МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

ННІ	Економіки та бізнес-освіти
Кафедра	Економіки та цифрового бізнесу
Спеціальність	122 Комп'ютерні науки
Форма навчання	Денна

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

Гончарова Сергія Олександровича
(прізвище, ім'я, по батькові)

на тему Проєктування та реалізація веб-платформи управління блог-контентом із системою категоризації, тегування та персоналізації
(повна назва теми)

за матеріалами _____
(повна назва бази дослідження)

науковий керівник _____ Шокотько Л.М.

Робота допущена до захисту в ЕК
Протокол засідання кафедри
від 12.06.2026 р. № 15

Завідувач кафедри _____
(підпис)

к.е.н., доцент Радько В.М.
наук. ступень, вчене звання прізвище, ініціали

Кривий Ріг − 2026

ЗАТВЕРДЖЕНО
Наказ Міністерства освіти і науки, молоді та спорту України
29 березня 2012 року № 384

Форма № Н-9.01

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕКОНОМІКИ ТА БІЗНЕС-ОСВІТИ
(повне найменування вищого навчального закладу)

Кафедра економіки та цифрового бізнесу
Освітній ступінь бакалавр
Спеціальність 122 Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри _____В.М. Радько

"30" березня 2026 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ ЗДОБУВАЧУ

_____Гончарову_Сергію Олександровичу_____

1. Тема роботи __Проєктування та реалізація вебплатформи управління блог-контентом із системою категоризації, тегування та персоналізації_____

науковий керівник роботи _____Шокотько Людмила Миколаївна,_____
затверджені наказом вищого навчального закладу від «23» березня 2026 р. № 193-ст

2. Строк подання здобувачем роботи 29.05.2026р.

3. Зміст кваліфікаційної бакалаврської роботи, об'єкт, предмет та мета дослідження:

Розділ 1 Теоретичний аналіз інформаційних потоків та особливостей предметної області дослідження

Розділ 2 Проєктування та розробка веб-платформи управління блог-контентом _____

Розділ 3 Реалізація та тестування веб-платформи управління блог-контентом _____

Об'єкт дослідження – процес проєктування та реалізації веб-платформи управління блог-контентом на основі сучасних веб-технологій.

Предмет дослідження − методи та технології створення веб-додатків на стеку MERN

Мета кваліфікаційної бакалаврської роботи – проєктування та реалізація повнофункціональної веб-платформи управління блог-контентом

4. Дата видачі завдання 03.04.2026р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної бакалаврської роботи	Строк виконання етапів роботи	Відмітка керівника про виконання етапів (дата, підпис)
1	Підготовка розділу 1	до 17.04.2026р.	17.04.2026р.
2	Підготовка розділу 2	до 08.05.2026р.	08.05.2026р.
3	Підготовка розділу 3	до 25.05.2026р.	25.05.2026р
4	Реєстрація завершеної кваліфікаційної роботи	до 29.05.2026р.	29.05.2026р.
5	Отримання відгуку від наукового керівника	04.06.2026р.	04.06.2026р.
6	Отримання зовнішньої рецензії	05.06.2026р.	05.06.2026р.
7	Попередній захист кваліфікаційної роботи на кафедрі	08.06.2026р.	08.06.2026р.
8	Перевірка кваліфікаційної роботи на плагіат	12.06.2026р.	12.06.2026р.
9	Допуск кафедрою кваліфікаційної роботи до захисту	12.06.2026р.	12.06.2026р.
10	Підготовка студента до захисту в ЕК	до 19.06.2026р.	19.06.2026р.

Завдання підготував науковий керівник _____ Шокотько Л. М.

Завдання одержав здобувач _____ Гончаров С.О.

(підпис) (прізвище та ініціали)

РЕФЕРАТ

Кваліфікаційна робота бакалавра: 68 стор., 5 рисунків, 11 таблиць, 25 використаних джерел, 8 додатків.

Метою роботи є проєктування та реалізація повнофункціональної веб-платформи управління блог-контентом із системою категоризації, тегування, повнотекстового пошуку, коментування з лайками та адміністративною панеллю.

Об’єкт дослідження − процес проєктування та реалізації веб-платформи управління блог-контентом на основі сучасних веб-технологій.

Предмет дослідження − методи та технології створення веб-додатків на стеку MERN (MongoDB, Express.js, React, Node.js) з інтеграцією Firebase, Redux Toolkit та JWT-аутентифікації.

У ході виконання роботи було проаналізовано сучасні технології веб-розробки, визначено функціональні та нефункціональні вимоги до системи, спроєктовано архітектуру веб-додатку та структуру бази даних, реалізовано серверну та клієнтську частини застосунку, систему аутентифікації (JWT + Google OAuth), хмарне зберігання зображень, адміністративну панель та проведено тестування функціоналу.

Експлуатаційне призначення програмного комплексу полягає у наданні користувачам зручного інструменту для публікації та управління блог-контентом, а також можливості адміністраторам ефективно модерувати контент і користувачів.

Ключові слова: адміністративна панель, веб-платформа, управління блог-контентом, JWT-аутентифікація, MERN-стек, Express.js, Firebase, Google OAuth, MongoDB, Node.js, React, Redux Toolkit, REST API, Single Page Application.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API (Application Programming Interface) − інтерфейс програмування застосунків.

BSON (Binary JSON) − бінарний JSON.

CMS (Content Management System) − система управління контентом.

CORS (Cross-Origin Resource Sharing) − обмін ресурсами між джерелами.

CRUD (Create, Read, Update, Delete) − створення, читання, оновлення та видалення.

CSS (Cascading Style Sheets) − каскадні таблиці стилів.

DOM (Document Object Model) − об’єктна модель документа.

ECMAScript (ES) − міжнародний стандарт мови JavaScript.

HTML (HyperText Markup Language) − мова розмітки гіпертексту.

HTTPS (HyperText Transfer Protocol Secure) − захищений протокол передачі гіпертексту.

JSX (JavaScript XML) − JavaScript XML.

JWT (JSON Web Token) − JSON Web Token.

MERN (MongoDB, Express.js, React, Node.js) − стек технологій.

MVC (Model-View-Controller) − модель-представлення-контролер.

NoSQL (Not Only SQL) − не лише SQL.

OAuth (Open Authorization) − відкрита авторизація.

ODM (Object Document Mapper) − об’єктно-документний мапер.

REST (Representational State Transfer) − передача репрезентативного стану.

SEO (Search Engine Optimization) − оптимізація для пошукових систем.

SPA (Single Page Application) − односторінковий застосунок.

SQL (Structured Query Language) − структурована мова запитів.

UI (User Interface) − інтерфейс користувача.

URL (Uniform Resource Locator) − уніфікований покажчик ресурсу.

YSIWYG (What You See Is What You Get) − що бачиш, те й отримуєш.

ВСТУП

У мережі існують цифрові майданчики, які люди використовують для роботи з даними. Блоги функціонують як засіб, через який користувачі оприлюднюють тексти. Користувачі часто потребують сервіси для ведення записів. З цієї причини розробники створюють системи, що групують матеріали за розділами. У таких системах працює механізм для знаходження слів у текстах. Також у програмах існують різні права доступу для тих, хто керує ресурсом.

Для створення цієї платформи потрібні певні програмні інструменти. В описі фігурує структура MERN. MongoDB зберігає інформацію у формі документів. Express.js функціонує як основа для логіки на сервері. React є набором інструментів для створення зовнішнього вигляду сторінок. Node.js працює як простір, де виконується код на боці сервера. Технологія Single Page Application дозволяє переходити між розділами сайту. При цьому вікно програми не оновлюється повністю. Від цього робота з інтерфейсом стає стабільною.

На клієнтській стороні працює React. В основі глобального управління станом лежить бібліотека Redux Toolkit. Збереження сесії після перезавантаження сторінки забезпечує інструмент redux persist. Для побудови інтерфейсу вибрано Flowbite React. Клієнтська маршрутизація реалізована через React Router DOM. Firebase виконує перевірку особи через Google OAuth. Також сервіс Firebase зберігає зображення профілів у хмарі. Окремо хмарне сховище приймає файли публікацій. На сервері діє RESTful API. Маршрути контролюють дані користувачів. Інші маршрути керують публікаціями. Також існують шляхи для коментарів. Middleware перевіряє JWT-токени для захисту цих маршрутів.

Захист даних є основною характеристикою платформи - bcryptjs перетворює паролі користувачів у хеш-код. JSON Web Tokens підтверджують особу під час доступу. Система передає ці токени у cookie-файлах із

прапорцем httpOnly. У структурі діє розподіл прав за ролями. Адміністратори мають доступ до панелі керування користувачами. Також адміністратори модерують публікації. Окремий доступ надано для керування коментарями. Звичайні користувачі переглядають контент. Додатково користувачі залишають коментарі. Редагування власного профілю є доступним для власників акаунтів.

Актуальність даної роботи полягає у необхідності створення повнофункціональної веб-платформи управління блог-контентом, яка демонструє практичне застосування сучасного стеку MERN, реалізує систему категоризації публікацій, пошук за контентом, коментування, Google-аутентифікацію через Firebase та повноцінну адміністративну панель керування.

Мета кваліфікаційної роботи полягає у проєктуванні та реалізації повнофункціональної веб-платформи управління блог-контентом із використанням стеку MERN (MongoDB, Express.js, React, Node.js), що включає систему аутентифікації з підтримкою Google OAuth через Firebase, категоризацію публікацій, повнотекстовий пошук, систему коментування з лайками, хмарне зберігання зображень та адміністративну панель керування контентом і користувачами.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

- проаналізувати сучасні технології та підходи до розробки веб-платформ для управління блог-контентом;
- дослідити архітектурні патерни побудови клієнт-серверних застосунків та підходи до управління глобальним станом у React-додатках;
- розробити функціональні вимоги та технічні характеристики системи;
- спроектувати архітектуру веб-додатку та структуру бази даних;
- реалізувати серверну частину на Node.js та Express.js з підключенням до MongoDB через Mongoose, налаштуванням CORS та middleware верифікації JWT;

- розробити клієнтську частину з використанням React, Redux Toolkit, React Router DOM, Flowbite-React та Firebase SDK;
- реалізувати систему аутентифікації на основі JWT у httpOnly cookie та Google OAuth через Firebase, з розмежуванням прав між адміністраторами і звичайними користувачами;
- впровадити систему категоризації публікацій, повнотекстовий пошук, коментування з лайками, завантаження зображень через Firebase Storage та адміністративну панель керування;
- протестувати розроблений веб-додаток та проаналізувати результати.

Об’єктом дослідження є процес проєктування та реалізації повнофункціональної веб-платформи управління блог-контентом на основі стеку MERN з інтеграцією хмарних сервісів Firebase.

Предметом дослідження виступають методи та технології створення веб-платформ управління блог-контентом, включаючи архітектурні рішення на базі MERN, механізми JWT-аутентифікації та Google OAuth, управління глобальним станом через Redux Toolkit, систему категоризації публікацій, коментування та адміністрування контенту.

Методи дослідження

У процесі виконання кваліфікаційної роботи використовувалися наступні методи:

- аналіз та синтез теоретичних відомостей з веб-програмування;
- порівняльний аналіз існуючих технологій та фреймворків;
- об’єктно-орієнтоване проектування системи;
- методи реляційного та документо-орієнтованого моделювання даних;
- алгоритмічний підхід до розробки функціоналу;
- методи тестування програмного забезпечення.

Інформаційна база дослідження включає навчальні посібники з веб-програмування, офіційну документацію технологій React, Node.js, Express.js,

MongoDB, наукові статті та публікації у сфері розробки веб-додатків, онлайн-ресурси та навчальні матеріали провідних IT-платформ.

Практичне значення роботи полягає у створенні повністю робочої веб-платформи управління блог-контентом, що розгорнута на Vercel і може слугувати як основа для комерційних блог-сервісів, портфоліо-проєкт для демонстрації навичок повного стеку MERN, або навчальний зразок інтеграції React, Redux Toolkit, Firebase та JWT-аутентифікації. Реалізована архітектура з розмежуванням ролей, системою коментування, категоризацією публікацій та адмін-панеллю може бути адаптована для медіа-порталів, корпоративних блогів та інших систем управління контентом.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ ВЕБ-ДОДАТКІВ ДЛЯ БЛОГІВ

1.1 Аналіз існуючих рішень для управління блог-контентом із категоризацією та персоналізацією

Сучасні веб-додатки стали невід'ємною частиною цифрового простору, забезпечуючи ефективне управління інформаційним контентом, його поширення та взаємодію з користувачами. Вони поєднують у собі складні архітектурні рішення, сучасні технології фронтенду та бекенду, бази даних і хмарні сервіси, що дозволяє створювати масштабовані, зручні та персоналізовані платформи. У контексті управління блог-контентом особливої актуальності набувають системи, які підтримують категоризацію матеріалів, тегування, повнотекстовий пошук, коментування та персоналізацію контенту для різних груп користувачів. Аналіз існуючих рішень дозволяє виявити найкращі практики реалізації таких функціональних можливостей, оцінити їх сильні та слабкі сторони, а також обґрунтувати вибір технологічного стеку для розробки власної веб-платформи.

