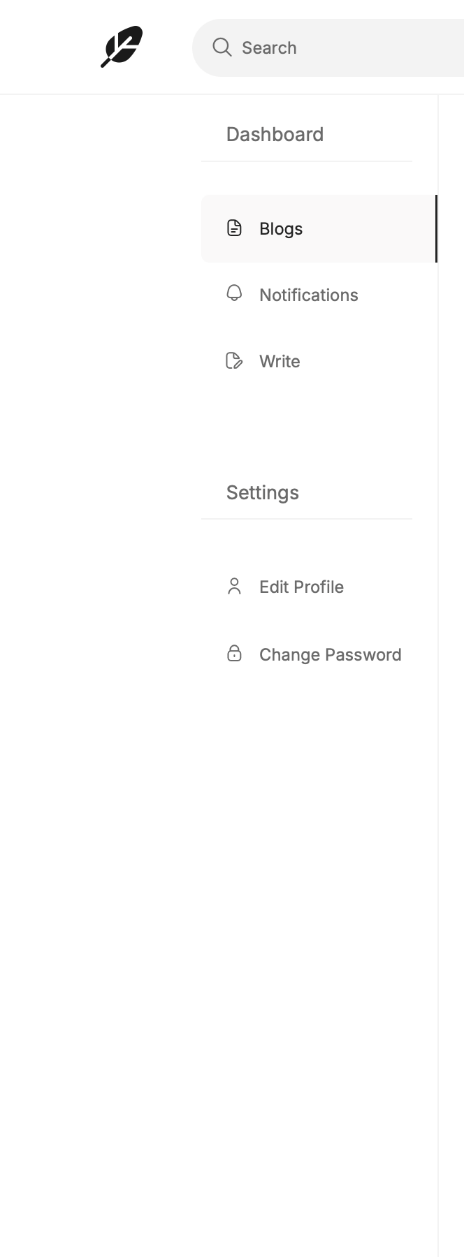


Рисунок 3.2.15

ThemeContext - управління темою оформлення (світла/темна) (рис.3.2.16)

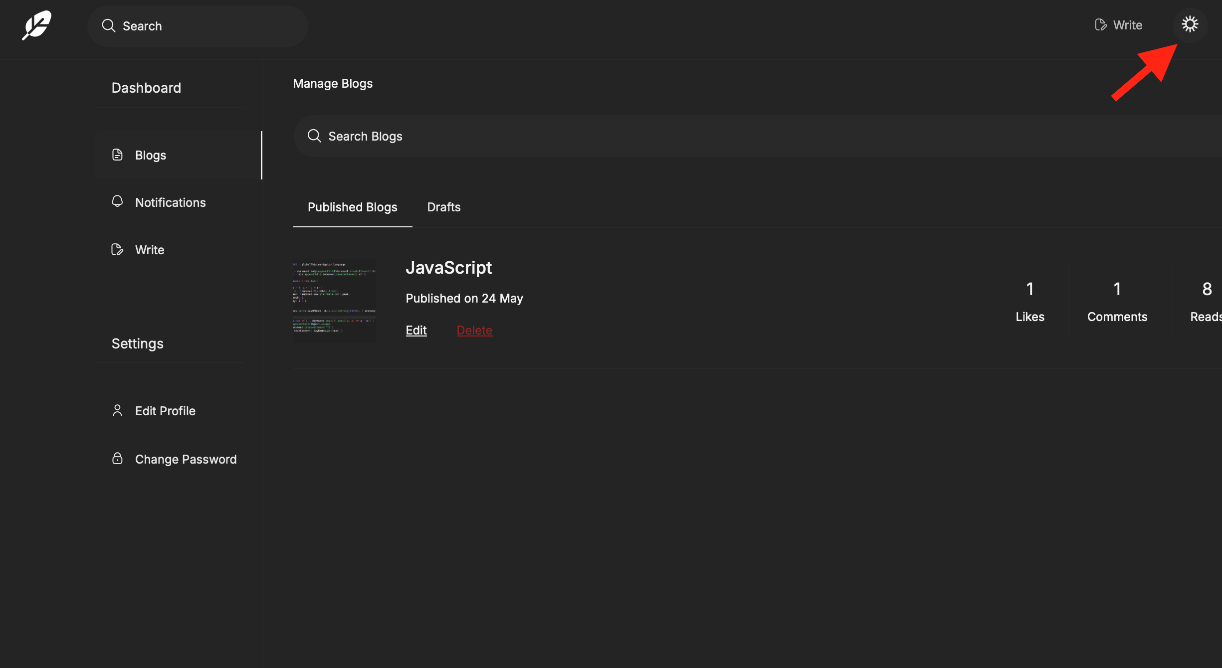


Рисунок 3.2.16

Така архітектура забезпечує масштабованість додатку та зручність використання.

## 3.3. Технічна реалізація веб-сайту

Технічна реалізація веб-сайту блогу базується на сучасному стеку технологій який скорочено називають MERN (MongoDB, Express.js, React.js, Node.js), що забезпечує високу продуктивність, масштабованість та зручність підтримки. Архітектура додатку побудована за принципом розділення клієнтської та серверної частин, що дозволяє незалежно розвивати та оптимізувати кожен компонент системи.

Система маршрутизації є основою навігації користувачів по веб-сайту. Вона реалізована з використанням React Router v6, що забезпечує декларативну маршрутизацію та підтримку вкладених маршрутів.

Головний компонент маршрутизації розташований у файлі App.tsx та використовує контекстні провайдери для управління глобальним станом додатку (рис. 3.3.1). Структура маршрутизації включає два основні контексти: ThemeContext для управління темою оформлення та UserContext для зберігання інформації про аутентифікованого користувача.

<ThemeContext.Provider value={{ theme, setTheme }}>

<UserContext.Provider value={{ userAuth, setUserAuth }}>

<Routes>

<Route path="/editor" element={<Editor />} />

<Route path="/editor/:blog\_id" element={<Editor />} />

<Route path="/" element={<Navbar />}>

<Route index element={<HomePage />} />

<Route path="dashboard" element={<SideNav />}>

<Route path="blogs" element={<ManageBlogs />} />

<Route path="notifications" element={<Notifications />} />

</Route>

<Route path="settings" element={<SideNav />}>

<Route path="edit-profile" element={<EditProfile />} />

<Route path="change-password" element={<ChangePassword />} />

</Route>

<Route path="signin" element={<UserAuthForm type="sign-in" />} />

<Route path="signup" element={<UserAuthForm type="sign-up" />} />

<Route path="search/:query" element={<SearchPage />} />

<Route path="user/:id" element={<ProfilePage />} />

<Route path="blog/:blog\_id" element={<BlogPage />} />

<Route path="\*" element={<PageNotFound />} />

</Route>

</Routes>

</UserContext.Provider>

</ThemeContext.Provider>

Маршрутизація організована ієрархічно з використанням вкладених маршрутів. Компонент Navbar виступає як layout-компонент для більшості публічних сторінок, забезпечуючи єдину навігаційну панель. Для адміністративних функцій використовується окремий layout-компонент SideNav, що створює бічну панель навігації.

Далі розберемо головну сторінку (Homepage). Головна сторінка представляє основний контент блогу та реалізована у файлі home.page.jsx (рис. 3.3.2). Сторінка відображає останні публікації, популярні статті та забезпечує інтерфейс для пошуку контенту.

<AnimationWrapper>

<section className="h-cover flex justify-center gap-10">

<div className="w-full">

<InPageNavigation routes={[pageState, "trending blogs"]} defaultHidden={["trending blogs"]}>

<>

{blogs == null ? <Loader /> : blogs.results.length ? blogs.results.map((blog, i) => (

<AnimationWrapper transition={{ duration: 1, delay: i \* 0.1 }} key={i}>

<BlogPostCard content={blog} author={blog.author.personal\_info} />

</AnimationWrapper>

)) : <NoDataMessage message="No blogs published" />}

<LoadMoreDataBtn state={blogs} fetchDataFun={pageState == "home" ? fetchLatestBlogs : fetchBlogsByCategory} />

</>

{trendingBlogs == null ? <Loader /> : trendingBlogs.length ? trendingBlogs.map((blog, i) => (

<AnimationWrapper transition={{ duration: 1, delay: i \* 0.1 }} key={i}>

<MinimalBlogPost blog={blog} index={i} />

</AnimationWrapper>

)) : <NoDataMessage message="No trending blogs" />}

</InPageNavigation>

</div>

<div className="min-w-[40%] lg:min-w-[400px] max-w-min border-l border-grey pl-8 pt-3 max-md:hidden">

<div className="flex flex-col gap-10">

<div>

<h1 className="font-medium text-xl mb-8">Tags</h1>

<div className="flex gap-3 flex-wrap">

{categories.map((category, i) => (

<button onClick={loadBlogByCategory} className={"tag " + (pageState == category ? " bg-black text-white " : " ")} key={i}>

{category}

</button>

))}

</div>

</div>

<div>

<h1 className="font-medium text-xl mb-8">Trending <i className="fi fi-rr-arrow-trend-up"></i></h1>

{trendingBlogs == null ? <Loader /> : trendingBlogs.length ? trendingBlogs.map((blog, i) => (

<AnimationWrapper transition={{ duration: 1, delay: i \* 0.1 }} key={i}>

<MinimalBlogPost blog={blog} index={i} />

</AnimationWrapper>

)) : <NoDataMessage message="No trending blogs" />}

</div>

</div>

</div>

</section>

</AnimationWrapper>

Компонент використовує хуки React для управління станом та виконання HTTP-запитів до API сервера. Реалізовано пагінацію для ефективного завантаження великої кількості статей та оптимізацію продуктивності через lazy loading зображень.