Існує велика кількість готових платформ та фреймворків для створення блогів, кожна з яких має свої особливості. Аналіз існуючих рішень дозволяє виявити найкращі практики та обґрунтувати вибір власної реалізації.

WordPress. WordPress − найпопулярніша CMS у світі, написана на PHP із MySQL. Підтримує категорії, теги, систему ролей та тисячі плагінів для розширення функціоналу. Переваги: простота використання, величезна екосистема, вбудована SEO-оптимізація. Недоліки для даного проєкту: PHP-стек несумісний з MERN, монолітна архітектура ускладнює розгортання на Vercel, а надлишковий функціонал не відповідає навчальним цілям.

Ghost. Ghost − сучасна блог-платформа на Node.js із вбудованою підтримкою категорій, тегів та підписок. Технологічно ближча до даного проєкту. Переваги: висока швидкість, зручний редактор, монетизація.

Недоліки: Ghost є готовим монолітним продуктом із власною архітектурою, що виключає можливість самостійної реалізації JWT-аутентифікації, Firebase-інтеграції та адмін-панелі.

Medium. Medium − хмарна платформа з персоналізованою стрічкою та системою рекомендацій контенту. Переваги: вбудована аудиторія, зручний редактор. Недоліки: відсутність контролю над даними та кодом, неможливість кастомізації та інтеграції зовнішніх сервісів.

Jekyll та статичні генератори сайтів. Jekyll, Hugo та Gatsby генерують статичні HTML-сторінки з Markdown-файлів. Переваги: максимальна швидкість, безкоштовний хостинг на GitHub Pages. Недоліки: відсутність динамічного контенту унеможливлює реалізацію реєстрації користувачів, системи коментарів із лайками, пошуку та адмін-панелі - ключових функцій даного проєкту.

Власні рішення на базі фреймворків. Розробка власної платформи на React та Node.js дає повний контроль над архітектурою, функціоналом та безпекою. Переваги: гнучкість реалізації, відсутність надлишкового коду, можливість інтеграції Firebase, Redux та будь-яких зовнішніх сервісів, навчальна цінність вивчення повного стеку. Недоліки: вищі початкові витрати часу на розробку кожного компонента з нуля.

Порівняльний аналіз показує, що готові рішення підходять для швидкого запуску без програмування, але обмежені у кастомізації. Власна реалізація на MERN оптимальна для навчальних цілей та демонстрації навичок розробки, оскільки охоплює весь цикл: від проєктування бази даних до розгортання на Vercel.

Для навчальних цілей та демонстрації сучасних технологій розробки власного блог-додатку на базі React та Node.js є оптимальним вибором. Такий підхід дозволяє глибоко вивчити весь стек веб-технологій − від роботи з базою даних до побудови користувацького інтерфейсу та отримати практичний досвід реалізації повнофункціонального продукту.

Основні функціональні вимоги до платформи. На основі аналізу існуючих рішень визначено перелік функцій, реалізованих у даній роботі:

- реєстрація та вхід через форму (email + пароль, bcryptjs) та Google OAuth через Firebase з JWT-аутентифікацією у httpOnly cookie;
- створення, редагування та видалення публікацій із заголовком, контентом (React Quill), категорією та зображенням обкладинки (Firebase Storage);
- категоризація публікацій та фільтрація за категорією на сторінці пошуку;
- повнотекстовий пошук публікацій за заголовком та контентом із сортуванням та пагінацією;
- система коментарів із лайками, редагуванням та видаленням власних коментарів;
- адміністративна панель із вкладками: статистика (користувачі, публікації, коментарі за останній місяць), управління публікаціями, користувачами та коментарями;
- управління профілем: зміна імені, email, пароля та фото через Firebase Storage із відображенням прогресу завантаження;
- адаптивний інтерфейс із підтримкою темної теми через Redux та Tailwind CSS;
- захищені маршрути з розмежуванням доступу між адміністраторами та звичайними користувачами.

1.2 Аналіз сучасних технологій розробки веб-додатків

Сучасна веб-розробка базується на широкому використанні технологій, фреймворків та бібліотек, що дозволяють будувати складні, масштабовані й високопродуктивні системи. Вибір технологічного стеку є стратегічним рішенням: він визначає архітектуру, швидкість розробки та довгострокову підтримуваність проєкту. У даній роботі застосовується стек MERN -

MongoDB, Express.js, React та Node.js - що забезпечує використання єдиної мови програмування на всіх рівнях системи.

JavaScript як універсальна мова веб-розробки. JavaScript залишається найпопулярнішою мовою програмування у веб-розробці, що підтверджується щорічними дослідженнями Stack Overflow Developer Survey. (Stack Overflow Developer Survey 2024) [25] Принципова перевага JavaScript полягає у можливості використовувати одну мову як у браузері, так і на сервері завдяки Node.js. Це дозволяє розробникам працювати з єдиним технологічним стеком, спрощуючи розробку та підтримку проєктів.

Основні переваги JavaScript охоплюють асинхронну модель виконання з підтримкою async/await та Promise, розвинену екосистему npm-пакетів, активну спільноту розробників та широку браузерну підтримку. Стандарт ECMAScript регулярно оновлюється, вводячи нові можливості: стрілкові функції, деструктуризацію, spread-оператори, класи та ES-модулі, які активно застосовуються у даному проєкті.

Node.js як серверне середовище. Node.js − середовище виконання JavaScript на стороні сервера, побудоване на базі рушія V8 від Google Chrome. Головна особливість Node.js - подієво-орієнтована архітектура з неблокуючим вводом-виводом, що дозволяє ефективно обробляти велику кількість одночасних запитів з мінімальними витратами ресурсів. (Морган Дж. Node.js) [3], (Official Express.js Documentation) [8]

Node.js застосовує модель single-threaded event loop, яка відрізняється від традиційної багатопотокової моделі PHP або Java. Завдяки цьому Node.js може обслуговувати тисячі з'єднань на одному сервері без виділення окремого потоку на кожен запит, що робить його ефективним для API-сервісів.

Екосистема Node.js включає пакетний менеджер npm з доступом до понад мільйона модулів, які дозволяють швидко інтегрувати готові рішення для типових завдань: аутентифікації, валідації даних, роботи з базами даних тощо.

Express.js як веб-фреймворк. Express.js є найпоширенішим веб-фреймворком для Node.js, що надає мінімалістичний та гнучкий набір інструментів для побудови веб-додатків та API. Ключовою особливістю Express.js є middleware-архітектура, яка дозволяє модульно організовувати код та повторно використовувати функціональність. (Official Express.js Documentation) [8]

Express.js підтримує маршрутизацію HTTP-запитів, обробку різних HTTP-методів та гнучку систему middleware для логування, аутентифікації, валідації та обробки помилок. Фреймворк дозволяє швидко будувати RESTful API - стандарт взаємодії між клієнтською та серверною частинами сучасних веб-додатків.

MongoDB як NoSQL база даних. MongoDB − документо-орієнтована NoSQL СУБД, що зберігає дані у форматі BSON (бінарне JSON). На відміну від реляційних баз даних, MongoDB не потребує жорсткої схеми, що забезпечує гнучкість при роботі з даними змінної структури.

Переваги MongoDB включають горизонтальне масштабування через шардинг, вбудовану реплікацію, високу продуктивність операцій читання та запису, потужну мову запитів та підтримку індексів. MongoDB особливо підходить для зберігання даних блог-платформ, де публікації мають різну структуру, а системи категоризації та коментування потребують гнучкого зберігання масивів і вкладених документів. (Official MongoDB Documentation) [9], (Mongoose ODM Documentation) [16]

Mongoose є ODM-бібліотекою для MongoDB у Node.js, яка надає інструменти для визначення схем, валідації даних, створення запитів та роботи зі зв'язками між документами. Mongoose дозволяє організувати роботу з MongoDB у структурований спосіб, зберігаючи всю гнучкість NoSQL-підходу.

React як бібліотека для побудови інтерфейсів. React − JavaScript-бібліотека компанії Facebook для створення декларативних компонентних інтерфейсів. React базується на компонентному підході, де UI розбивається на незалежні багаторазово використовувані блоки, кожен з яких управляє

власним станом. (Браун І. Вивчаємо React / Ів Браун, Алекс Бенкс)[2], (Official React Documentation) [6]

Ключовою особливістю React є Virtual DOM − віртуальне представлення DOM-дерева в пам'яті. При зміні стану React порівнює попередній і поточний Virtual DOM та оновлює лише змінені елементи реального DOM, що підвищує продуктивність при роботі зі складними інтерфейсами.

React застосовує декларативний підхід: розробник описує бажаний вигляд інтерфейсу, а React самостійно синхронізує DOM зі станом. JSX дозволяє записувати HTML-подібну розмітку безпосередньо у JavaScript, роблячи код більш читабельним. Хуки (useState, useEffect, useContext тощо) надають функціональним компонентам доступ до стану та lifecycle-методів без використання класів.

Екосистема React охоплює React Router для маршрутизації, Redux або Context API для управління глобальним станом, а також бібліотеки компонентів на кшталт Flowbite-React, що прискорюють розробку інтерфейсу.

Система контролю версій Git. Git - розподілена система контролю версій для відстеження змін у коді, командної роботи та управління різними версіями програмного забезпечення. Платформи GitHub, GitLab та Bitbucket надають хостинг репозиторіїв, інструменти для code review та автоматизацію CI/CD. Використання Git є стандартною практикою у веб-розробці, що забезпечує безпечне внесення змін, можливість відкату та паралельну роботу у гілках.

На основі аналізу вимог до веб-платформи для управління блог-контентом (продуктивність, зручність розробки, масштабованість, підтримка автентифікації, робота з медіафайлами та швидке розгортання) пропонується такий стек:

Управління глобальним станом − Redux Toolkit

Для централізованого та ефективного управління станом застосунку рекомендується використовувати Redux Toolkit. (Банкс А. React і Redux: функціональна веб-розробка) [4]

Планується створення двох основних slice:

- userSlice − зберігатиме дані поточного користувача (currentUser, loading, error) та міститиме actions/thunks для операцій signin, signout, update та delete;
- themeSlice − відповідатиме за поточну тему застосунку та дію toggleTheme.

Для збереження стану між сесіями пропонується підключити redux-persist з використанням localStorage.

2. Хмарні сервіси − Firebase + Cloudinary

Firebase планується використовувати для ключових backend-функцій:

- Firebase Authentication - реалізація Google OAuth авторизації через GoogleAuthProvider + signInWithPopup;
- Firebase Firestore (за потреби) − для зберігання метаданих публікацій, користувачів тощо.

Cloudinary рекомендується як основна платформа для роботи з медіафайлами (зображення профілів, обкладинки публікацій, галереї тощо).

Переваги Cloudinary порівняно/доповнення до Firebase Storage:

- потужні можливості трансформацій зображень «на льоту» (автоматична оптимізація, ресайз, crop, формат WebP/AVIF, lazy loading тощо);
- вбудована підтримка CDN для швидкої доставки медіа по всьому світу.
- автоматичне керування версіями файлів і легке оновлення;
- багатий SDK для JavaScript (cloudinary-react, @cloudinary/react) та зручний widget для завантаження;
- краща продуктивність і якість зображень для блогу.

Завантаження файлів планується реалізовувати через upload / uploadPreset з відстеженням прогресу та отриманням оптимізованих URL. При необхідності можна комбінувати обидва сервіси: Firebase Authentication + Cloudinary для медіа.

3. Бібліотека UI-компонентів - Flowbite-React

Для прискорення розробки інтерфейсу та забезпечення сучасного вигляду рекомендується Flowbite-React (на базі Tailwind CSS).

Бібліотека надає готові компоненти: Button, Modal, Table, TextInput, Navbar, Sidebar, Dropdown, Alert, Spinner, FileInput, Select, Textarea тощо, з вбудованою підтримкою темної та світлої теми.

4. Інструмент збірки - Vite

Для клієнтської частини оптимально використовувати Vite. Переваги:

- миттєвий dev-сервер завдяки нативним ES-модулям;
- швидка production-збірка з tree-shaking та code-splitting;
- зручна робота зі змінними середовища через префікс VITE_.

5. Платформа розгортання - Vercel

Для розгортання серверної частини (serverless-функції Node.js) рекомендується Vercel.

Особливості налаштування:

- конфігураційний файл vercel.json;
- ліниве підключення до MongoDB (флаг isConnected);
- оптимальні параметри пулу з’єднань (maxPoolSize: 1, minPoolSize: 0) для serverless-середовища.

1.3 Огляд архітектурних патернів веб-додатків

Архітектура веб-додатку визначає його структуру, взаємодію компонентів та підходи до організації коду. Правильний вибір архітектурного підходу є критичним чинником для забезпечення масштабованості, підтримуваності та продуктивності системи.

Монолітна архітектура. Традиційно веб-додатки будувалися за монолітним принципом, де весь застосунок - інтерфейс, бізнес-логіка та доступ до даних - розміщений в одній кодовій базі. Перевагою є простота розробки і розгортання для невеликих проєктів, відсутність міжсервісної комунікації та підтримка транзакцій бази даних. Проте при зростанні проєкту будь-яка зміна вимагає перерозгортання всього застосунку, а масштабування можливе лише вертикально.

Клієнт-серверна архітектура. Сучасні веб-додатки переважно використовують клієнт-серверну модель, де функціональність розділена між браузерною клієнтською частиною та серверною. Клієнт відповідає за відображення даних та обробку взаємодії користувача, сервер - за бізнес-логіку, роботу з базою даних та надання API. Взаємодія відбувається через HTTP у форматі JSON, що дозволяє розробляти клієнт і сервер незалежно.

REST API архітектура. REST - архітектурний стиль для проєктування мережевих сервісів на основі стандартних HTTP-методів: GET для отримання даних, POST для створення, PUT або PATCH для оновлення та DELETE для видалення ресурсів. Ключові принципи REST включають безстатевість запитів, єдиноманітність інтерфейсу та можливість кешування відповідей. RESTful API є стандартом де-факто для веб-сервісів завдяки простоті, масштабованості та незалежності клієнта від сервера. (Тілков С. REST і RESTful: розуміння основ) [5], (REST API Design Best Practices) [23]

Single Page Application архітектура. SPA - підхід, при якому весь HTML, JavaScript та CSS завантажуються при першому відвідуванні, а подальша навігація відбувається без перезавантаження сторінки. JavaScript динамічно оновлює контент, надсилаючи асинхронні запити до API. Переваги SPA: висока швидкість взаємодії після першого завантаження, плавні переходи між сторінками, менше навантаження на сервер. Недоліки: більший початковий розмір бандлу та складнощі з SEO-індексацією. (Мюллер Дж. Архітектура Single Page Applications) [18]

MVC патерн. Model-View-Controller розділяє застосунок на три компоненти: Model (дані та бізнес-логіка), View (відображення) та Controller (обробка вводу та координація). У серверних веб-додатках Model представляє роботу з базою даних, View - генерацію відповідей, Controller - обробку HTTP-запитів. На клієнтській стороні у React MVC трансформується у компонентну архітектуру, де стан виступає моделлю. (Хармс Д. Паттерни проектування веб-додатків) [13]

Мікросервісна архітектура. Мікросервісний підхід передбачає розбиття застосунку на множину незалежних сервісів, кожен з яких виконує конкретну функцію і може розгортатися окремо. Переваги: незалежне масштабування сервісів, технологічна гнучкість, покращена відмовостійкість. Для проєктів середнього розміру, як дана блог-платформа, монолітна клієнт-серверна архітектура з RESTful API є оптимальним рішенням.

Архітектурні рішення, які ми розглядаємо / плануємо реалізувати у проєкті.

Проєкт пропонується побудувати на клієнт-серверній архітектурі з чітким розділенням відповідальності між фронтендом (React SPA) та бекендом (Express.js RESTful API).

Серверна частина

Для серверної частини пропонується застосувати MVC-патерн:

- моделі (user.model.js, post.model.js, comment.model.js) — визначатимуть Mongoose-схеми;
- контролери (auth.controller.js, post.controller.js, user.controller.js, comment.controller.js) - міститимуть основну бізнес-логіку;
- маршрути (auth.route.js, post.route.js, user.route.js, comment.route.js) - забезпечуватимуть зв’язок URL з контролерами та відповідними middleware.

Додатково планується використовувати утиліти error.js (для централізованої обробки помилок) та verifyUser.js / verifyToken (для верифікації JWT).

Очікується реалізація близько п'ятнадйяти ендпоінтів, згрупованих за ресурсами відповідно до REST-конвенцій. Зокрема:

- GET /api/post/getposts повинен підтримувати фільтрацію за userId, category, slug, postId, searchTerm, а також сортування та пагінацію;
- критичні операції (створення та оновлення постів) - POST /api/post/create, PUT /api/post/updatepost/:postId/:userId - будуть захищені middleware авторизації.

Кожну групу маршрутів планується підключати через middleware connectMiddleware, який забезпечуватиме ліниве підключення до MongoDB. Це рішення особливо важливе для оптимізації роботи в serverless-середовищі (Vercel).

Клієнтська частина

На фронтенді пропонується використовувати React Router DOM v7 для SPA-маршрутизації.

Для захисту маршрутів планується реалізувати компоненти:

- PrivateRoute - перевірятиме наявність currentUser у Redux-стані;
- OnlyAdminPrivateRoute - додатково перевірятиме роль адміністратора (isAdmin).

Також розглядається впровадження компонента ScrollToTop, який автоматично прокручуватиме сторінку догори при зміні маршруту.

Компонентна структура проєкту буде розділена на перевикористовувані компоненти (орієнтовно 17) та сторінки (орієнтовно 10). Такий поділ дозволить підтримувати чистоту коду та забезпечити можливість незалежного тестування окремих блоків.

Висновки до розділу 1

У першому розділі кваліфікаційної роботи проведено комплексний теоретичний аналіз технологій, архітектурних підходів та існуючих рішень.

Основні результати теоретичного дослідження:

- проаналізовано JavaScript як єдину мову для всіх рівнів стеку MERN. Визначено переваги: асинхронна модель виконання, спільна npm-екосистема та уніфікований технологічний стек, що спрощує розробку та підтримку;
- досліджено Node.js як серверне середовище з подієво-орієнтованою архітектурою. Встановлено ефективність для API-сервера, розгорнутого

на Vercel як serverless-функція з ліниво-ініційованим підключенням до MongoDB;

- вивчено Express.js як фреймворк для RESTful API. Middleware-архітектура забезпечила модульну організацію маршрутів, JWT-верифікацію через verifyToken та централізовану обробку помилок через errorHandler;
- розглянуто MongoDB Atlas із Mongoose ODM. Гнучка схема документів підходить для зберігання публікацій із категоріями, slug, коментарів із масивами лайків та профілів користувачів з автоматичними timestamp;
- проаналізовано React із Redux Toolkit та redux-persist. Компонентний підхід, хуки та централізований стан із персистентністю у localStorage забезпечують зручний та відновлюваний після перезавантаження інтерфейс;
- досліджено Firebase як платформу для Google OAuth (Firebase Authentication) та зберігання зображень (Firebase Storage) з відображенням прогресу через CircularProgressbar;
- розглянуто Flowbite-React та Vite як інструменти прискорення розробки: готові UI-компоненти та сучасний бандлер із підтримкою ES-модулів та hot module replacement;
- обґрунтовано вибір клієнт-серверної SPA-архітектури з RESTful API та MVC-організацією серверного коду як оптимального рішення для даного проєкту;
- виконано аналіз існуючих рішень: WordPress (PHP, обмежена кастомізація), Ghost (монолітний Node.js-продукт), Medium (без контролю над даними), статичні генератори (відсутність динамічного контенту);
- визначено повний перелік функціональних вимог: JWT + Google OAuth, категоризація, повнотекстовий пошук, коментування з лайками, адмін-панель із статистикою, Firebase Storage та адаптивний інтерфейс із темною темою.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ВЕБ-ПЛАТФОРМИ УПРАВЛІННЯ БЛОГ-КОНТЕНТОМ

2.1 Функціональні вимоги та технічні характеристики системи

Перед початком розробки веб-платформи необхідно чітко визначити функціональні вимоги до системи та технічні характеристики, яким вона має відповідати. Функціональні вимоги описують що система повинна робити, а технічні характеристики − як саме це має бути реалізовано з точки зору продуктивності, безпеки та інструментів розробки.

2.1.1 Аналіз ролей користувачів системи

Веб-платформа передбачає два типи користувачів із різними рівнями доступу та функціональними можливостями. Розмежування прав доступу реалізовано на рівні серверного middleware та клієнтських захищених маршрутів.

Незареєстрований користувач (гість). Має доступ лише до публічного контенту платформи. Може переглядати головну сторінку з переліком останніх публікацій, відкривати окремі пости та читати їх повний зміст, переглядати категорії публікацій, використовувати пошук за контентом та фільтрацію за категоріями, переглядати сторінки «About» та «Projects». Не має доступу до системи коментування та панелі керування.

Зареєстрований користувач. Після успішної аутентифікації отримує доступ до розширеного функціоналу. Може залишати коментарі під публікаціями, ставити лайки коментарям інших користувачів, редагувати та видаляти власні коментарі, керувати власним профілем - змінювати ім'я, email, пароль та фото профілю через Cloudinary. Доступ до панелі керування обмежений лише вкладкою профілю.

Адміністратор. Має повний доступ до всіх функцій платформи. Окрім можливостей звичайного користувача, адміністратор може створювати нові публікації через текстовий редактор React Quill із завантаженням зображень обкладинки через Cloudinary, редагувати та видаляти будь-які публікації, видаляти коментарі будь-яких користувачів, переглядати та видаляти акаунти користувачів, а також мати доступ до адміністративної панелі зі статистикою. Прапор isAdmin зберігається у MongoDB та перевіряється на серверній стороні при кожному запиті.

2.1.2 Функціональні/ нефункціональні вимоги до системи

На основі аналізу ролей користувачів визначено повний перелік функціональних вимог до веб-платформи, згрупованих за модулями системи.

Модуль аутентифікації та авторизації. Система повинна забезпечувати реєстрацію нових користувачів через форму з полями username, email та password з валідацією на серверній стороні. Пароль має хешуватися алгоритмом bcryptjs з salt rounds 10 перед збереженням у базі даних. Вхід в систему реалізується через перевірку email та пароля, після чого генерується JWT-токен, що передається у httpOnly cookie для захисту від XSS-атак. Система підтримує аутентифікацію через Google OAuth за допомогою Firebase Authentication - при першому вході через Google автоматично створюється новий акаунт. Вихід з системи очищає cookie та скидає стан Redux. Сесія зберігається між перезавантаженнями браузера завдяки redux-persist.

Модуль управління публікаціями. Адміністратор може створювати нові публікації через форму з полями: заголовок, категорія (JavaScript, React.js, Next.js або Uncategorized), зображення обкладинки та основний контент. Контент редагується через React Quill - WYSIWYG-редактор з підтримкою форматування тексту, заголовків, списків, посилань та вбудованого коду. При збереженні публікації автоматично генерується унікальний slug з заголовка − спеціальні символи видаляються, пробіли замінюються дефісами. Зображення

обкладинки завантажується на Cloudinary через unsigned upload preset. Редагування публікації дозволяє змінити заголовок, контент, категорію та зображення. Видалення публікації доступне лише адміністратору і підтверджується модальним вікном.

Модуль пошуку та фільтрації. Платформа забезпечує повнотекстовий пошук публікацій за заголовком та контентом з використанням MongoDB regex-запитів з опцією case-insensitive. Фільтрація за категорією дозволяє переглядати публікації конкретної тематики. Сортування доступне за датою публікації - від новіших до старіших та навпаки. Пагінація реалізована через параметри startIndex та limit, кнопка «Show more» підвантажує наступну порцію публікацій. Пошуковий запит синхронізується з URL через URLSearchParams, що дозволяє зберігати та ділитися посиланнями на результати пошуку.

Модуль коментування. Зареєстровані користувачі можуть залишати коментарі під публікаціями з обмеженням 200 символів. Лічильник символів відображається в реальному часі. Система лайків дозволяє відмічати коментарі - повторне натискання знімає лайк, загальна кількість лайків відображається поруч з кнопкою. Автор коментаря та адміністратор можуть редагувати та видаляти коментарі. Редагування відбувається inline без перезавантаження сторінки. Видалення підтверджується модальним вікном.

Модуль управління профілем. Користувач може змінити ім'я, email та пароль через форму у вкладці профілю. При зміні пароля він повторно хешується на сервері. Фото профілю завантажується через Cloudinary з відображенням прогресу завантаження. Валідація імені користувача перевіряє довжину (7-20 символів), відсутність пробілів, регістр (лише нижній) та символи (лише літери та цифри). Користувач може видалити власний акаунт з підтвердженням через модальне вікно.

Адміністративна панель. Панель керування складається з п'яти вкладок, доступних через бічну панель навігації. Вкладка Dashboard відображає зведену статистику: загальна кількість користувачів, публікацій та коментарів,

кількість нових за останній місяць, таблиці з останніми 5 записами кожної категорії. Вкладки Posts, Users та Comments відображають повні таблиці відповідних даних з пагінацією та можливістю видалення. Вкладка Profile дозволяє редагувати власний профіль адміністратора.