Далі розберемо сторінку окремої статті (BlogPage). BlogPage сторінка реалізована у файлі blog.page.jsx та відповідає за відображення повного контенту статті з можливістю коментування (рис. 3.3.3). Компонент отримує ідентифікатор статті з URL параметрів та виконує запит до API для завантаження контенту.

export const blogStructure = { title: "", des: "", conent: [], author: { personal\_info: {} }, banner: "", publishedAt: "" };

export const BlogContext = createContext({});

const BlogPage = () => {

let { blog\_id } = useParams();

const [blog, setBlog] = useState(blogStructure);

const [similarBlogs, setSimilrBlogs] = useState(null);

const [loading, setLoading] = useState(true);

const [islikedByUser, setLikedByUser] = useState(false);

const [commentsWrapper, setCommentsWrapper] = useState(false);

const [totalParentCommentsLoaded, setTotalParentCommentsLoaded] = useState(0);

let { title, content, banner, author: { personal\_info: { fullname, username: author\_username, profile\_img } }, publishedAt } = blog;

const fetchBlog = () => {

axios.post(import.meta.env.VITE\_SERVER\_DOMAIN + "/get-blog", { blog\_id }).then(async ({ data: { blog } }) => {

blog.comments = await fetchComments({ blog\_id: blog.\_id, setParentCommentCountFun: setTotalParentCommentsLoaded });

setBlog(blog);

axios.post(import.meta.env.VITE\_SERVER\_DOMAIN + "/search-blogs", {

tag: blog.tags[0], limit: 6, eliminate\_blog: blog\_id

}).then(({ data }) => setSimilrBlogs(data.blogs));

setLoading(false);

}).catch(err => { console.log(err); setLoading(false); });

};

useEffect(() => { resetStates(); fetchBlog(); }, [blog\_id]);

const resetStates = () => {

setBlog(blogStructure); setSimilrBlogs(null); setLoading(true);

setLikedByUser(false); setCommentsWrapper(false); setTotalParentCommentsLoaded(0);

};

return (

<AnimationWrapper>

{loading ? <Loader /> : (

<BlogContext.Provider value={{ blog, setBlog, islikedByUser, setLikedByUser, commentsWrapper, setCommentsWrapper, totalParentCommentsLoaded, setTotalParentCommentsLoaded }}>

<CommentsContainer />

<div className="max-w-[900px] center py-10 max-lg:px-[5vw]">

<img src={banner} className="aspect-video" />

<div className="mt-12">

<h2>{title}</h2>

<div className="flex max-sm:flex-col justify-between my-8">

<div className="flex gap-5 items-start">

<img src={profile\_img} className="w-12 h-12 rounded-full" />

<p className="capitalize">

{fullname}<br />@<Link to={`/user/${author\_username}`} className="underline">{author\_username}</Link>

</p>

</div>

<p className="text-dark-grey opacity-75 max-sm:mt-6 max-sm:ml-12 max-sm:pl-5">

Published on {getDay(publishedAt)}

</p>

</div>

</div>

<BlogInteraction />

<div className="my-12 font-gelasio blog-page-content">

{content[0].blocks.map((block, i) => (

<div key={i} className="my-4 md:my-8"><BlogContent block={block} /></div>

))}

</div>

<BlogInteraction />

{similarBlogs != null && similarBlogs.length ? (

<>

<h1 className="text-2xl mt-14 mb-10 font-medium">Similar Blogs</h1>

{similarBlogs.map((blog, i) => {

let { author: { personal\_info } } = blog;

return (

<AnimationWrapper key={i} transition={{ duration: 1, delay: i \* 0.08 }}>

<BlogPostCard content={blog} author={personal\_info} />

</AnimationWrapper>

);

})}

</>

) : " "}

</div>

</BlogContext.Provider>

)}

</AnimationWrapper>

);

};

Особливу увагу приділено SEO-оптимізації цієї сторінки. Динамічно генеруються мета-теги на основі контенту статті, включаючи заголовок, опис та Open Graph теги для соціальних мереж.

Далі розберемо редактор статей (Editor). Компонент редактора статей розташований у файлі editor.pages.jsx та забезпечує повнофункціональний інтерфейс для створення та редагування контенту (рис. 3.3.4). Редактор підтримує форматування тексту, вставку зображень та попередній перегляд статті.

import { useContext, useEffect, useState, createContext } from "react";

import { UserContext } from "../App";

import { Navigate, useParams } from "react-router-dom";

import BlogEditor from "../components/blog-editor.component";

import PublishForm from "../components/publish-form.component";

import Loader from "../components/loader.component";

import axios from "axios";

const blogStructure = { title: "", banner: "", conent: [], tags: [], des: "", author: { personal\_info: {} } };

export const EditorContext = createContext({});

const Editor = () => {

const { blog\_id } = useParams();

const [blog, setBlog] = useState(blogStructure);

const [editorState, setEditorState] = useState("editor");

const [textEditor, setTextEditor] = useState({ isReady: false });

const [loading, setLoading] = useState(true);

const { userAuth: { access\_token } } = useContext(UserContext);

useEffect(() => {

if (!blog\_id) return setLoading(false);

axios.post(import.meta.env.VITE\_SERVER\_DOMAIN + "/get-blog", { blog\_id, draft: true, mode: "edit" })

.then(({ data: { blog } }) => { setBlog(blog); setLoading(false); })

.catch(() => { setBlog(null); setLoading(false); });

}, []);

return (

<EditorContext.Provider value={{ blog, setBlog, editorState, setEditorState, textEditor, setTextEditor }}>

{access\_token === null ? <Navigate to="/signin" /> : loading ? <Loader /> : editorState == "editor" ? <BlogEditor /> : <PublishForm />}

</EditorContext.Provider>

);

};

Редактор використовує WYSIWYG бібліотеку Editor.js, процес редагування тексту чимось схожий на те як він працює в популярній програмі Notion, тому деякі користувачі будуть задоволені уже звичним для них workflow.

Далі розглянемо як працює система аутентифікації (UserAuthForm). Форми входу та реєстрації реалізовані в компоненті UserAuthForm у файлі userAuthForm.page.jsx (рис. 3.3.5). Компонент використовує пропс type для визначення режиму роботи між формами входу та реєстрації.

const UserAuthForm = ({ type }) => {

let { userAuth: { access\_token }, setUserAuth } = useContext(UserContext);

const userAuthThroughServer = (serverRoute, formData) => {

axios.post(import.meta.env.VITE\_SERVER\_DOMAIN + serverRoute, formData)

.then(({ data }) => {

storeInSession("user", JSON.stringify(data));

setUserAuth(data);

})

.catch(({ response }) => toast.error(response.data.error));

};

const handleSubmit = (e) => {

e.preventDefault();

let serverRoute = type == "sign-in" ? "/signin" : "/signup";

let emailRegex = /^\w+([\.-]?\w+)\*@\w+([\.-]?\w+)\*(\.\w{2,3})+$/;

let passwordRegex = /^(?=.\*\d)(?=.\*[a-z])(?=.\*[A-Z]).{6,20}$/;

let form = new FormData(formElement), formData = {};

for (let [key, value] of form.entries()) formData[key] = value;

let { fullname, email, password } = formData;

if (fullname && fullname.length < 3) return toast.error("Fullname must be at least 3 letters long");

if (!email.length) return toast.error("Enter Email");

if (!emailRegex.test(email)) return toast.error("Email is invalid");

if (!passwordRegex.test(password)) return toast.error("Password should be 6 to 20 characters long with a numeric, 1 lowercase and 1 uppercase letters");

userAuthThroughServer(serverRoute, formData);

};

return access\_token ? <Navigate to="/" /> : (

<AnimationWrapper keyValue={type}>

<section className="h-cover flex items-center justify-center">

<Toaster />

<form id="formElement" className="w-[80%] max-w-[400px]">

<h1 className="text-4xl font-gelasio capitalize text-center mb-24">{type == "sign-in" ? "Welcome back" : "Join us today"}</h1>

{type != "sign-in" ? <InputBox name="fullname" type="text" placeholder="Full Name" icon="fi-rr-user" /> : ""}

<InputBox name="email" type="email" placeholder="Email" icon="fi-rr-envelope" />

<InputBox name="password" type="password" placeholder="Password" icon="fi-rr-key" />

<button className="btn-dark center mt-14" type="submit" onClick={handleSubmit}>{type.replace("-", " ")}</button>

{type == "sign-in" ? (

<p className="mt-6 text-dark-grey text-xl text-center">Don't have an account ?<Link to="/signup" className="underline text-black text-xl ml-1">Join us today</Link></p>

) : (

<p className="mt-6 text-dark-grey text-xl text-center">Already a member ?<Link to="/signin" className="underline text-black text-xl ml-1">Sign in here.</Link></p>

)}

</form>

</section>

</AnimationWrapper>

);

};

Система аутентифікації включає валідацію електронної пошти, перевірку складності пароля та захист від CSRF атак. Після успішної аутентифікації користувач отримує JWT токен, який зберігається у localStorage для подальших API запитів.

Тепер поглянемо на навігаційний компонент (Navbar). Компонент навігації забезпечує єдиний інтерфейс для переміщення по сайту та розташований у директорії components (рис. 3.3.6). Navbar адаптується до стану аутентифікації користувача та відображає відповідні елементи меню.

const Navbar = () => {

const [searchBoxVisibility, setSearchBoxVisibility] = useState(false);

const [userNavPanel, setUserNavPanel] = useState(false);

const { theme, setTheme } = useContext(ThemeContext);

const navigate = useNavigate();

const { userAuth, userAuth: { access\_token, profile\_img, new\_notification\_available }, setUserAuth } = useContext(UserContext);

useEffect(() => {

if (access\_token) {

axios.get(import.meta.env.VITE\_SERVER\_DOMAIN + "/new-notification", { headers: { Authorization: `Bearer ${access\_token}` } })

.then(({ data }) => setUserAuth({ ...userAuth, ...data }))

.catch(console.log);

}

}, [access\_token]);

const handleUserNavPanel = () => setUserNavPanel(v => !v);

const handleSearch = e => { let q = e.target.value; if (e.keyCode == 13 && q.length) navigate(`/search/${q}`); };

const handleBlur = () => setTimeout(() => setUserNavPanel(false), 200);

const changeTheme = () => {

let newTheme = theme == "light" ? "dark" : "light";

setTheme(newTheme);

document.body.setAttribute("data-theme", newTheme);

storeInSession("theme", newTheme);

};

return <>

<nav className="navbar z-50">

<Link to="/" className="flex-none w-10">

<img src={theme == "light" ? darkLogo : lightLogo} className="w-full" />

</Link>

<div className={"absolute bg-white w-full left-0 top-full mt-0.5 border-b border-grey py-4 px-[5vw] md:border-0 md:block md:relative md:inset-0 md:p-0 md:w-auto md:show " + (searchBoxVisibility ? "show" : "hide")}>

<input type="text" placeholder="Search" className="w-full md:w-auto bg-grey p-4 pl-6 pr-[12%] md:pr-6 rounded-full placeholder:text-dark-grey md:pl-12" onKeyDown={handleSearch} />

<i className="fi fi-rr-search absolute right-[10%] md:pointer-events-none md:left-5 top-1/2 -translate-y-1/2 text-xl text-dark-grey"></i>

</div>

<div className="flex items-center gap-3 md:gap-6 ml-auto">

<button className="md:hidden bg-grey w-12 h-12 rounded-full flex items-center justify-center" onClick={() => setSearchBoxVisibility(v => !v)}>

<i className="fi fi-rr-search text-xl"></i>

</button>

<Link to="/editor" className="hidden md:flex gap-2 link">

<i className="fi fi-rr-file-edit"></i><p>Write</p>

</Link>

<button className="w-12 h-12 rounded-full bg-grey relative hover:bg-black/10" onClick={changeTheme}>

<i className={"fi fi-rr-" + (theme == "light" ? "moon-stars" : "sun") + " text-2xl block mt-1"}></i>

</button>

{access\_token ? <>

<Link to="/dashboard/notifications">

<button className="w-12 h-12 rounded-full bg-grey relative hover:bg-black/10">

<i className="fi fi-rr-bell text-2xl block mt-1"></i>

{new\_notification\_available && <span className="bg-red w-3 h-3 rounded-full absolute z-10 top-2 right-2"></span>}

</button>

</Link>

<div className="relative" onClick={handleUserNavPanel} onBlur={handleBlur}>

<button className="w-12 h-12 mt-1">

<img src={profile\_img} className="w-full h-full object-cover rounded-full" />

</button>

{userNavPanel && <UserNavigationPanel />}

</div>

</> : <>

<Link className="btn-dark py-2" to="/signin">Sign In</Link>

<Link className="btn-light py-2 hidden md:block" to="/signup">Sign Up</Link>

</>}

</div>

</nav>

<Outlet />

</>;

};

Компонент включає адаптивний дизайн для мобільних пристроїв з використанням hamburger menu та підтримку темної теми оформлення.

Далі розберемо як працює бічна навігація (SideNav). Компонент SideNav створює інтерфейс адміністративної панелі з бічною навігацією (рис. 3.3.7). Він використовується для сторінок управління контентом та налаштувань профілю.

import { useContext, useEffect, useRef, useState } from "react"

import { Outlet, Navigate, NavLink } from "react-router-dom"

import { UserContext } from "../App"

const SideNav = () => {

let { userAuth: { access\_token, new\_notification\_available } } = useContext(UserContext)

let page = location.pathname.split("/")[2]

let [pageState, setPageState] = useState(page.replace("-", " "))

let [showSideNav, setShowSideNav] = useState(false)

let activeTabLine = useRef(), sideBarIconTab = useRef(), pageStateTab = useRef()

const changePageState = (e) => {

let { offsetWidth, offsetLeft } = e.target

activeTabLine.current.style.width = offsetWidth + "px"

activeTabLine.current.style.left = offsetLeft + "px"

setShowSideNav(e.target == sideBarIconTab.current)

}

useEffect(() => {

setShowSideNav(false)

pageStateTab.current.click()

}, [pageState])

return access\_token === null ? <Navigate to="/signin" /> : <>

<section className="relative flex gap-10 py-0 m-0 max-md:flex-col">

<div className="sticky top-[80px] z-30">

<div className="md:hidden bg-white py-1 border-b border-grey flex flex-nowrap overflow-x-auto">

<button ref={sideBarIconTab} className="p-5 capitalize" onClick={changePageState}><i className="fi fi-rr-bars-staggered pointer-events-none" /></button>

<button ref={pageStateTab} className="p-5 capitalize" onClick={changePageState}>{pageState}</button>

<hr ref={activeTabLine} className="absolute bottom-0 duration-500" />

</div>

<div className={"min-w-[200px] h-[calc(100vh-80px-60px)] md:h-cover md:sticky top-24 overflow-y-auto p-6 md:pr-0 md:border-grey md:border-r absolute max-md:top-[64px] bg-white max-md:w-[calc(100%+80px)] max-md:px-16 max-md:-ml-7 duration-500 " + (!showSideNav ? "max-md:opacity-0 max-md:pointer-events-none" : "opacity-100 pointer-events-auto")}>

<h1 className="text-xl text-dark-grey mb-3">Dashboard</h1>

<hr className="border-grey -ml-6 mb-8 mr-6" />

<NavLink to="/dashboard/blogs" onClick={(e) => setPageState(e.target.innerText)} className="sidebar-link"><i className="fi fi-rr-document" />Blogs</NavLink>

<NavLink to="/dashboard/notifications" onClick={(e) => setPageState(e.target.innerText)} className="sidebar-link">

<div className="relative">

<i className="fi fi-rr-bell" />

{new\_notification\_available && <span className="bg-red w-2 h-2 rounded-full absolute z-10 top-0 right-0" />}

</div>

Notifications

</NavLink>

<NavLink to="/editor" onClick={(e) => setPageState(e.target.innerText)} className="sidebar-link"><i className="fi fi-rr-file-edit" />Write</NavLink>

<h1 className="text-xl text-dark-grey mt-20 mb-3">Settings</h1>

<hr className="border-grey -ml-6 mb-8 mr-6" />

<NavLink to="/settings/edit-profile" onClick={(e) => setPageState(e.target.innerText)} className="sidebar-link"><i className="fi fi-rr-user" />Edit Profile</NavLink>

<NavLink to="/settings/change-password" onClick={(e) => setPageState(e.target.innerText)} className="sidebar-link"><i className="fi fi-rr-lock" />Change Password</NavLink>

</div>

</div>

<div className="max-md:-mt-8 mt-5 w-full"><Outlet /></div>

</section>

</>

}

Розлянемо як працює конфігурація сервера (server.js), всього коду не буде так як він більше ніж 900 рядків коду (!!!).

Серверна частина реалізована з використанням Express.js фреймворку та розташована у файлі server.js (рис. 3.3.8). Сервер забезпечує RESTful API для взаємодії з клієнтською частиною та інтеграцію з MongoDB базою даних.

const server = express();

let PORT = 3000;

let emailRegex = /^\w+([\.-]?\w+)\*@\w+([\.-]?\w+)\*(\.\w{2,3})+$/;

let passwordRegex = /^(?=.\*\d)(?=.\*[a-z])(?=.\*[A-Z]).{6,20}$/;

server.use(express.json());

server.use(cors());

const storage = multer.diskStorage({

destination: (req, file, cb) => cb(null, "uploads/"),

filename: (req, file, cb) => cb(null, file.fieldname + "-" + Date.now() + "-" + Math.round(Math.random() \* 1e9) + "." + file.originalname.split(".").pop())

});

const upload = multer({ storage });

server.use("/uploads", express.static("uploads"));

mongoose.connect(process.env.DB\_LOCATION, { autoIndex: true });

const verifyJWT = (req, res, next) => {

const token = req.headers["authorization"]?.split(" ")[1];

if (!token) return res.status(401).json({ error: "No access token" });

jwt.verify(token, process.env.SECRET\_ACCESS\_KEY, (err, user) => {

if (err) return res.status(403).json({ error: "Access token is invalid" });

req.user = user.id;

next();

});