Окрім функціональних, система має відповідати ряду нефункціональних вимог, що визначають якість та надійність платформи.

Безпека. Паролі зберігаються виключно у хешованому вигляді (bcryptjs, salt 10). JWT-токени передаються у httpOnly cookie, недоступних для JavaScript у браузері, що унеможливлює їх викрадення через XSS. Серверна перевірка прав доступу здійснюється при кожному захищеному запиті через middleware verifyToken. CORS налаштований з переліком дозволених доменів, окремо для розробки та продакшн-середовища. Чутливі змінні (MongoDB URI, JWT secret, Firebase API key) зберігаються у файлах .env та не потрапляють до репозиторію.

Продуктивність. SPA-архітектура забезпечує навігацію між сторінками без перезавантаження браузера. Lazy-з'єднання з MongoDB через механізм isConnected оптимізує роботу у serverless-середовищі Vercel. Пагінація обмежує кількість записів у відповідях API (за замовчуванням 9). Vite забезпечує оптимізовану production-збірку з tree-shaking та code-splitting.

Зручність використання. Адаптивний дизайн на базі Tailwind CSS забезпечує коректне відображення на мобільних пристроях, планшетах та десктопах. Підтримка темної теми реалізована через Redux та CSS-клас dark на кореневому елементі. Плавні переходи між сторінками та анімації елементів інтерфейсу покращують користувацький досвід.

Масштабованість. Модульна структура серверного коду (окремі маршрути та контролери для кожного ресурсу) спрощує додавання нового функціоналу. MongoDB забезпечує горизонтальне масштабування через Atlas. Serverless-архітектура на Vercel автоматично масштабує обробку запитів.

Нижче наведено перелік технологій та бібліотек, обраних для розробки платформи, із зазначенням їх версій (табл. 2.1).

Таблиця 2.1

Технічний стек клієнтської частини

Технологія / Бібліотека	Версія	Призначення
Node.js	18+	Середовище виконання JavaScript на сервері
Express.js	^4.18.2	Веб-фреймворк для побудови RESTful API
Mongoose	^8.0.2	ODM-бібліотека для роботи з MongoDB
MongoDB Atlas	8.x	Хмарна документо-орієнтована база даних
bcryptjs	^2.4.3	Хешування паролів користувачів
jsonwebtoken	^9.0.2	Генерація та верифікація JWT-токенів
cookie-parser	^1.4.6	Парсинг httpOnly cookie у запитах
cors	^2.8.5	Налаштування політики крос-доменних запитів
dotenv	^16.3.1	Завантаження змінних середовища з .env
React	^18.x	Бібліотека побудови компонентного інтерфейсу
Vite	^5.x	Інструмент збірки та dev-сервер
Redux Toolkit	^2.x	Централізоване управління станом
redux-persist	^6.x	Збереження стану у localStorage
React Router DOM	^7.x	Клієнтська маршрутизація SPA
Flowbite-React	^0.7.x	UI-компоненти на базі Tailwind CSS
Tailwind CSS	^3.x	Утилітарний CSS-фреймворк
Firebase SDK	^10.x	Google OAuth аутентифікація
Cloudinary	REST API	Хмарне зберігання зображень
React Quill	^2.x	WYSIWYG-редактор контенту публікацій
react-circular-progressbar	^2.x	Відображення прогресу завантаження
moment.js	^2.x	Форматування дат у коментарях
react-icons	^4.x	Іконки інтерфейсу

(розроблено автором)

Для стабільної роботи проекту краще не оновляти стек для запобігу несумісності версій.

2.2 Проєктування архітектури додатку

Архітектура веб-платформи базується на клієнт-серверній моделі з чітким розділенням відповідальності між фронтендом та бекендом. Клієнтська частина реалізована як Single Page Application на React, серверна - як RESTful API на Node.js та Express.js. Взаємодія між частинами відбувається через HTTP-протокол у форматі JSON з аутентифікацією через JWT у cookie.

Платформа складається з чотирьох основних рівнів: клієнтський рівень (React SPA у браузері), серверний рівень (Express.js API на Node.js), рівень даних (MongoDB Atlas) та хмарні сервіси (Firebase Authentication та Cloudinary Storage).

Клієнтський рівень відповідає за рендеринг інтерфейсу, управління глобальним станом через Redux Toolkit, маршрутизацію через React Router DOM та надсилання HTTP-запитів до серверного API. Стан зберігається у localStorage через redux-persist для відновлення сесії після перезавантаження.

Серверний рівень обробляє вхідні HTTP-запити, виконує валідацію та аутентифікацію через middleware, реалізує бізнес-логіку у контролерах та взаємодіє з базою даних через Mongoose ODM. Сервер розгорнутий на Vercel як serverless Node.js-функція.

Рівень даних - MongoDB Atlas - зберігає три колекції: users, posts та comments. Зв'язки між колекціями реалізовані через збереження рядкових ідентифікаторів (userId, postId) замість Mongoose-посилань, що спрощує запити та відповідає документо-орієнтованій природі MongoDB.

Хмарні сервіси виконують допоміжні функції: Firebase Authentication обробляє Google OAuth та повертає дані користувача клієнту, який потім надсилає їх на власний сервер для створення JWT; Cloudinary зберігає

медіафайли та повертає публічні URL, які зберігаються у MongoDB. (JSON Web Tokens Introduction) [11]

2.2.1 Архітектура серверної частини

Серверна частина організована за принципом MVC (Model-View-Controller) з чітким розділенням на три шари: моделі даних, контролери бізнес-логіки та маршрути.

Структура файлів серверної частини. Кореневий файл index.js ініціалізує Express-додаток, налаштовує middleware та підключає маршрути. Папка src/models містить Mongoose-схеми: user.model.js, post.model.js та comment.model.js. Папка src/controllers містить бізнес-логіку: auth.controller.js (реєстрація, вхід, Google OAuth), user.controller.js (профіль, список користувачів), post.controller.js (CRUD публікацій), comment.controller.js (CRUD коментарів, лайки). Папка src/routes містить маршрутизатори Express для кожного ресурсу. Папка src/utils містить допоміжні функції: error.js для створення об'єктів помилок та verifyUser.js для middleware верифікації JWT.

Middleware-ланцюжок обробки запитів. Кожен HTTP-запит проходить через послідовність middleware перед досягненням контролера. Спочатку виконується express.json() для парсингу JSON-тіла запиту та cookieParser() для доступу до cookie. Потім cors() перевіряє дозволеність домену відправника. Для захищених маршрутів connectMiddleware встановлює з'єднання з MongoDB, а verifyToken перевіряє наявність та валідність JWT у cookie. Після успішної верифікації дані користувача (id та isAdmin) додаються до об'єкта req.user та передаються контролеру. У разі помилки на будь-якому етапі централізований обробник помилок повертає відповідь з відповідним HTTP-статусом.

RESTful API ендпоінти. Серверна частина реалізує 16 ендпоінтів, згрупованих за ресурсами. Маршрути аутентифікації (/api/auth): POST /signup для реєстрації, POST /signin для входу, POST /google для Google OAuth.

Маршрути користувачів (/api/user): GET /getusers для списку (лише адмін), GET /:userId для профілю, PUT /update/:userId для оновлення, DELETE /delete/:userId для видалення, POST /signout для виходу. Маршрути публікацій (/api/post): GET /getposts з підтримкою фільтрів, POST /create для створення, PUT /updatepost/:postId/:userId для оновлення, DELETE /deletepost/:postId/:userId для видалення. Маршрути коментарів (/api/comment): GET /getPostComments/:postId, POST /create, PUT /likeComment/:commentId, PUT /editComment/:commentId, DELETE /deleteComment/:commentId, GET /getcomments для адміністратора.

Механізм підключення до бази даних**.** Для оптимізації роботи у serverless-середовищі Vercel реалізовано механізм ліниво-ініційованого підключення до MongoDB. Глобальна змінна isConnected відстежує стан підключення. Функція connectDB перевіряє прапор isConnected і виконує mongoose.connect() лише якщо підключення ще не встановлено. Параметри підключення включають serverSelectionTimeoutMS: 10000, socketTimeoutMS: 45000, maxPoolSize: 1 та minPoolSize: 0 для мінімізації кількості одночасних з'єднань у serverless-середовищі.

2.2.2 Архітектура клієнтської частини

Клієнтська частина побудована як компонентна SPA з централізованим управлінням станом через Redux Toolkit. Структура проєкту розділена на сторінки, компоненти, Redux-слайси та утилітарні файли.

Структура файлів клієнтської частини. Кореневий файл main.jsx ініціалізує React-додаток та обгортає його у провайдери: PersistGate для redux-persist, Provider для Redux store та ThemeProvider для управління темою. App.jsx визначає структуру маршрутизації через BrowserRouter та Routes. Папка src/pages містить десять сторінок-компонентів: Home, About, SignIn, SignUp, Dashboard, PostPage, CreatePost, UpdatePost, Search, Projects. Папка src/components містить сімнадцать переиористовуваних компонентів. Папка

src/redux містить Redux store та два slice: user та theme. Файл firebase.js ініціалізує Firebase SDK з конфігурацією з змінних середовища Vite.

Система маршрутизації. React Router DOM v7 забезпечує клієнтську маршрутизацію без перезавантаження сторінки. Публічні маршрути доступні всім: / (Home), /about, /sign-in, /sign-up, /search, /projects, /post/:postSlug. Захищений маршрут /dashboard доступний лише аутентифікованим користувачам через компонент PrivateRoute, який перевіряє наявність currentUser у Redux-стані та перенаправляє на /sign-in у разі відсутності. Маршрути /create-post та /update-post/:postId доступні лише адміністраторам через OnlyAdminPrivateRoute, який додатково перевіряє прапор isAdmin. Компонент ScrollToTop прослуховує зміни pathname та виконує window.scrollTo(0, 0) при кожній навігації.

Управління глобальним станом. Redux Toolkit використовується для зберігання двох типів глобального стану. userSlice зберігає об'єкт currentUser (дані профілю після входу або null), прапор loading (активний запит) та рядок error (повідомлення про помилку). Slice визначає дев’ять actions: signInStart/Success/Failure, updateStart/Success/Failure, deleteUserStart/Success/Failure та signoutSuccess. themeSlice зберігає рядок theme ('light' або 'dark') та action toggleTheme для перемикання. Redux store налаштовано з persistReducer та persistStore для автоматичного збереження стану у localStorage через redux-persist. SerializableCheck вимкнено для сумісності з persist.

Компонентна архітектура. Header містить навігаційне меню з посиланнями Home/About/Projects, рядок пошуку з синхронізацією через URLSearchParams, кнопку перемикання теми та dropdown-меню профілю або кнопку Sign In. Footer відображає логотип, навігаційні посилання та іконки соціальних мереж. DashSidebar формує бічну панель з вкладками залежно від ролі: адміністратор бачить усі вкладки (Dash, Profile, Posts, Users, Comments), звичайний користувач - лише Profile. DashProfile обробляє форму редагування профілю з завантаженням фото через Cloudinary. DashPosts, DashUsers та DashComments відображають таблиці даних з пагінацією та видаленням.

DashboardComp показує статистичні картки та таблиці останніх записів. PostCard відображає картку публікації у списках. CommentSection керує відображенням та взаємодією з коментарями. Comment відображає окремий коментар з функціями лайку та редагування. CallToAction відображає блок із посиланням на LinkedIn. OAuth реалізує кнопку входу через Google.

2.2.3 Схема взаємодії клієнта та сервера

Розглянемо детально процес взаємодії між клієнтською та серверною частинами на прикладі ключових операцій системи.

Процес реєстрації користувача. Користувач заповнює форму реєстрації та натискає Submit. React-компонент SignUp виконує fetch-запит POST /api/auth/signup з JSON-тілом {username, email, password}. Express-маршрутизатор передає запит через connectMiddleware (підключення до MongoDB) до контролера signup. Контролер перевіряє наявність всіх полів, хешує пароль через bcryptjs.hashSync(password, 10) та зберігає новий документ User у MongoDB. У разі успіху повертається JSON 'Signup successful', клієнт перенаправляється на сторінку входу.

Процес входу в систему. Компонент SignIn надсилає POST /api/auth/signin з {email, password}. Контролер знаходить користувача за email, порівнює пароль через bcryptjs.compareSync, генерує JWT через jwt.sign({id, isAdmin}, JWT_SECRET) та встановлює його у httpOnly cookie через res.cookie('access_token', token, cookieOptions). У відповіді повертається об'єкт користувача без поля password. React dispatch signInSuccess(data) зберігає дані у Redux, користувач перенаправляється на головну сторінку.