};

const formatDatatoSend = (user) => ({

access\_token: jwt.sign({ id: user.\_id }, process.env.SECRET\_ACCESS\_KEY),

profile\_img: user.personal\_info.profile\_img,

username: user.personal\_info.username,

fullname: user.personal\_info.fullname

});

const generateUsername = async (email) => {

let username = email.split("@")[0];

if (await User.exists({ "personal\_info.username": username })) username += nanoid().substring(0, 5);

return username;

};

server.post("/signup", (req, res) => {

let { fullname, email, password } = req.body;

if (fullname.length < 3) return res.status(403).json({ error: "Fullname must be at least 3 letters long" });

if (!email.length) return res.status(403).json({ error: "Enter Email" });

if (!emailRegex.test(email)) return res.status(403).json({ error: "Email is invalid" });

if (!passwordRegex.test(password)) return res.status(403).json({ error: "Password should be 6 to 20 characters long with a numeric, 1 lowercase and 1 uppercase letters" });

bcrypt.hash(password, 10, async (err, hashed\_password) => {

let username = await generateUsername(email);

let user = new User({ personal\_info: { fullname, email, password: hashed\_password, username } });

user.save().then(u => res.status(200).json(formatDatatoSend(u)))

.catch(err => res.status(err.code == 11000 ? 500 : 500).json({ error: err.code == 11000 ? "Email already exists" : err.message }));

});

});

server.post("/signin", (req, res) => {

let { email, password } = req.body;

User.findOne({ "personal\_info.email": email }).then(user => {

if (!user) return res.status(403).json({ error: "Email not found" });

bcrypt.compare(password, user.personal\_info.password, (err, result) => {

if (err) return res.status(403).json({ error: "Error occured while login please try again" });

if (!result) return res.status(403).json({ error: "Incorrect password" });

return res.status(200).json(formatDatatoSend(user));

});

}).catch(err => res.status(500).json({ error: err.message }));

});

server.post("/change-password", verifyJWT, (req, res) => {

let { currentPassword, newPassword } = req.body;

if (!passwordRegex.test(currentPassword) || !passwordRegex.test(newPassword)) return res.status(403).json({ error: "Password should be 6 to 20 characters long with a numeric, 1 lowercase and 1 uppercase letters" });

User.findOne({ \_id: req.user }).then(user => {

bcrypt.compare(currentPassword, user.personal\_info.password, (err, result) => {

if (err) return res.status(500).json({ error: "Some error occured while changing the password, please try again later" });

if (!result) return res.status(403).json({ error: "Incorrect current password" });

bcrypt.hash(newPassword, 10, (err, hashed\_password) => {

User.findOneAndUpdate({ \_id: req.user }, { "personal\_info.password": hashed\_password })

.then(() => res.status(200).json({ status: "password changed" }))

.catch(err => res.status(500).json({ error: "Some error occured while saving new password, please try again later" }));

});

});

}).catch(err => res.status(500).json({ error: "User not found" }));

});

server.post("/latest-blogs", (req, res) => {

let { page } = req.body;

Blog.find({ draft: false }).populate("author", "personal\_info.profile\_img personal\_info.username personal\_info.fullname -\_id")

.sort({ publishedAt: -1 }).select("blog\_id title des banner activity tags publishedAt -\_id")

.skip((page - 1) \* 5).limit(5).then(blogs => res.status(200).json({ blogs }))

.catch(err => res.status(500).json({ error: err.message }));

});

server.post("/all-latest-blogs-count", (req, res) => {

Blog.countDocuments({ draft: false }).then(count => res.status(200).json({ totalDocs: count }))

.catch(err => res.status(500).json({ error: err.message }));

});

server.get("/trending-blogs", (req, res) => {

Blog.find({ draft: false }).populate("author", "personal\_info.profile\_img personal\_info.username personal\_info.fullname -\_id")

.sort({ "activity.total\_read": -1, "activity.total\_likes": -1, publishedAt: -1 })

.select("blog\_id title publishedAt -\_id").limit(5).then(blogs => res.status(200).json({ blogs }))

.catch(err => res.status(500).json({ error: err.message }));

});

server.post("/search-blogs", (req, res) => {

let { tag, query, author, page, limit, eliminate\_blog } = req.body;

let findQuery;

if (tag) findQuery = { tags: tag, draft: false, blog\_id: { $ne: eliminate\_blog } };

else if (query) findQuery = { draft: false, title: new RegExp(query, "i") };

else if (author) findQuery = { author, draft: false };

Blog.find(findQuery).populate("author", "personal\_info.profile\_img personal\_info.username personal\_info.fullname -\_id")

.sort({ publishedAt: -1 }).select("blog\_id title des banner activity tags publishedAt -\_id")

.skip((page - 1) \* (limit || 2)).limit(limit || 2).then(blogs => res.status(200).json({ blogs }))

.catch(err => res.status(500).json({ error: err.message }));

});

server.listen(PORT, () => console.log("listening on port -> " + PORT));

Сервер включає middleware для обробки CORS запитів, парсингу JSON даних, автентифікації JWT токенів та логування запитів. Реалізовано обробку помилок та валідацію вхідних даних для забезпечення безпеки додатку.

Моделі даних визначені з використанням Mongoose ODM та розташовані у директорії Schema (рис. 3.3.9). Основні схеми включають модель користувача, статті блогу, коментарів та категорій.

const blogSchema = mongoose.Schema({

blog\_id: { type: String, required: true, unique: true },

title: { type: String, required: true },

banner: { type: String },

des: { type: String, maxlength: 200 },

content: { type: [] },

tags: { type: [String] },

author: { type: Schema.Types.ObjectId, required: true, ref: "users" },

activity: {

total\_likes: { type: Number, default: 0 },

total\_comments: { type: Number, default: 0 },

total\_reads: { type: Number, default: 0 },

total\_parent\_comments: { type: Number, default: 0 }

},

comments: { type: [Schema.Types.ObjectId], ref: "comments" },

draft: { type: Boolean, default: false }

}, { timestamps: { createdAt: "publishedAt" } });

const commentSchema = mongoose.Schema({

blog\_id: { type: Schema.Types.ObjectId, required: true, ref: "blogs" },

blog\_author: { type: Schema.Types.ObjectId, required: true, ref: "blogs" },

comment: { type: String, required: true },

children: { type: [Schema.Types.ObjectId], ref: "comments" },

commented\_by: { type: Schema.Types.ObjectId, require: true, ref: "users" },

isReply: { type: Boolean, default: false },

parent: { type: Schema.Types.ObjectId, ref: "comments" }

}, { timestamps: { createdAt: "commentedAt" } });

const notificationSchema = mongoose.Schema({

type: { type: String, enum: ["like", "comment", "reply"], required: true },

blog: { type: Schema.Types.ObjectId, required: true, ref: "blogs" },

notification\_for: { type: Schema.Types.ObjectId, required: true, ref: "users" },

user: { type: Schema.Types.ObjectId, required: true, ref: "users" },

comment: { type: Schema.Types.ObjectId, ref: "comments" },

reply: { type: Schema.Types.ObjectId, ref: "comments" },

replied\_on\_comment: { type: Schema.Types.ObjectId, ref: "comments" },

seen: { type: Boolean, default: false }

}, { timestamps: true });

let profile\_imgs\_name\_list = ["Garfield", "Tinkerbell", "Annie", "Loki", "Cleo", "Angel", "Bob", "Mia", "Coco", "Gracie", "Bear", "Bella", "Abby", "Harley", "Cali", "Leo", "Luna", "Jack", "Felix", "Kiki"];

let profile\_imgs\_collections\_list = ["notionists-neutral", "adventurer-neutral", "fun-emoji"];

const userSchema = mongoose.Schema({

personal\_info: {

fullname: { type: String, lowercase: true, required: true, minlength: [3, "fullname must be 3 letters long"] },

email: { type: String, required: true, lowercase: true, unique: true },

password: String,

username: { type: String, minlength: [3, "Username must be 3 letters long"], unique: true },

bio: { type: String, maxlength: [200, "Bio should not be more than 200"], default: "" },

profile\_img: {

type: String,

default: () => `https://api.dicebear.com/6.x/${profile\_imgs\_collections\_list[Math.floor(Math.random() \* profile\_imgs\_collections\_list.length)]}/svg?seed=${profile\_imgs\_name\_list[Math.floor(Math.random() \* profile\_imgs\_name\_list.length)]}`

}

},

social\_links: {

youtube: { type: String, default: "" },

instagram: { type: String, default: "" },

facebook: { type: String, default: "" },

twitter: { type: String, default: "" },

github: { type: String, default: "" },

website: { type: String, default: "" }

},

account\_info: {

total\_posts: { type: Number, default: 0 },

total\_reads: { type: Number, default: 0 }

},

blogs: { type: [Schema.Types.ObjectId], ref: "blogs", default: [] }

}, { timestamps: { createdAt: "joinedAt" } });

Схеми включають валідацію полів, індекси для оптимізації запитів та middleware для автоматичної обробки даних перед збереженням.

Система аналітики інтегрована через PostHog з додатковими custom events для відстеження поведінки користувачів (рис. 3.3.10). SEO оптимізація включає динамічне генерування мета-тегів, створення XML sitemap та структурованих даних.

Така технічна реалізація забезпечує створення повнофункціонального, масштабованого та SEO-оптимізованого веб-сайту блогу з сучасним користувацьким інтерфейсом та надійною серверною архітектурою.

## Висновки до розділу 3

У розділі 3 «Реалізація веб-сайту блогу з інтеграцією SEO оптимізації та аналітики» представлено процес розробки сучасної блог-платформи з використанням передових веб-технологій. Цей розділ охоплює детальний опис архітектурних рішень, структури проєкту та реалізації ключових функціональних компонентів системи.

Архітектура проєкту побудована за принципом розділення клієнтської та серверної частин, що забезпечує високу масштабованість та можливості для незалежного розвитку компонентів. Серверна частина реалізована на базі Express.js з використанням MongoDB для зберігання даних, що створює надійну основу для обробки запитів та управління контентом. Структура backend включає папку Schema для моделей даних, директорію uploads для завантажених медіафайлів та головний файл server.js, який координує роботу всієї серверної логіки.

Клієнтська частина розроблена з використанням React та TypeScript, що забезпечує типобезпечність коду та покращену продуктивність розробки. Структура frontend організована у логічні модулі з чітким розділенням компонентів, сторінок та загальних ресурсів. Папка components містить перевикористовувані елементи інтерфейсу, директорія pages об'єднує основні розділи додатку, а папка common зберігає спільні утиліти та константи. Конфігураційні файли включають налаштування для Vite, Tailwind CSS, PostCSS та ESLint, що забезпечує оптимізовану збірку та підтримку якості коду.

Інтеграція SEO-оптимізації реалізована на рівні архітектури через серверний рендеринг критично важливих елементів та автоматичну генерацію метаданих для кожної сторінки. Система автоматично створює семантичну розмітку, оптимізує зображення та забезпечує швидке завантаження сторінок, що позитивно впливає на позиції у пошукових системах.

Аналітичні можливості інтегровано через спеціалізовані компоненти, які відстежують користувацьку активність та збирають дані про ефективність контенту. Система забезпечує детальну статистику відвідуваності, аналіз поведінки читачів та метрики залученості аудиторії.

Особливу увагу приділено забезпеченню адаптивності інтерфейсу через використання Tailwind CSS, що дозволяє створювати responsive дизайн з мінімальними зусиллями. Система автоматично адаптується до різних розмірів екрану та забезпечує оптимальний користувацький досвід на всіх типах пристроїв.

Безпека системи забезпечується через імплементацію сучасних практик захисту даних, включаючи валідацію вхідних параметрів, захист від основних типів атак та безпечне зберігання користувацької інформації. Модульна архітектура дозволяє легко масштабувати систему та додавати нові функціональні можливості без порушення існуючої логіки.

Використання TypeScript у поєднанні з сучасними інструментами розробки забезпечує високу якість коду, зменшує кількість помилок на етапі розробки та полегшує процес підтримки проєкту. Налаштування автоматичної збірки через Vite значно прискорює процес розробки та оптимізує фінальний код для продакшн-середовища.

Результатом реалізації стала повнофункціональна блог-платформа, яка поєднує сучасні технологічні рішення з інтуїтивним користувацьким інтерфейсом. Система демонструє високу продуктивність, надійність та готовність до промислового використання, створюючи міцну основу для ефективного управління контентом та взаємодії з аудиторією у сучасному цифровому просторі.

# ВИСНОВКИ

Розробка веб-сайту блогу з інтеграцією SEO оптимізації та аналітики представляє собою комплексне технічне завдання, яке вимагає глибокого розуміння сучасних технологій веб-розробки та принципів цифрового маркетингу. У ході виконання дипломного проєкту було успішно реалізовано повнофункціональну блогову платформу, яка відповідає високим стандартам користувацького досвіду та забезпечує ефективну присутність у пошукових системах.

Основною метою проєкту було створення інноваційної блогової платформи, яка поєднує зручність управління контентом з потужними інструментами пошукової оптимізації та детальної аналітики. Для досягнення цієї мети було застосовано сучасний технологічний стек, що включає React для інтерактивного фронтенду, Express.js для серверної логіки, MongoDB з Mongoose для гнучкого управління даними та Tailwind CSS для адаптивного дизайну.

Результатом виконання проєкту став створений високофункціональний веб-сайт блогу з комплексною системою SEO оптимізації. Платформа забезпечує авторам інтуїтивні інструменти для створення та публікації контенту, включаючи розширений текстовий редактор, систему управління мультимедійним контентом та можливість попереднього перегляду публікацій. Читачі отримують оптимізований досвід споживання контенту з швидкою навігацією, адаптивним дизайном та зручним пошуком по матеріалах.

Інтеграція SEO оптимізації забезпечує максимальну видимість контенту у пошукових системах завдяки серверному рендерингу, оптимізації швидкості завантаження, структуруванню даних та комплексному налаштуванню метатегів. Система аналітики надає авторам детальну інформацію про ефективність їхнього контенту, поведінку аудиторії та джерела трафіку, що дозволяє приймати обґрунтовані рішення щодо контент-стратегії.

Створена платформа демонструє значні переваги порівняно з традиційними блоговими рішеннями. Завдяки використанню React архітектури, система забезпечує високу продуктивність та плавну взаємодію користувачів з інтерфейсом. Інтеграція Next.js для серверного рендерингу гарантує швидке індексування контенту пошуковими роботами та покращені показники Core Web Vitals, що критично важливо для SEO ранжування.

Проєкт підтверджує ефективність застосування сучасних JavaScript технологій для створення комплексних веб-додатків з високими вимогами до продуктивності та SEO. Використання MongoDB забезпечило необхідну гнучкість для зберігання різноманітного контенту, тоді як Express.js дозволив швидко реалізувати RESTful API для управління даними та інтеграції з зовнішніми сервісами.

Особливу цінність представляє реалізована система аналітики, яка не обмежується базовими метриками відвідувань, а надає глибокі інсайти про взаємодію користувачів з контентом. Це дозволяє оптимізувати контент-стратегію на основі реальних даних та підвищувати ефективність блогу як інструменту цифрового маркетингу.

Розроблена платформа створює міцну основу для майбутнього розвитку та масштабування. Модульна архітектура системи дозволяє легко додавати нові функціональні можливості, такі як система коментарів, інтеграція з соціальними мережами або розширені аналітичні інструменти. Використання сучасних стандартів веб-розробки забезпечує довгострокову підтримку та сумісність з майбутніми технологічними оновленнями.

Таким чином, успішна реалізація проєкту веб-сайту блогу з інтеграцією SEO оптимізації та аналітики демонструє важливість комплексного підходу до створення сучасних веб-додатків. Поєднання технічної досконалості з глибоким розумінням потреб користувачів та вимог цифрового маркетингу дозволило створити платформу, яка не лише відповідає поточним стандартам галузі, але й закладає основу для інноваційного розвитку у сфері контент-маркетингу та веб-технологій.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. HTML. Веб-сайт. URL: https://www.w3schools.com/html/
2. CSS. Веб-сайт. URL: https://www.w3schools.com/css/
3. JavaScript Guides. Mozilla Developer Network. Веб-сайт. URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide>
4. React Documentation. React. Веб-сайт. URL: <https://reactjs.org/docs/getting-started.html>
5. Node.js Documentation. Node.js. Веб-сайт. URL: <https://nodejs.org/en/docs/>
6. Express Documentation. Express.js. Веб-сайт. URL: <https://expressjs.com/en/starter/installing.html>
7. MongoDB Manual. MongoDB. Веб-сайт. URL: <https://docs.mongodb.com/manual/>
8. PostHog Documentation. Веб-сайт. URL: https://posthog.com/docs
9. Tailwind CSS Documentation. Веб-сайт. URL: <https://tailwindcss.com/docs/>
10. SEO Starter Guide. Google Search Central. Веб-сайт. URL: <https://developers.google.com/search/docs/beginner/seo-starter-guide>

# ДОДАТКИ

**Лістинг App.tsx**

import { Routes, Route, Navigate } from "react-router-dom";

import Navbar from "./components/navbar.component";

import UserAuthForm from "./pages/userAuthForm.page";

import { createContext, useEffect, useState } from "react";

import { lookInSession } from "./common/session";

import Editor from "./pages/editor.pages";

import HomePage from "./pages/home.page";

import SearchPage from "./pages/search.page";

import PageNotFound from "./pages/404.page";

import ProfilePage from "./pages/profile.page";

import BlogPage from "./pages/blog.page";

import SideNav from "./components/sidenavbar.component";

import ChangePassword from "./pages/change-password.page";

import EditProfile from "./pages/edit-profile.page";

import Notifications from "./pages/notifications.page";

import ManageBlogs from "./pages/manage-blogs.page";

export const UserContext = createContext({});

export const ThemeContext = createContext({});

const darkThemePreference = () =>

window.matchMedia("(prefers-color-scheme: dark)").matches;

const App = () => {

const [userAuth, setUserAuth] = useState({});

const [theme, setTheme] = useState(() =>

darkThemePreference() ? "dark" : "light"

);

useEffect(() => {

let userInSession = lookInSession("user");

let themeInSession = lookInSession("theme");

userInSession

? setUserAuth(JSON.parse(userInSession))

: setUserAuth({ access\_token: null });

if (themeInSession) {

setTheme(() => {

document.body.setAttribute("data-theme", themeInSession);

return themeInSession;

});

} else {

document.body.setAttribute("data-theme", theme);