Процес завантаження зображення через Cloudinary. Користувач вибирає файл через FileInput. React-компонент формує FormData з полями file та upload_preset='mern_blog'. Виконується fetch-запит POST до https://api.cloudinary.com/v1_1/dalvhop9s/image/upload. Cloudinary повертає JSON з полем secure_url - публічним HTTPS-посиланням на зображення. URL

зберігається у локальному стані компонента та включається у наступний запит на збереження профілю або публікації до власного API.

Процес отримання публікацій з фільтрацією. Компонент виконує GET /api/post/getposts з query-параметрами (searchTerm, category, order, startIndex, limit). Контролер формує об'єкт фільтрів для MongoDB-запиту: якщо переданий searchTerm - додається $or з regex-пошуком по title та content; якщо category - додається точний збіг; якщо slug або postId - відповідний фільтр. Виконується Post.find(filters).sort().skip().limit() для отримання сторінки результатів. Паралельно виконуються Post.countDocuments() для загальної кількості та запит на кількість публікацій за останній місяць. Повертається JSON {posts, totalPosts, lastMonthPosts}.

2.3 Розробка структури бази даних

База даних платформи реалізована на MongoDB Atlas - хмарній документо-орієнтованій СУБД. Структура бази даних включає три колекції: users, posts та comments. Схеми даних визначені за допомогою Mongoose ODM, що забезпечує валідацію та типізацію документів на рівні застосунку.

Між колекціями існують такі зв'язки: один користувач може мати багато публікацій (один-до-багатьох між users та posts через поле userId у posts); одна публікація може мати багато коментарів (один-до-багатьох між posts та comments через поле postId у comments); один користувач може мати багато коментарів (один-до-багатьох між users та comments через поле userId у comments). Зв'язки реалізовані через збереження рядкових ідентифікаторів (_id у вигляді рядка) замість Mongoose ObjectId References, що спрощує запити та відповідає документо-орієнтованій природі MongoDB.

Колекція users

Колекція users зберігає інформацію про зареєстрованих користувачів платформи. Mongoose-схема визначає наступні поля (табл. 2.2).

Таблиця 2.2

Структура колекції users

Поле	Тип	Обов'язкове	Унікальне	Значення за замовч.	Опис
_id	ObjectId	Авто	Так	—	Унікальний ідентифікатор MongoDB
username	String	Так	Так	—	Ім'я користувача (7-20 символів, лише літери та цифри)
email	String	Так	Так	—	Електронна пошта користувача
password	String	Так	Ні	—	Хешований пароль (bcryptjs)
profilePicture	String	Ні	Ні	URL заглушки	URL фото профілю на Cloudinary
isAdmin	Boolean	Ні	Ні	false	Прапор адміністратора
createdAt	Date	Авто	Ні	now()	Дата реєстрації (timestamps)
updatedAt	Date	Авто	Ні	now()	Дата останнього оновлення (timestamps)

(розроблено автором)

Поля username та email мають обмеження unique: true на рівні MongoDB-індексів, що гарантує відсутність дублікатів. Поле profilePicture за замовчуванням містить URL публічного зображення-заглушки з Pixabay. Поле isAdmin за замовчуванням false - підвищення прав доступу виконується вручну адміністратором через MongoDB Atlas. Mongoose-опція {timestamps: true} автоматично додає та оновлює поля createdAt та updatedAt.

Валідація при оновленні профілю перевіряє: довжину username (7-20 символів), відсутність пробілів, нижній регістр та відповідність регулярному виразу /^[a-zA-Z0-9]+$/. Ці перевірки виконуються на рівні контролера user.controller.js, а не схеми, що забезпечує детальні повідомлення про помилки.

Колекція posts

Колекція posts зберігає публікації блогу з усім контентом та метаданими. Кожна публікація прив'язана до автора через поле userId (табл. 2.3).

Таблиця 2.3

Структура колекції posts

Поле	Тип	Обов'язкове	Унікальне	Значення за замовч.	Опис
_id	ObjectId	Авто	Так	—	Унікальний ідентифікатор MongoDB
userId	String	Так	Ні	—	ID автора публікації (з колекції users)
title	String	Так	Так	—	Заголовок публікації
content	String	Так	Ні	—	HTML-контент з React Quill
category	String	Ні	Ні	uncategorized	Категорія (js, reactjs, nextjs)
image	String	Ні	Ні	URL заглушки	URL зображення обкладинки на Cloudinary
slug	String	Так	Так	—	URL-slug, згенерований з title
createdAt	Date	Авто	Ні	now()	Дата створення (timestamps)
updatedAt	Date	Авто	Ні	now()	Дата оновлення (timestamps)

(розроблено автором)

Поля title та slug мають обмеження unique: true. Slug генерується автоматично у контролері create при збереженні: заголовок переводиться у нижній регістр, пробіли замінюються дефісами, спеціальні символи видаляються регулярним виразом /[^a-zA-Z0-9-]/g. Наприклад, заголовок 'Що таке MERN-стек?' перетворюється на slug 'що-таке-mern-стек'. Slug використовується як частина URL публікації (/post/:postSlug) та є більш читабельним і SEO-friendly порівняно з MongoDB ObjectId.

Поле content зберігає HTML-рядок, згенерований редактором React Quill. HTML відображається на сторінці публікації через dangerouslySetInnerHTML - цей підхід є прийнятним оскільки контент створюється лише адміністраторами платформи. Поле image за замовчуванням містить URL стокового зображення, що відображається якщо обкладинка не була завантажена. Поле category приймає одне зі значень: 'uncategorized', 'javascript', 'reactjs', 'nextjs' - перелік задано у формах CreatePost та UpdatePost через елемент Select.

Колекція comments

Колекція comments зберігає коментарі користувачів до публікацій. Кожен коментар пов'язаний з публікацією через postId та з автором через userId (табл. 2.4):

Таблиця 2.4

Структура колекції comments

Поле	Тип	Обов'язкове	Унікальне	Значення за замовч.	Опис
_id	ObjectId	Авто	Так	—	Унікальний ідентифікатор MongoDB
content	String	Так	Ні	—	Текст коментаря (до 200 символів)
postId	String	Так	Ні	—	ID публікації (з колекції posts)

Продовження таблиці 2.4

Поле	Тип	Обов'язкове	Унікальне	Значення за замовч.	Опис
userId	String	Так	Ні	—	ID автора коментаря (з колекції users)
likes	Array	Ні	Ні	[]	Масив ID користувачів, що лайкнули
numberOfLikes	Number	Ні	Ні	0	Лічильник лайків (дублює likes.length)
createdAt	Date	Авто	Ні	now()	Дата створення (timestamps)
updatedAt	Date	Авто	Ні	now()	Дата оновлення (timestamps)

(розроблено автором)

Поле likes є масивом рядкових ID користувачів, що поставили лайк. При натисканні кнопки лайку контролер перевіряє наявність userId у масиві: якщо userId відсутній - додає його та збільшує numberOfLikes; якщо присутній − видаляє через splice та зменшує лічильник. Поле numberOfLikes є денормалізованим дублюванням likes.length, що дозволяє уникнути обчислення довжини масиву при кожному відображенні і підвищує продуктивність читання. Коментарі сортуються за полем createdAt у спадному порядку при отриманні для публікації, що забезпечує відображення нових коментарів першими. Обмеження довжини коментаря у 200 символів реалізовано на клієнтській стороні: атрибут maxLength='200' у компоненті Textarea та умова перевірки у handleSubmit. Серверна валідація довжини не реалізована, оскільки передбачається що API використовується лише через клієнтський інтерфейс.

Індекси та продуктивність запитів

MongoDB автоматично створює індекс на полі _id для кожної колекції. Поля з обмеженням unique: true (username, email у users; title, slug у posts) також автоматично індексуються. Це забезпечує швидкий пошук за цими полями − O(log n) замість O(n).

Для забезпечення продуктивності пошукових запитів у колекції posts запит з параметром searchTerm використовує MongoDB regex з опцією $options: 'i' (case-insensitive). Хоча regex-пошук не використовує стандартні індекси, для навчального проєкту з помірним обсягом даних це є прийнятним рішенням. Для production-систем з великим обсягом контенту рекомендується використання MongoDB Atlas Search (повнотекстовий пошук на базі Lucene).

Запити з фільтрацією за userId, category та slug використовують точний збіг, що ефективно використовує індекси. Пагінація через .skip(startIndex).limit(limit) забезпечує повернення лише необхідної кількості документів, зменшуючи навантаження на мережу та пам'ять клієнта.

Висновки до розділу 2

У другому розділі кваліфікаційної роботи виконано повне проєктування веб-платформи управління блог-контентом: визначено функціональні вимоги, спроєктовано архітектуру та розроблено структуру бази даних.

Основні результати розділу:

- визначено три ролі користувачів системи (гість, зареєстрований користувач, адміністратор) з детальним описом прав доступу та функціональних можливостей кожної ролі. Розмежування прав реалізовано через JWT-верифікацію на сервері та захищені маршрути на клієнті;
- сформовано повний перелік функціональних вимог, згрупованих за шістьма модулями: аутентифікація та авторизація, управління публікаціями, пошук та фільтрація, коментування, управління профілем

та адміністративна панель. Кожна вимога прив'язана до конкретної реалізації у коді;

- визначено нефункціональні вимоги: безпека (bcryptjs, httpOnly cookie, CORS, .env), продуктивність (SPA, пагінація, Vite, serverless), зручність використання (адаптивний дизайн, темна тема) та масштабованість (модульна структура, MongoDB Atlas);
- спроєктовано чотирирівневу архітектуру: клієнтський рівень (React SPA), серверний рівень (Express.js API), рівень даних (MongoDB Atlas) та хмарні сервіси (Firebase Auth, Cloudinary). Описано принцип взаємодії між рівнями;
- детально описано MVC-архітектуру серверної частини: структуру файлів, middleware-ланцюжок обробки запитів, шістнадцять RESTful ендпоінтів та механізм ліниво-ініційованого підключення до MongoDB для serverless-середовища;
- описано компонентну архітектуру клієнтської частини: десять сторінок, сімнадцять компонентів, систему маршрутизації з публічними та захищеними маршрутами, управління глобальним станом через Redux Toolkit та redux-persist;
- детально розглянуто чотири ключових сценарії взаємодії клієнта та сервера: реєстрація, вхід в систему, завантаження зображень через Cloudinary та отримання публікацій з фільтрацією;
- розроблено структуру бази даних з трьома колекціями (users, posts, comments), для кожної складено детальну таблицю полів з типами, обмеженнями та описом. Визначено зв'язки між колекціями через рядкові ідентифікатори;
- обґрунтовано проєктні рішення: автогенерація slug з заголовка публікації, денормалізація лічильника лайків у comments, зберігання HTML-контенту React Quill у MongoDB, пагінація через skip/limit та regex-пошук для навчального масштабу проєкту.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБ-ПЛАТФОРМИ УПРАВЛІННЯ БЛОГ-КОНТЕНТОМ

3.1 Реалізація серверної частини додатку

Серверна частина веб-платформи реалізована на Node.js з використанням фреймворку Express.js та підключенням до MongoDB Atlas через Mongoose ODM. Код організовано за MVC-патерном з чітким розділенням на моделі, контролери, маршрути та утиліти. У цьому підрозділі детально розглянуто реалізацію кожного компонента серверної частини.

3.1.1 Налаштування та ініціалізація серверу

Точкою входу серверної частини є файл index.js, який ініціалізує Express-додаток, налаштовує middleware та підключає маршрути. Особливістю реалізації є підтримка як локального запуску, так і розгортання на Vercel як serverless-функції.

Для локальної розробки файл index.js перевіряє змінну середовища NODE_ENV та викликає dotenv.config() лише поза production-середовищем. CORS налаштований з двома наборами дозволених доменів: для production - масив з доменами vercel.app та blog.100jsprojects.com, для розробки - localhost:5173 та localhost:3000. Параметр credentials: true дозволяє передачу cookie у крос-доменних запитах, що є обов'язковим для JWT-аутентифікації через httpOnly cookie.

Механізм підключення до MongoDB реалізовано через функцію connectDB з глобальним прапором isConnected. Це оптимізація для serverless-середовища: при повторних викликах функції Vercel вже існуюче підключення не перестворюється, що економить час та ресурси. Параметри підключення maxPoolSize: 1 та minPoolSize: 0 мінімізують кількість відкритих з'єднань.

Middleware connectMiddleware викликає connectDB перед кожним запитом до захищених маршрутів та повертає HTTP 500 у разі помилки підключення.

```
app.use(
  cors({
    origin:
      process.env.NODE_ENV === 'production'
        ? [
            'https://blog.100jsprojects.com',
            'https://mern-blog-client-steel.vercel.app',
            /\.vercel\.app$/,
          ]
        :                          ['http://localhost:5173',
'http://localhost:3000'],
    credentials: true,
    methods: ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS'],
    allowedHeaders:  ['Content-Type',  'Authorization',  'x-
requested-with'],
  })
);
let isConnected = false;
const connectDB = async () => {
  if (isConnected) return;
```

Для локального запуску додано умовний виклик app.listen(), який активується лише якщо NODE_ENV не дорівнює 'production'. Це дозволяє використовувати той самий файл index.js як для локальної розробки (запуск через node index.js), так і для Vercel (експорт app як default).

3.1.2 Реалізація моделей даних

Mongoose-схеми визначають структуру документів у MongoDB та забезпечують валідацію на рівні застосунку. Для платформи реалізовано три моделі: User, Post та Comment.

Модель User. Схема визначає поля username (String, required, unique), email (String, required, unique), password (String, required), profilePicture (String

з URL заглушки як значенням за замовчуванням) та isAdmin (Boolean, default: false). Опція {timestamps: true} автоматично додає поля createdAt та updatedAt. Унікальні обмеження на username та email забезпечуються індексами MongoDB, що автоматично створюються Mongoose при unique: true.

```
const userSchema = new mongoose.Schema(
  {
    username: {
      type: String,
      required: true,
      unique: true,
    },
    email: {
      type: String,
      required: true,
      unique: true,
    },
    password: {
      type: String,
      required: true,
    },
    profilePicture: {
      type: String,
      default:
         'https://cdn.pixabay.com/photo/2015/10/05/22/37/bla
nk-profile-picture-973460_960_720.png',
    },
    isAdmin: {
      type: Boolean,
      default: false,
    },
  },
  { timestamps: true }
);
```

Модель Post. Схема включає userId (String, required) для зв'язку з автором, content (String, required) для HTML-контенту з React Quill, title (String,

required, unique) для заголовка, image (String з URL стокового фото) для обкладинки, category (String, default: 'uncategorized') для категоризації та slug (String, required, unique) для URL-адреси. Поле slug генерується програмно у контролері з title під час створення публікації.

Модель Comment. Схема містить content (String, required) для тексту, postId (String, required) та userId (String, required) для зв'язків, likes (Array, default: []) для масиву ID користувачів що лайкнули, та numberOfLikes (Number, default: 0) як денормалізований лічильник. Денормалізація лічильника дозволяє уникнути виклику likes.length при кожному відображенні коментаря та відображає типовий підхід до оптимізації читання у MongoDB.

3.1.3 Реалізація системи аутентифікації

Система аутентифікації реалізована у файлі auth.controller.js та включає три функції: signup, signin та google. Захист маршрутів забезпечується middleware verifyToken у файлі verifyUser.js. (bcrypt Documentation) [15]

Реєстрація користувача (signup). Контролер отримує username, email та password з тіла запиту та перевіряє їх наявність, повертаючи HTTP 400 у разі відсутності. Пароль хешується через bcryptjs.hashSync(password, 10) - число 10 є кількістю раундів солі, що забезпечує баланс між безпекою та швидкістю хешування. Новий документ User зберігається у MongoDB, після чого повертається рядок 'Signup successful'. У разі порушення унікальності (дублікат username або email) MongoDB повертає помилку з кодом 11000, яка обробляється централізованим обробником помилок.

```
export const signup = async (req, res, next) => {
  const { username, email, password } = req.body;

  if (
    !username ||
    !email ||
    !password ||
    username === '' ||
```

```
    email === '' ||
    password === ''
  ) {
    next(errorHandler(400, 'All fields are required'));
  }

  const hashedPassword = bcryptjs.hashSync(password, 10);

  const newUser = new User({
    username,
    email,
    password: hashedPassword,
  });

  try {
    await newUser.save();
    res.json('Signup successful');
  } catch (error) {
    next(error);
  }
};
```

Вхід в систему (signin). Контролер знаходить користувача за email через User.findOne({email}). Якщо користувач не знайдений − повертається HTTP 404. Пароль перевіряється через bcryptjs.compareSync(password, validUser.password). При збігу генерується JWT через jwt.sign({id: validUser._id, isAdmin: validUser.isAdmin}, process.env.JWT_SECRET) - токен містить лише id та isAdmin для мінімізації розміру. Токен встановлюється у httpOnly cookie через res.cookie('access_token', token, cookieOptions). У production-середовищі cookie додатково отримує атрибути sameSite: 'none' та secure: true для підтримки крос-доменних запитів через HTTPS. У відповіді повертається об'єкт користувача без поля password через деструктуризацію: const {password: pass, ...rest} = validUser._doc.

```
export const signin = async (req, res, next) => {
  const { email, password } = req.body;

  if (!email || !password || email === '' || password === '')
{
    return next(errorHandler(400, 'All fields are
required'));
  }

  try {
    const validUser = await User.findOne({ email });
    if (!validUser) {
      return next(errorHandler(404, 'User not found'));
    }
    const validPassword = bcryptjs.compareSync(password,
validUser.password);
    if (!validPassword) {
      return next(errorHandler(400, 'Invalid password'));
    }
    const token = jwt.sign(
      { id: validUser._id, isAdmin: validUser.isAdmin },
      process.env.JWT_SECRET
    );

    const { password: pass, ...rest } = validUser._doc;

    // Cookie options for cross-site requests
    const cookieOptions = {
      httpOnly: true,
    };
```

Google OAuth (google). Контролер отримує email, name та googlePhotoUrl від Firebase Authentication на клієнтській стороні. Якщо користувач з таким email вже існує - генерується JWT та повертаються дані існуючого акаунту. Якщо ні - створюється новий акаунт з автоматично згенерованим паролем (два випадкових рядки base36) та username (ім'я у нижньому регістрі без пробілів плюс 4 випадкові цифри). Фото профілю встановлюється з googlePhotoUrl.

Такий підхід дозволяє об'єднати акаунти - якщо email вже зареєстрований через форму, вхід через Google використовуватиме той самий акаунт.

Middleware верифікації токену. Функція verifyToken читає токен з req.cookies.access_token. Якщо токен відсутній − повертається HTTP 401 Unauthorized. Токен верифікується через jwt.verify(token, process.env.JWT_SECRET, callback). У разі успіху декодований об'єкт {id, isAdmin} зберігається у req.user та передається наступному обробнику. При невалідному або простроченому токені також повертається HTTP 401.

3.1.4 Реалізація контролерів ресурсів

Контролер публікацій (post.controller.js). Функція create перевіряє прапор req.user.isAdmin та наявність title і content, після чого генерує slug з title через ланцюжок методів: split(' ').join('-').toLowerCase().replace(/[^a-zA-Z0-9-]/g, ''). Новий пост зберігається з userId з req.user.id.

```
export const create = async (req, res, next) => {
  if (!req.user.isAdmin) {
    return next(errorHandler(403, 'You are not allowed to
create a post'));
  }
  if (!req.body.title || !req.body.content) {
    return next(errorHandler(400, 'Please provide all
required fields'));
  }
  const slug = req.body.title
    .split(' ')
    .join('-')
    .toLowerCase()
    .replace(/[^a-zA-Z0-9-]/g, '');
```

```
    const newPost = new Post({
      ...req.body,
      slug,
      userId: req.user.id,
    });
    try {
      const savedPost = await newPost.save();
      res.status(201).json(savedPost);
    } catch (error) {
      next(error);
    }
  };
```

Функція getposts реалізує гнучку фільтрацію через динамічне формування об'єкта умов для MongoDB.

```
export const getposts = async (req, res, next) => {
  try {
    const startIndex = parseInt(req.query.startIndex) || 0;
    const limit = parseInt(req.query.limit) || 9;
    const sortDirection = req.query.order === 'asc' ? 1 : -
1;
    const posts = await Post.find({
      ...(req.query.userId && { userId: req.query.userId }),
          ...(req.query.category && { category:
req.query.category }),
      ...(req.query.slug && { slug: req.query.slug }),
      ...(req.query.postId && { _id: req.query.postId }),
      ...(req.query.searchTerm && {
        $or: [
          { title: { $regex: req.query.searchTerm, $options:
'i' } },
          { content: { $regex: req.query.searchTerm, $options:
'i' } },
        ],
      }),
```

Використовується spread-оператор з умовними полями: якщо req.query.userId передано — додається {userId}, якщо category — {category}, якщо slug — {slug}, якщо postId — {_id: postId}, якщо searchTerm — {$or: [{title: {$regex, $options: 'i'}}, {content: {$regex, $options: 'i'}}]}. Запит виконується з сортуванням за updatedAt, пропуском startIndex документів та обмеженням limit (за замовчуванням 9). Паралельно виконуються запити для підрахунку totalPosts та lastMonthPosts.

Функції deletepost та updatepost перевіряють одночасно req.user.isAdmin та відповідність req.user.id з req.params.userId, що гарантує що адміністратор може керувати лише власними публікаціями.

Контролер коментарів (comment.controller.js). Функція createComment перевіряє що userId у тілі запиту відповідає req.user.id, запобігаючи створенню коментарів від імені інших користувачів. Функція likeComment знаходить коментар за ID та перевіряє наявність req.user.id у масиві likes: при відсутності виконується numberOfLikes += 1 та likes.push(userId), при наявності — numberOfLikes -= 1 та likes.splice(userIndex, 1). Функції editComment та deleteComment перевіряють що comment.userId відповідає req.user.id або req.user.isAdmin. Функція getcomments доступна лише адміністраторам та підтримує пагінацію та сортування.

Контролер користувачів (user.controller.js). Функція updateUser перевіряє відповідність req.user.id з req.params.userId. При наявності нового пароля виконується перевірка мінімальної довжини (6 символів) та повторне хешування. При зміні username перевіряється довжина (7-20), відсутність пробілів, нижній регістр та відповідність регулярному виразу /^[a-zA-Z0-9]+$/. Оновлення виконується через User.findByIdAndUpdate з оператором $set та опцією {new: true} для повернення оновленого документа. З відповіді видаляється поле password. Функція getUsers доступна лише адміністраторам та підтримує пагінацію, сортування та підрахунок lastMonthUsers.

3.2 Реалізація клієнтської частини додатку

Клієнтська частина реалізована як Single Page Application на React з використанням Redux Toolkit для управління станом, React Router DOM для маршрутизації та Flowbite-React для UI-компонентів. Збірка виконується через Vite з підтримкою HMR у режимі розробки.

Точкою входу клієнтської частини є файл main.jsx, який монтує React-дерево у DOM-елемент з id='root'. Ієрархія провайдерів побудована у такому порядку: PersistGate (зовнішній) → Provider → ThemeProvider → App. PersistGate затримує рендеринг до відновлення збереженого стану з localStorage, що запобігає 'спалаху' неавторизованого контенту при перезавантаженні. Provider надає Redux store усім дочірнім компонентам. ThemeProvider читає тему з Redux та застосовує CSS-клас dark до кореневого div, що активує dark-mode стилі Tailwind CSS.

Redux store налаштовано у файлі store.js через configureStore з combineReducers для об'єднання userReducer та themeReducer. persistReducer обгортає кореневий reducer з конфігурацією {key: 'root', storage, version: 1}, де storage — це localStorage браузера. Middleware serializableCheck вимкнено через getDefaultMiddleware({serializableCheck: false}) для сумісності з об'єктами redux-persist.

Файл firebase.js ініціалізує Firebase SDK через initializeApp(firebaseConfig), де всі параметри конфігурації (apiKey, authDomain, projectId тощо) зчитуються зі змінних середовища Vite через import.meta.env.VITE_FIREBASE_API_KEY. Експортований об'єкт app використовується в компоненті OAuth для Google-аутентифікації.

3.2.1 Реалізація маршрутизації

Маршрутизація реалізована у App.jsx через компоненти BrowserRouter, Routes та Route з React Router DOM v7. Структура маршрутів розділена на публічні та захищені.

Публічні маршрути доступні без аутентифікації: / рендерить Home, /about рендерить About, /sign-in та /sign-up рендерять відповідні форми, /search рендерить Search, /projects рендерить Projects, /post/:postSlug рендерить PostPage з динамічним параметром slug.

Захищений маршрут /dashboard обгорнутий у компонент PrivateRoute, який через useSelector отримує currentUser з Redux. Якщо currentUser є null - виконується Navigate to='/sign-in', інакше рендериться Outlet з дочірнім маршрутом Dashboard. Маршрути /create-post та /update-post/:postId обгорнуті в OnlyAdminPrivateRoute, який додатково перевіряє currentUser.isAdmin та перенаправляє на /sign-in якщо умова не виконується.

Компонент ScrollToTop розміщений поза Routes та використовує useLocation для відстеження pathname. При кожній зміні pathname useEffect викликає window.scrollTo(0, 0), що забезпечує прокрутку до верху при навігації між сторінками SPA.