}

}, []);

return (

<ThemeContext.Provider value={{ theme, setTheme }}>

<UserContext.Provider value={{ userAuth, setUserAuth }}>

<Routes>

<Route path="/editor" element={<Editor />} />

<Route path="/editor/:blog\_id" element={<Editor />} />

<Route path="/" element={<Navbar />}>

<Route index element={<HomePage />} />

<Route path="dashboard" element={<SideNav />}>

<Route path="blogs" element={<ManageBlogs />} />

<Route path="notifications" element={<Notifications />} />

</Route>

<Route path="settings" element={<SideNav />}>

<Route path="edit-profile" element={<EditProfile />} />

<Route path="change-password" element={<ChangePassword />} />

</Route>

<Route path="signin" element={<UserAuthForm type="sign-in" />} />

<Route path="signup" element={<UserAuthForm type="sign-up" />} />

<Route path="search/:query" element={<SearchPage />} />

<Route path="user/:id" element={<ProfilePage />} />

<Route path="blog/:blog\_id" element={<BlogPage />} />

<Route path="\*" element={<PageNotFound />} />

</Route>

</Routes>

</UserContext.Provider>

</ThemeContext.Provider>

);

};

export default App;

**Лістинг main.tsx**

import React from "react";

import ReactDOM from "react-dom/client";

import App from "./App.tsx";

import "./index.css";

import { BrowserRouter } from "react-router-dom";

const rootElement = document.getElementById("root");

if (!rootElement) throw new Error("Root element not found");

ReactDOM.createRoot(rootElement).render(

// <React.StrictMode>

<BrowserRouter>

<App />

</BrowserRouter>

// </React.StrictMode>,

);

**Лістинг blog-editor.jsx**

import { Link, useNavigate, useParams } from "react-router-dom";

import lightLogo from "../imgs/logo-light.png";

import darkLogo from "../imgs/logo-dark.png";

import AnimationWrapper from "../common/page-animation";

import lightBanner from "../imgs/blog banner light.png";

import darkBanner from "../imgs/blog banner dark.png";

import { useContext, useEffect } from "react";

import { Toaster, toast } from "react-hot-toast";

import { EditorContext } from "../pages/editor.pages";

import EditorJS from "@editorjs/editorjs";

import { tools } from "./tools.component";

import axios from "axios";

import { ThemeContext, UserContext } from "../App";

const BlogEditor = () => {

let { blog, blog: { title, banner, content, tags, des }, setBlog, textEditor, setTextEditor, setEditorState } = useContext(EditorContext)

let { userAuth: { access\_token } } = useContext(UserContext)

let { theme } = useContext(ThemeContext);

let { blog\_id } = useParams();

let navigate = useNavigate();

useEffect(() => {

if(!textEditor.isReady){

setTextEditor(new EditorJS({

holderId: "textEditor",

data: Array.isArray(content) ? content[0] : content,

tools: tools,

placeholder: "Let's write an awesome story"

}))

}

}, [])

const handleBannerUpload = (e) => {

let img = e.target.files[0];

if(img){

let loadingToast = toast.loading("Uploading...")

const formData = new FormData();

formData.append('image', img);

axios.post(import.meta.env.VITE\_SERVER\_DOMAIN + "/upload-image", formData, {

headers: {

'Authorization': `Bearer ${access\_token}`,

'Content-Type': 'multipart/form-data'

}

})

.then(response => {

toast.dismiss(loadingToast);

toast.success("Uploaded 👍");

setBlog({ ...blog, banner: response.data.imageUrl })

})

.catch(err => {

toast.dismiss(loadingToast);

toast.error(err.response?.data?.error || "Upload failed");

});

}

}

const handleTitleKeyDown = (e) => {

if(e.keyCode == 13) { // enter key

e.preventDefault();

}

}

const handleTitleChange = (e) => {

let input = e.target;

input.style.height = 'auto';

input.style.height = input.scrollHeight + "px";

setBlog({ ...blog, title: input.value })

}

const handleError = (e) => {

let img = e.target;

img.src = theme == "light" ? lightBanner : darkBanner;

}

const handlePublishEvent = () => {

if(!banner.length){

return toast.error("Upload a blog banner to publish it")

}

if(!title.length){

return toast.error("Write blog title to publish it")

}

if(textEditor.isReady){

textEditor.save().then(data => {

if(data.blocks.length){

setBlog({ ...blog, content: data });

setEditorState("publish")

} else{

return toast.error("Write something in your blog to publish it")

}

})

.catch((err) => {

console.log(err);

})

}

}

const handleSaveDraft = (e) => {

if(e.target.className.includes("disable")) {

return;

}

if(!title.length){

return toast.error("Write blog title before saving it as a draft")

}

let loadingToast = toast.loading("Saving Draft....");

e.target.classList.add('disable');

if(textEditor.isReady){

textEditor.save().then(content => {

let blogObj = {

title, banner, des, content, tags, draft: true

}

axios.post(import.meta.env.VITE\_SERVER\_DOMAIN + "/create-blog", { ...blogObj, id: blog\_id }, {

headers: {

'Authorization': `Bearer ${access\_token}`

}

})

.then(() => {

e.target.classList.remove('disable');

toast.dismiss(loadingToast);

toast.success("Saved 👍");

setTimeout(() => {

navigate("/dashboard/blogs?tab=draft")

}, 500);

})

.catch(( { response } ) => {

e.target.classList.remove('disable');

toast.dismiss(loadingToast);

return toast.error(response.data.error)

})

})

}

}

return (

<>

<nav className="navbar">

<Link to="/" className="flex-none w-10">

<img src={ theme == "light" ? darkLogo : lightLogo } />

</Link>

<p className="max-md:hidden text-black line-clamp-1 w-full">

{ title.length ? title : "New Blog" }

</p>

<div className="flex gap-4 ml-auto">

<button className="btn-dark py-2"

onClick={handlePublishEvent}

>

Publish

</button>

<button className="btn-light py-2"

onClick={handleSaveDraft}

>

Save Draft

</button>

</div>

</nav>

<Toaster />

<AnimationWrapper>

<section>

<div className="mx-auto max-w-[900px] w-full">

<div className="relative aspect-video hover:opacity-80 bg-white border-4 border-grey">

<label htmlFor="uploadBanner">

<img

src={banner}

className="z-20"

onError={handleError}

/>

<input

id="uploadBanner"

type="file"

accept=".png, .jpg, .jpeg"

hidden

onChange={handleBannerUpload}

/>

</label>

</div>

<textarea

defaultValue={title}

placeholder="Blog Title"

className="text-4xl font-medium w-full h-20 outline-none resize-none mt-10 leading-tight placeholder:opacity-40 bg-white"

onKeyDown={handleTitleKeyDown}

onChange={handleTitleChange}

></textarea>

<hr className="w-full opacity-10 my-5" />

<div id="textEditor" className="font-gelasio"></div>

</div>

</section>

</AnimationWrapper>

</>

)

}

export default BlogEditor;

**Лістинг index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<link rel="icon" type="image/svg+xml" href="/vite.svg" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<script>

!function(t,e){var o,n,p,r;e.\_\_SV||(window.posthog=e,e.\_i=[],e.init=function(i,s,a){function g(t,e){var o=e.split(".");2==o.length&&(t=t[o[0]],e=o[1]),t[e]=function(){t.push([e].concat(Array.prototype.slice.call(arguments,0)))}}(p=t.createElement("script")).type="text/javascript",p.crossOrigin="anonymous",p.async=!0,p.src=s.api\_host+"/static/array.js",(r=t.getElementsByTagName("script")[0]).parentNode.insertBefore(p,r);var u=e;for(void 0!==a?u=e[a]=[]:a="posthog",u.people=u.people||[],u.toString=function(t){var e="posthog";return"posthog"!==a&&(e+="."+a),t||(e+=" (stub)"),e},u.people.toString=function(){return u.toString(1)+".people (stub)"},o="capture identify alias people.set people.set\_once set\_config register register\_once unregister opt\_out\_capturing has\_opted\_out\_capturing opt\_in\_capturing reset isFeatureEnabled onFeatureFlags getFeatureFlag getFeatureFlagPayload reloadFeatureFlags group updateEarlyAccessFeatureEnrollment getEarlyAccessFeatures getActiveMatchingSurveys getSurveys getNextSurveyStep".split(" "),n=0;n<o.length;n++)g(u,o[n]);e.\_i.push([i,s,a])},e.\_\_SV=1)}(document,window.posthog||[]);

posthog.init('phc\_DiUrWqwrrhqixtnk0GGw75vuMvrz2PCnuDnLUbQV9ED',{api\_host:'https://eu.i.posthog.com'})

</script>

<title>Vite + React</title>

</head>

<body>

<div id="root"></div>

<script type="module" src="/src/main.tsx"></script>

</body>

</html>

**Лістинг server.js**

const express = require("express");

const cors = require("cors");

const mongoose = require("mongoose");

const bcrypt = require("bcrypt");

const jwt = require("jsonwebtoken");

const multer = require("multer");

const { nanoid } = require("nanoid");

require("dotenv").config(); // Assuming .env is used for DB\_LOCATION, SECRET\_ACCESS\_KEY

const server = express();

const PORT = process.env.PORT || 3000;

// Regex for email and password validation

const emailRegex = /^\w+([\.-]?\w+)\*@\w+([\.-]?\w+)\*(\.\w{2,3})+$/;

const passwordRegex = /^(?=.\*\d)(?=.\*[a-z])(?=.\*[A-Z]).{6,20}$/;