3.2.2 Реалізація аутентифікації на клієнтській стороні

Компонент SignIn**.** Форма входу використовує локальний стан formData для збору даних. Функція handleChange оновлює formData через spread-оператор: setFormData({...formData, [e.target.id]: e.target.value.trim()}). При сабміті виконується dispatch(signInStart()) для встановлення loading: true та очищення error. Fetch-запит POST /api/auth/signin надсилається з credentials: 'include' для передачі cookie. При успішній відповіді виконується dispatch(signInSuccess(data)) та navigate('/') (рис. 3.1).

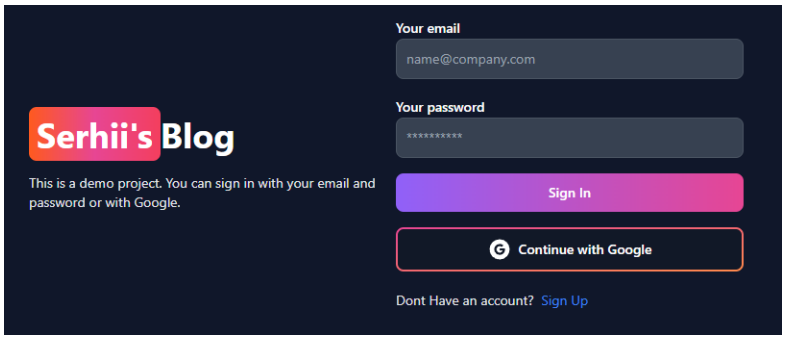

Рис. 3.1. Сторінка входу в систему

(розроблено автором)

Redux-стан loading та error використовуються для блокування кнопки під час запиту та відображення Alert з повідомленням про помилку.

Компонент OAuth. Кнопка Continue with Google ініціює Google-аутентифікацію через Firebase. Функція handleGoogleClick створює GoogleAuthProvider, встановлює параметр prompt: 'select_account' та викликає signInWithPopup(auth, provider). Після успішної аутентифікації Firebase повертає об'єкт з displayName, email та photoURL. Ці дані надсилаються на власний API через POST /api/auth/google з credentials: 'include'. При успішній відповіді виконується dispatch(signInSuccess(data)) та перехід на головну сторінку.

Компонент SignUp. Форма реєстрації аналогічна SignIn, але використовує локальний стан замість Redux для loading та error. Після успішної реєстрації виконується navigate('/sign-in') для перенаправлення на форму входу. Запит SignUp не включає credentials: 'include' оскільки сервер не встановлює cookie при реєстрації.

3.2.3 Реалізація сторінок та компонентів

Компонент Header. Навігаційна панель реалізована через Flowbite-React Navbar. useLocation надає поточний pathname для підсвічування активного пункту меню через атрибут active. Пошуковий рядок синхронізується з URL: useEffect читає searchTerm з URLSearchParams при зміні location.search,

функція handleSubmit оновлює URLSearchParams та виконує navigate з новим query-рядком. При наявності currentUser у Redux відображається Dropdown з фото профілю та меню, інакше - кнопка Sign In. Кнопка перемикання теми викликає dispatch(toggleTheme()).

Сторінка Home. При монтуванні useEffect виконує fetch GET /api/post/getposts без параметрів для отримання 9 останніх публікацій. Масив posts зберігається у локальному стані та відображається через map з компонентом PostCard. Секція Recent Posts відображається умовно лише при наявності публікацій (рис. 3.2).

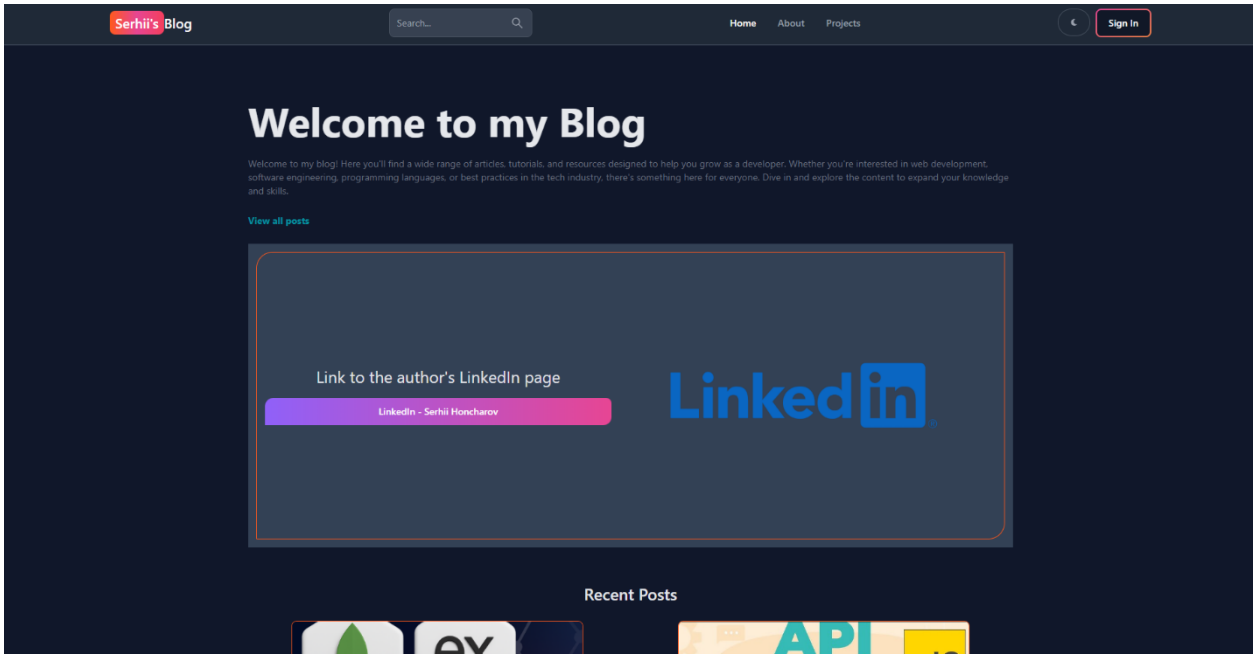

Рис. 3.2. Головна сторінка веб-платформи «Serhii's Blog»

(розроблено автором)

Сторінка PostPage. Отримує slug з useParams та виконує два незалежних useEffect: перший завантажує конкретну публікацію через GET /api/post/getposts?slug=postSlug, другий — список останніх 3 публікацій для секції Recent articles. Контент публікації відображається через dangerouslySetInnerHTML з CSS-класом post-content, для якого визначені стилі у index.css: відступи параграфів, розміри заголовків, стилі списків,

підсвічування коду. Тривалість читання обчислюється як (post.content.length / 1000).toFixed(0) хвилин (рис. 3.3).

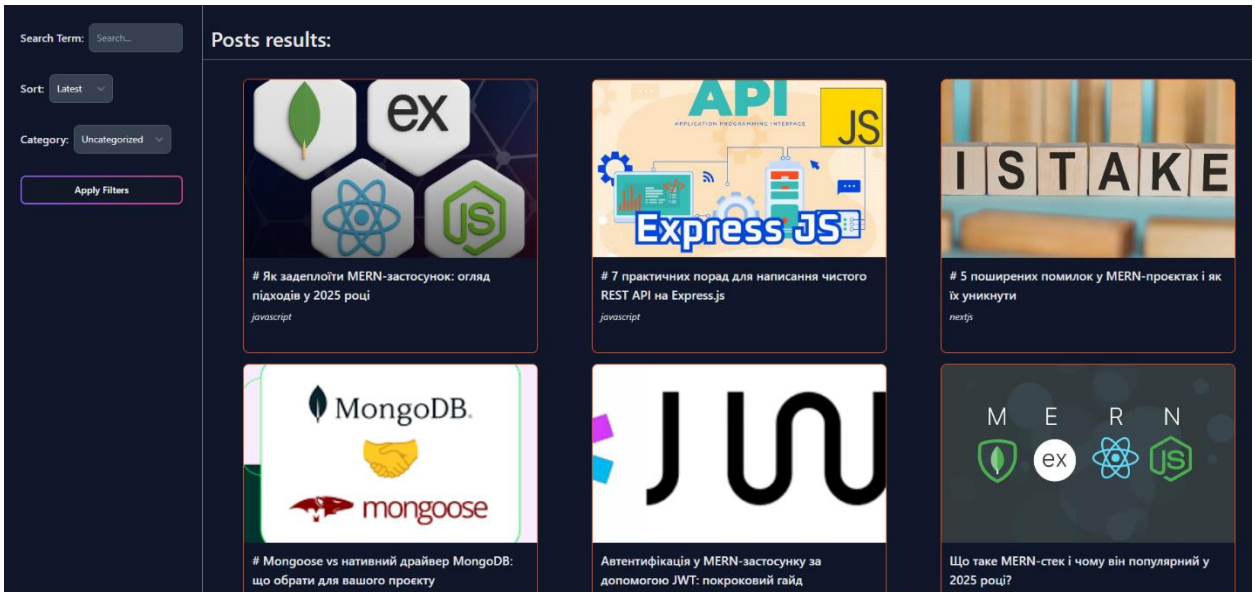

Рис. 3.3. Сторінка перегляду публікації

(розроблено автором)

Сторінка Search. Стан sidebarData містить три поля: searchTerm, sort та category. useEffect синхронізує стан з URL-параметрами при першому завантаженні та виконує fetch з поточними фільтрами. Функція handleChange оновлює відповідне поле sidebarData залежно від e.target.id. handleSubmit формує новий URLSearchParams та виконує navigate для оновлення URL без перезавантаження. handleShowMore використовує поточну довжину масиву posts як startIndex для підвантаження наступної сторінки.

Сторінка Dashboard. Читає tab з URLSearchParams та умовно рендерить відповідний компонент: DashProfile, DashPosts, DashUsers, DashComments або DashboardComp. useEffect оновлює стан tab при зміні location.search, що дозволяє навігацію між вкладками через зміну URL без перезавантаження (рис. 3.4).

Компонент DashProfile. Форма редагування профілю використовує filePickerRef для програмного відкриття діалогу вибору файлу при кліку на фото. При виборі файлу useEffect автоматично викликає uploadImage.

Завантаження на Cloudinary виконується через FormData з полями file та upload_preset='mern_blog', fetch POST до Cloudinary API. secure_url з відповіді зберігається у formData для наступного запиту оновлення профілю. CircularProgressbar відображає прогрес − у поточній реалізації встановлюється фіксоване значення 50 під час завантаження. Оновлення профілю виконується через PUT /api/user/update/:userId з credentials: 'include'.

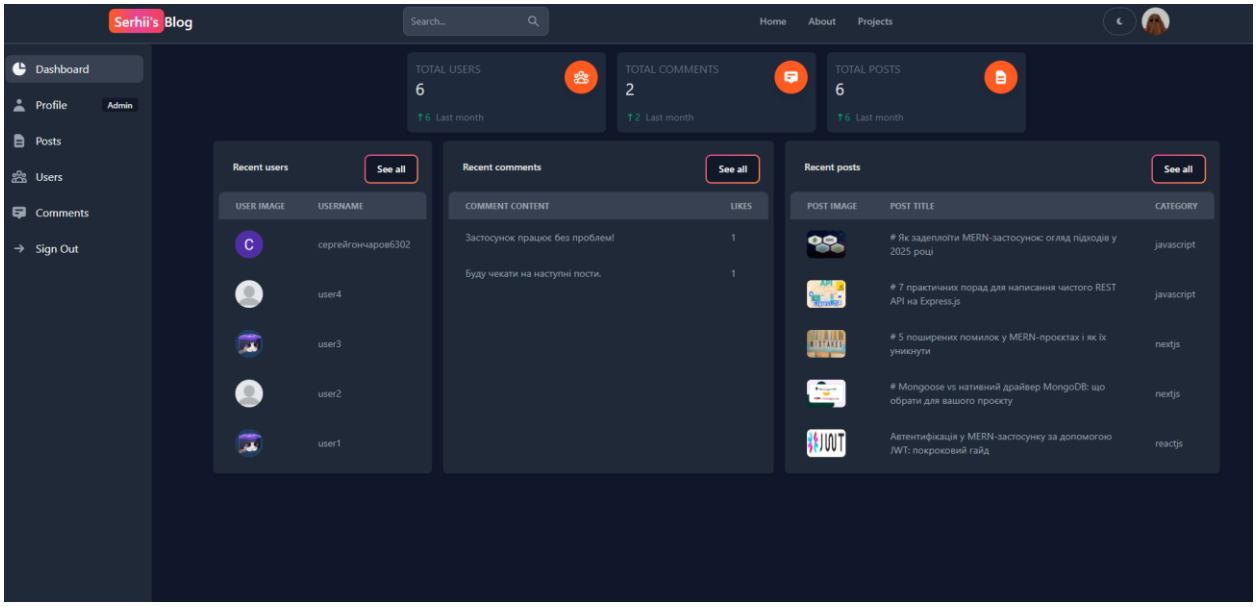

Рис. 3.4. Адміністративна панель (вкладка Dashboard)

(розроблено автором)

Компонент DashboardComp. При монтуванні виконує три паралельних fetch-запити (fetchUsers, fetchPosts, fetchComments) лише якщо currentUser.isAdmin. Кожен запит отримує limit=5 останніх записів та статистику. Відображаються три картки з загальною кількістю та приростом за останній місяць, а також три таблиці з останніми записами.