// Middleware setup

server.use(express.json());

server.use(cors());

// Multer storage for file uploads

const storage = multer.diskStorage({

destination: (req, file, cb) => cb(null, "uploads/"),

filename: (req, file, cb) => {

const uniqueSuffix = Date.now() + "-" + Math.round(Math.random() \* 1e9);

cb(null, file.fieldname + "-" + uniqueSuffix + "." + file.originalname.split(".").pop());

},

});

const upload = multer({ storage: storage });

server.use("/uploads", express.static("uploads"));

// Database connection

mongoose.connect(process.env.DB\_LOCATION, { autoIndex: true })

.then(() => console.log("MongoDB connected"))

.catch((err) => console.error("MongoDB connection error:", err));

// JWT Verification Middleware

const verifyJWT = (req, res, next) => {

const token = req.headers["authorization"]?.split(" ")[1];

if (!token) return res.status(401).json({ error: "No access token" });

jwt.verify(token, process.env.SECRET\_ACCESS\_KEY, (err, user) => {

if (err) return res.status(403).json({ error: "Access token invalid" });

req.user = user.id;

next();

});

};

// Data formatting for sending user info

const formatDatatoSend = (user) => {

const access\_token = jwt.sign({ id: user.\_id }, process.env.SECRET\_ACCESS\_KEY);

return { access\_token, profile\_img: user.personal\_info.profile\_img, username: user.personal\_info.username, fullname: user.personal\_info.fullname };

};

// Generate unique username

const generateUsername = async (email) => {

let username = email.split("@")[0];

let isUsernameNotUnique = await User.exists({ "personal\_info.username": username });

return isUsernameNotUnique ? username + nanoid().substring(0, 5) : username;

};

// --- Authentication Routes ---

// Signup

server.post("/signup", async (req, res) => {

const { fullname, email, password } = req.body;

if (fullname.length < 3 || !email.length || !emailRegex.test(email) || !passwordRegex.test(password)) {

return res.status(403).json({ error: "Validation failed" });

}

try {

const hashed\_password = await bcrypt.hash(password, 10);

const username = await generateUsername(email);

const user = new User({ personal\_info: { fullname, email, password: hashed\_password, username } });

await user.save();

res.status(200).json(formatDatatoSend(user));

} catch (err) {

res.status(500).json({ error: err.code === 11000 ? "Email exists" : err.message });

}

});

// Signin

server.post("/signin", async (req, res) => {

const { email, password } = req.body;

try {

const user = await User.findOne({ "personal\_info.email": email });

if (!user) return res.status(403).json({ error: "Email not found" });

const result = await bcrypt.compare(password, user.personal\_info.password);

if (!result) return res.status(403).json({ error: "Incorrect password" });

res.status(200).json(formatDatatoSend(user));

} catch (err) {

res.status(500).json({ error: err.message });

}

});

// Change Password

server.post("/change-password", verifyJWT, async (req, res) => {

const { currentPassword, newPassword } = req.body;

if (!passwordRegex.test(currentPassword) || !passwordRegex.test(newPassword)) {

return res.status(403).json({ error: "Password validation failed" });

}

try {

const user = await User.findOne({ \_id: req.user });

const isMatch = await bcrypt.compare(currentPassword, user.personal\_info.password);

if (!isMatch) return res.status(403).json({ error: "Incorrect current password" });

const hashed\_password = await bcrypt.hash(newPassword, 10);

await User.findOneAndUpdate({ \_id: req.user }, { "personal\_info.password": hashed\_password });

res.status(200).json({ status: "password changed" });

} catch (err) {

res.status(500).json({ error: err.message });

}

});

// --- Blog & User Data Routes ---

// Latest Blogs

server.post("/latest-blogs", (req, res) => {

const { page } = req.body;

const maxLimit = 5;

Blog.find({ draft: false }).populate("author", "personal\_info.profile\_img personal\_info.username personal\_info.fullname -\_id").sort({ publishedAt: -1 }).select("blog\_id title des banner activity tags publishedAt -\_id").skip((page - 1) \* maxLimit).limit(maxLimit)

.then((blogs) => res.status(200).json({ blogs }))

.catch((err) => res.status(500).json({ error: err.message }));

});

// All Latest Blogs Count

server.post("/all-latest-blogs-count", (req, res) => {

Blog.countDocuments({ draft: false })

.then((count) => res.status(200).json({ totalDocs: count }))

.catch((err) => res.status(500).json({ error: err.message }));

});

// Trending Blogs

server.get("/trending-blogs", (req, res) => {

Blog.find({ draft: false }).populate("author", "personal\_info.profile\_img personal\_info.username personal\_info.fullname -\_id").sort({ "activity.total\_read": -1, "activity.total\_likes": -1, publishedAt: -1 }).select("blog\_id title publishedAt -\_id").limit(5)

.then((blogs) => res.status(200).json({ blogs }))

.catch((err) => res.status(500).json({ error: err.message }));

});

// Search Blogs

server.post("/search-blogs", (req, res) => {

const { tag, query, author, page, limit, eliminate\_blog } = req.body;

let findQuery;

if (tag) findQuery = { tags: tag, draft: false, blog\_id: { $ne: eliminate\_blog } };

else if (query) findQuery = { draft: false, title: new RegExp(query, "i") };

else if (author) findQuery = { author, draft: false };

const maxLimit = limit || 2;

Blog.find(findQuery).populate("author", "personal\_info.profile\_img personal\_info.username personal\_info.fullname -\_id").sort({ publishedAt: -1 }).select("blog\_id title des banner activity tags publishedAt -\_id").skip((page - 1) \* maxLimit).limit(maxLimit)

.then((blogs) => res.status(200).json({ blogs }))

.catch((err) => res.status(500).json({ error: err.message }));

});

// Search Blogs Count

server.post("/search-blogs-count", (req, res) => {

const { tag, author, query } = req.body;

let findQuery;

if (tag) findQuery = { tags: tag, draft: false };

else if (query) findQuery = { draft: false, title: new RegExp(query, "i") };

else if (author) findQuery = { author, draft: false };

Blog.countDocuments(findQuery)

.then((count) => res.status(200).json({ totalDocs: count }))

.catch((err) => res.status(500).json({ error: err.message }));

});

// Search Users

server.post("/search-users", (req, res) => {

const { query } = req.body;

User.find({ "personal\_info.username": new RegExp(query, "i") }).limit(50).select("personal\_info.fullname personal\_info.username personal\_info.profile\_img -\_id")

.then((users) => res.status(200).json({ users }))

.catch((err) => res.status(500).json({ error: err.message }));

});

// Get User Profile

server.post("/get-profile", (req, res) => {

const { username } = req.body;

User.findOne({ "personal\_info.username": username }).select("-personal\_info.password -updatedAt -blogs")

.then((user) => res.status(200).json(user))

.catch((err) => res.status(500).json({ error: err.message }));

});

// Update Profile Image

server.post("/update-profile-img", verifyJWT, (req, res) => {

const { url } = req.body;

User.findOneAndUpdate({ \_id: req.user }, { "personal\_info.profile\_img": url })

.then(() => res.status(200).json({ profile\_img: url }))

.catch((err) => res.status(500).json({ error: err.message }));

});

// Update Profile

server.post("/update-profile", verifyJWT, async (req, res) => {

const { username, bio, social\_links } = req.body;

const bioLimit = 150;

if (username.length < 3 || bio.length > bioLimit) return res.status(403).json({ error: "Validation failed" });

try {

for (const key of Object.keys(social\_links)) {

if (social\_links[key].length && key !== "website" && !new URL(social\_links[key]).hostname.includes(`${key}.com`)) {

return res.status(403).json({ error: `${key} link is invalid` });

}

}

await User.findOneAndUpdate({ \_id: req.user }, { "personal\_info.username": username, "personal\_info.bio": bio, social\_links }, { runValidators: true });

res.status(200).json({ username });

} catch (err) {

res.status(500).json({ error: err.code === 11000 ? "username taken" : err.message });

}

});

// Create/Update Blog

server.post("/create-blog", verifyJWT, async (req, res) => {

const authorId = req.user;

let { title, des, banner, tags, content, draft, id } = req.body;

if (!title.length || (!draft && (!des.length || des.length > 200 || !banner.length || !content.blocks.length || !tags.length || tags.length > 10))) {

return res.status(403).json({ error: "Validation failed" });

}

tags = tags.map((tag) => tag.toLowerCase());

let blog\_id = id || title.replace(/[^a-zA-Z0-9]/g, " ").replace(/\s+/g, "-").trim() + nanoid();

try {

if (id) {

await Blog.findOneAndUpdate({ blog\_id }, { title, des, banner, content, tags, draft: draft ?? false });

} else {

const blog = new Blog({ title, des, banner, content, tags, author: authorId, blog\_id, draft: Boolean(draft) });

await blog.save();

await User.findOneAndUpdate({ \_id: authorId }, { $inc: { "account\_info.total\_posts": draft ? 0 : 1 }, $push: { blogs: blog.\_id } });

}

res.status(200).json({ id: blog\_id });

} catch (err) {

res.status(500).json({ error: err.message });

}

});

// Get Blog by ID

server.post("/get-blog", (req, res) => {

const { blog\_id, draft, mode } = req.body;

const incrementVal = mode !== "edit" ? 1 : 0;

Blog.findOneAndUpdate({ blog\_id }, { $inc: { "activity.total\_reads": incrementVal } })

.populate("author", "personal\_info.fullname personal\_info.username personal\_info.profile\_img")

.select("title des content banner activity publishedAt blog\_id tags")

.then(async (blog) => {

if (blog.draft && !draft) return res.status(500).json({ error: "Cannot access draft blogs" });

await User.findOneAndUpdate({ "personal\_info.username": blog.author.personal\_info.username }, { $inc: { "account\_info.total\_reads": incrementVal } });

res.status(200).json({ blog });

})

.catch((err) => res.status(500).json({ error: err.message }));

});

// Like Blog

server.post("/like-blog", verifyJWT, async (req, res) => {

const user\_id = req.user;

const { \_id, islikedByUser } = req.body;

const incrementVal = !islikedByUser ? 1 : -1;

try {

const blog = await Blog.findOneAndUpdate({ \_id }, { $inc: { "activity.total\_likes": incrementVal } });

if (!islikedByUser) {

const like = new Notification({ type: "like", blog: \_id, notification\_for: blog.author, user: user\_id });

await like.save();

} else {

await Notification.findOneAndDelete({ user: user\_id, blog: \_id, type: "like" });

}

res.status(200).json({ liked\_by\_user: !islikedByUser });

} catch (err) {

res.status(500).json({ error: err.message });

}

});

// Is Liked by User

server.post("/isliked-by-user", verifyJWT, (req, res) => {

const { \_id } = req.body;

Notification.exists({ user: req.user, type: "like", blog: \_id })

.then((result) => res.status(200).json({ result }))

.catch((err) => res.status(500).json({ error: err.message }));

});

// Add Comment

server.post("/add-comment", verifyJWT, async (req, res) => {

const user\_id = req.user;

const { \_id, comment, blog\_author, replying\_to, notification\_id } = req.body;

if (!comment.length) return res.status(403).json({ error: "Write something" });

let commentObj = { blog\_id: \_id, blog\_author, comment, commented\_by: user\_id, isReply: Boolean(replying\_to) };

if (replying\_to) commentObj.parent = replying\_to;

try {

const commentFile = await new Comment(commentObj).save();

await Blog.findOneAndUpdate({ \_id }, { $push: { comments: commentFile.\_id }, $inc: { "activity.total\_comments": 1, "activity.total\_parent\_comments": replying\_to ? 0 : 1 } });

let notificationObj = { type: replying\_to ? "reply" : "comment", blog: \_id, notification\_for: blog\_author, user: user\_id, comment: commentFile.\_id };

if (replying\_to) {

const replyingToCommentDoc = await Comment.findOneAndUpdate({ \_id: replying\_to }, { $push: { children: commentFile.\_id } });

notificationObj.replied\_on\_comment = replying\_to;

notificationObj.notification\_for = replyingToCommentDoc.commented\_by;

if (notification\_id) await Notification.findOneAndUpdate({ \_id: notification\_id }, { reply: commentFile.\_id });

}

await new Notification(notificationObj).save();

res.status(200).json({ comment: commentFile.comment, commentedAt: commentFile.commentedAt, \_id: commentFile.\_id, user\_id, children: commentFile.children });

} catch (err) {

res.status(500).json({ error: err.message });

}

});

// Get Blog Comments

server.post("/get-blog-comments", (req, res) => {

const { blog\_id, skip } = req.body;

const maxLimit = 5;

Comment.find({ blog\_id, isReply: false }).populate("commented\_by", "personal\_info.username personal\_info.fullname personal\_info.profile\_img").skip(skip).limit(maxLimit).sort({ commentedAt: -1 })

.then((comments) => res.status(200).json(comments))

.catch((err) => res.status(500).json({ error: err.message }));

});

// Get Replies

server.post("/get-replies", (req, res) => {

const { \_id, skip } = req.body;

const maxLimit = 5;

Comment.findOne({ \_id }).populate({ path: "children", options: { limit: maxLimit, skip: skip, sort: { commentedAt: -1 } }, populate: { path: "commented\_by", select: "personal\_info.profile\_img personal\_info.fullname personal\_info.username" }, select: "-blog\_id -updatedAt" }).select("children")

.then((doc) => res.status(200).json({ replies: doc.children }))

.catch((err) => res.status(500).json({ error: err.message }));

});

// Helper: Delete Comments recursively

const deleteComments = async (\_id) => {

try {

const comment = await Comment.findOneAndDelete({ \_id });

if (comment.parent) await Comment.findOneAndUpdate({ \_id: comment.parent }, { $pull: { children: \_id } });

await Notification.findOneAndDelete({ comment: \_id });

await Notification.findOneAndUpdate({ reply: \_id }, { $unset: { reply: 1 } });

await Blog.findOneAndUpdate({ \_id: comment.blog\_id }, { $pull: { comments: \_id }, $inc: { "activity.total\_comments": -1, "activity.total\_parent\_comments": comment.parent ? 0 : -1 } });

if (comment.children.length) for (const replyId of comment.children) await deleteComments(replyId);

} catch (err) {

console.error("Error deleting comment:", err.message);

}

};

// Delete Comment

server.post("/delete-comment", verifyJWT, async (req, res) => {

const user\_id = req.user;

const { \_id } = req.body;

try {

const comment = await Comment.findOne({ \_id });

if (user\_id === comment.commented\_by.toString() || user\_id === comment.blog\_author.toString()) {

await deleteComments(\_id);

res.status(200).json({ status: "done" });

} else {

res.status(403).json({ error: "Unauthorized to delete" });

}

} catch (err) {

res.status(500).json({ error: err.message });

}

});

// New Notification Check

server.get("/new-notification", verifyJWT, (req, res) => {

Notification.exists({ notification\_for: req.user, seen: false, user: { $ne: req.user } })

.then((result) => res.status(200).json({ new\_notification\_available: Boolean(result) }))

.catch((err) => res.status(500).json({ error: err.message }));

});

// Get Notifications

server.post("/notifications", verifyJWT, (req, res) => {

const { page, filter, deletedDocCount } = req.body;

const maxLimit = 10;

let findQuery = { notification\_for: req.user, user: { $ne: req.user } };

let skipDocs = (page - 1) \* maxLimit;

if (filter !== "all") findQuery.type = filter;

if (deletedDocCount) skipDocs -= deletedDocCount;

Notification.find(findQuery).skip(skipDocs).limit(maxLimit)

.populate("blog", "title blog\_id")

.populate("user", "personal\_info.fullname personal\_info.username personal\_info.profile\_img")

.populate("comment", "comment")

.populate("replied\_on\_comment", "comment")

.populate("reply", "comment")

.sort({ createdAt: -1 }).select("createdAt type seen reply")

.then(async (notifications) => {

await Notification.updateMany({ \_id: { $in: notifications.map(n => n.\_id) } }, { seen: true });

res.status(200).json({ notifications });

})

.catch((err) => res.status(500).json({ error: err.message }));

});

// All Notifications Count

server.post("/all-notifications-count", verifyJWT, (req, res) => {

const { filter } = req.body;

let findQuery = { notification\_for: req.user, user: { $ne: req.user } };

if (filter !== "all") findQuery.type = filter;

Notification.countDocuments(findQuery)

.then((count) => res.status(200).json({ totalDocs: count }))

.catch((err) => res.status(500).json({ error: err.message }));

});

// User Written Blogs

server.post("/user-written-blogs", verifyJWT, (req, res) => {

const { page, draft, query, deletedDocCount } = req.body;

const maxLimit = 5;

let skipDocs = (page - 1) \* maxLimit;

if (deletedDocCount) skipDocs -= deletedDocCount;

Blog.find({ author: req.user, draft, title: new RegExp(query, "i") }).skip(skipDocs).limit(maxLimit).sort({ publishedAt: -1 }).select("title banner publishedAt blog\_id activity des draft -\_id")

.then((blogs) => res.status(200).json({ blogs }))

.catch((err) => res.status(500).json({ error: err.message }));

});

// User Written Blogs Count

server.post("/user-written-blogs-count", verifyJWT, (req, res) => {

const { draft, query } = req.body;

Blog.countDocuments({ author: req.user, draft, title: new RegExp(query, "i") })

.then((count) => res.status(200).json({ totalDocs: count }))

.catch((err) => res.status(500).json({ error: err.message }));

});

// Delete Blog

server.post("/delete-blog", verifyJWT, async (req, res) => {

const user\_id = req.user;

const { blog\_id } = req.body;

try {

const blog = await Blog.findOneAndDelete({ blog\_id, author: user\_id }); // Added author check for security

if (!blog) return res.status(404).json({ error: "Blog not found or unauthorized" });

await Promise.all([

Notification.deleteMany({ blog: blog.\_id }),

Comment.deleteMany({ blog\_id: blog.\_id }),

User.findOneAndUpdate({ \_id: user\_id }, { $pull: { blogs: blog.\_id }, $inc: { "account\_info.total\_posts": -1 } }),

]);

res.status(200).json({ status: "done" });

} catch (err) { res.status(500).json({ error: err.message }); }

});

server.post("/upload-image", verifyJWT, upload.single("image"), (req, res) => {

if (!req.file) return res.status(400).json({ error: "No image uploaded" });

const imageUrl = `${req.protocol}://${req.get("host")}/uploads/${req.file.filename}`;

res.status(200).json({ imageUrl });

});

server.listen(PORT, () => console.log(`Listening on port -> ${PORT}`));