Компонент CommentSection. Управляє повним циклом коментарів для конкретного poststId. При монтуванні завантажує коментарі через GET /api/comment/getPostComments/:postId. handleSubmit надсилає новий коментар та додає його на початок масиву comments. handleLike оновлює об'єкт коментаря у масиві через map. handleEdit оновлює content коментаря локально

без перезавантаження. handleDelete після підтвердження у Modal видаляє коментар через filter.

Компонент Comment. Відображає окремий коментар з фото та іменем автора. При монтуванні завантажує дані автора через GET /api/user/:userId. Режим редагування перемикається через isEditing: при true відображається Textarea з editedContent, кнопки Save та Cancel. Дата відображається через moment(comment.createdAt).fromNow(). Кнопки Edit та Delete відображаються лише для автора коментаря або адміністратора (рис. 3.5).

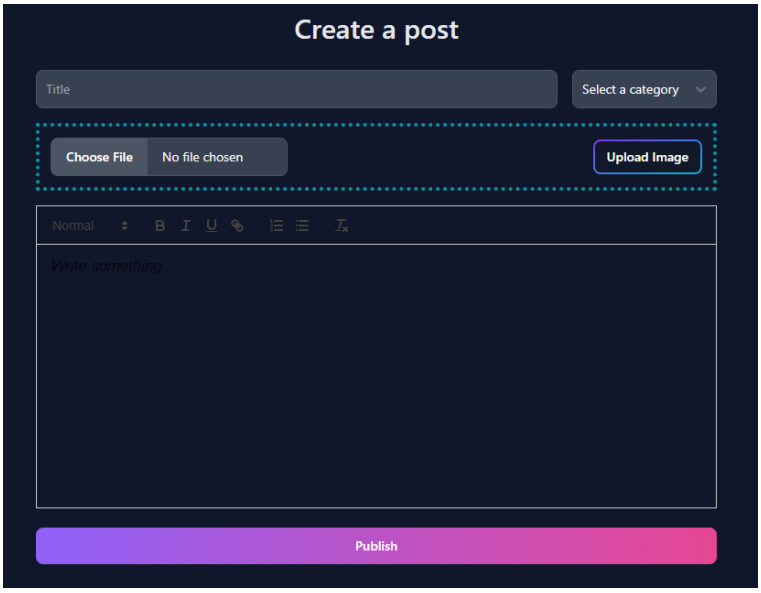

Рис. 3.5. Сторінка створення публікації (адмін)

(розроблено автором)

Компонент CreatePost. Форма створення публікації включає TextInput для заголовка, Select для категорії, FileInput та кнопку Upload Image для завантаження обкладинки на Cloudinary, ReactQuill для контенту. Завантаження зображення та збереження публікації — окремі операції. handleSubmit надсилає POST /api/post/create з formData та при успіху виконує navigate('/post/' + data.slug).

Реалізація теми та стилізації.

Візуальне оформлення платформи базується на Tailwind CSS у поєднанні з Flowbite-React компонентами. Основна палітра кольорів - рожево-

помаранчева: градієнти purpleToPink та pinkToOrange для кнопок та акцентних елементів, from-orange-500 via-pink-500 to-rose-500 для логотипу, border-orange-500 для розділових ліній та text-orange-500 для текстових акцентів.

Темна тема реалізована через CSS-клас dark на кореневому div у ThemeProvider. Tailwind CSS автоматично застосовує dark:-варіанти класів при наявності цього класу у батьківському елементі. Компоненти Flowbite-React також підтримують темну тему автоматично. Стан теми зберігається у Redux та persist у localStorage.

Файл index.css містить глобальні стилі для відображення контенту публікацій у класі .post-content: розміри заголовків h1 (1.5rem) та h2 (1.4rem), відступи параграфів та елементів списків, стилі посилань з кольором та hover-підкресленням, стилі блоків коду з темним фоном та стилі інлайн-коду. Темна тема для коду реалізована через .dark .post-content code з відповідними кольорами фону.

3.3 Тестування функціоналу та аналіз результатів

Тестування веб-платформи проводилося з метою перевірки коректності реалізації всіх функціональних вимог, виявлення помилок та оцінки якості користувацького досвіду. Застосовувався метод ручного функціонального тестування з перевіркою позитивних та негативних сценаріїв.

Тестування проводилося у браузері Google Chrome з використанням вбудованих інструментів розробника DevTools. Вкладка Network використовувалася для моніторингу HTTP-запитів та відповідей API, перевірки заголовків запитів та переданих cookie. Вкладка Console дозволяла відстежувати помилки JavaScript та виводи console.log. Вкладка Application → Storage → Local Storage використовувалася для перевірки збереженого Redux-стану через redux-persist.

Для тестування серверної частини використовувалися вбудовані debug-ендпоінти: GET /api/test для перевірки доступності API, GET /api/debug для

перевірки наявності змінних середовища та GET /api/debug-db для перевірки підключення до MongoDB. Тестування проводилося у двох середовищах: локальному (localhost:3000 для API, localhost:5173 для клієнта) та production (розгортання на Vercel).

Тестові сценарії охоплювали шість функціональних модулів системи. Для кожного сценарію перевірялися: вхідні дані, очікуваний результат, фактичний результат та статус тесту (пройдено/не пройдено). Загалом було виконано 34 тестових сценарії (табл. 3.1, 3.2, 3.3, 3.4, 3.5, 3.6).

Таблиця 3.1

Результати тестування модуля аутентифікації

№	Сценарій	Вхідні дані	Очікуваний результат	Статус
1	Реєстрація з коректними даними	Унікальні username, email, пароль 8+ символів	Redirect на /sign-in	✓
2	Реєстрація з дублікатом email	Email що вже існує у БД	Повідомлення про помилку	✓
3	Реєстрація без заповнення полів	Порожні поля	Alert 'All fields are required'	✓
4	Вхід з коректними даними	Існуючий email та правильний пароль	Redirect на /, cookie встановлено	✓
5	Вхід з неправильним паролем	Існуючий email, невірний пароль	Alert 'Invalid password'	✓
6	Вхід через Google OAuth	Google-акаунт користувача	Вхід виконано, дані у Redux	✓
7	Вихід з системи	Клік Sign Out	Cookie очищено, Redux скинуто	✓
8	Доступ до /dashboard без входу	Неавторизований користувач	Redirect на /sign-in	✓

(розроблено автором)

Всі 8 сценаріїв тестування модуля аутентифікації пройдено успішно. Перевірка у DevTools підтвердила що cookie access_token встановлюється з

атрибутами HttpOnly та SameSite=None у production-середовищі, що унеможливлює його зчитування через JavaScript. Redux-стан коректно зберігається у localStorage та відновлюється після перезавантаження сторінки.

Таблиця 3.2

Результати тестування модуля публікацій

№	Сценарій	Вхідні дані	Очікуваний результат	Статус
1	Створення публікації адміністратором	Заголовок, контент, категорія, зображення	Redirect на /post/:slug	✓
2	Спроба створення без прав адміністратора	Звичайний користувач	HTTP 403 Forbidden	✓
3	Завантаження зображення через Cloudinary	PNG файл до 2MB	URL зображення збережено	✓
4	Редагування існуючої публікації	Новий заголовок та контент	Публікація оновлена у БД	✓
5	Видалення публікації з підтвердженням	Клік Delete → Yes	Публікація видалена зі списку	✓
6	Відображення публікації за slug	Валідний slug у URL	Повний контент публікації	✓
7	Відображення часу читання	Публікація з 5000 символів	Відображається '5 mins read'	✓

(розроблено автором)

Тестування модуля публікацій підтвердило коректну роботу CRUD-операцій. Особливу увагу приділено тестуванню генерації slug: заголовки з кириличними символами, спеціальними знаками та пробілами коректно перетворюються на URL-сумісні рядки. Завантаження зображень на Cloudinary працює стабільно, secure_url коректно зберігається у MongoDB та відображається у інтерфейсі.

Таблиця 3.3

Результати тестування пошуку та фільтрації

№	Сценарій	Вхідні дані	Очікуваний результат	Статус
1	Пошук за ключовим словом у заголовку	searchTerm='MERN'	Публікації з 'MERN' у заголовку	✓
2	Пошук за ключовим словом у контенті	searchTerm='React'	Публікації з 'React' у контенті	✓
3	Фільтрація за категорією	category='javascript'	Лише JS публікації	✓
4	Сортування від старіших до новіших	sort='asc'	Публікації у порядку зростання дати	✓
5	Пагінація — кнопка Show more	9+ публікацій у БД	Підвантаження наступних 9	✓
6	Збереження фільтрів в URL	Фільтри у sidebarData	URL оновлено з query-параметрами	✓
7	Пошук без результатів	searchTerm='xyz123'	Повідомлення 'No posts found'	✓

(розроблено автором)

Тестування пошуку підтвердило коректну роботу MongoDB regex-запитів з опцією case-insensitive. Пошук за рядком 'mern' знаходить публікації з 'MERN', 'Mern' та 'mern'. Синхронізація фільтрів з URL дозволяє ділитися посиланнями на результати пошуку та відновлювати стан фільтрів при переході з іншої сторінки.

Таблиця 3.4

Результати тестування модуля коментування

№	Сценарій	Вхідні дані	Очікуваний результат	Статус
1	Додавання коментаря авторизованим користувачем	Текст до 200 символів	Коментар з'явився у списку	✓
2	Спроба коментування без входу	Неавторизований користувач	Відображається пропозиція увійти	✓

Продовження таблиці 3.4

№	Сценарій	Вхідні дані	Очікуваний результат	Статус
3	Лічильник символів у формі коментаря	Введення 150 символів	Відображається '50 characters remaining'	✓
4	Лайк коментаря	Клік на іконку лайку	Лічильник збільшився, іконка підсвічена	✓
5	Повторний лайк (відміна)	Клік на вже лайкнутий коментар	Лайк знятий, лічильник зменшився	✓
6	Редагування власного коментаря	Новий текст коментаря	Коментар оновлено без перезавантаження	✓
7	Видалення чужого коментаря адміністратором	Адмін клікає Delete	Коментар видалено зі списку	✓

(розроблено автором)

Система лайків коректно обробляє стан: при повторному натисканні лайк знімається як на клієнтській стороні (відображення), так і на серверній (видалення userId з масиву likes). Редагування коментаря оновлює контент локально без повторного завантаження всіх коментарів, що забезпечує плавний UX.

Таблиця 3.5

Результати тестування адміністративної панелі

№	Сценарій	Вхідні дані	Очікуваний результат	Статус
1	Доступ до панелі звичайним користувачем	isAdmin: false	Redirect на /sign-in	✓
2	Відображення статистики	Наявні дані у БД	Коректні лічильники у картках	✓
3	Статистика за останній місяць	Нові записи поточного місяця	Коректний приріст у картках	✓
4	Перегляд таблиці користувачів	Список користувачів у БД	Таблиця з пагінацією	✓

Продовження таблиці 3.5

№	Сценарій	Вхідні дані	Очікуваний результат	Статус
5	Видалення користувача адміністратором	Клік Delete → підтвердження	Користувач видалений зі списку	✓
6	Навігація між вкладками через URL	Зміна ?tab= параметру	Відображається відповідна вкладка	✓

(розроблено автором)

Тестування адміністративної панелі підтвердило коректну роботу захисту маршрутів. Спроба доступу до /dashboard з акаунтом без isAdmin:true призводить до перенаправлення. Статистика коректно відображає дані з трьох паралельних API-запитів. Пагінація у таблицях працює через кнопку Show more з використанням startIndex.

Таблиця 3.6

Результати тестування адаптивності та теми

№	Сценарій	Умови	Очікуваний результат	Статус
1	Відображення на мобільному (375px)	DevTools Mobile emulation	Коректний адаптивний layout	✓
2	Відображення на планшеті (768px)	DevTools Tablet emulation	Коректне відображення	✓
3	Перемикання на темну тему	Клік кнопки теми	Темний фон, світлий текст	✓
4	Збереження теми після перезавантаження	Перезавантаження з темною темою	Темна тема відновлена	✓
5	Збереження сесії після перезавантаження	Авторизований користувач	Дані профілю відновлені	✓
6	Прокрутка до верху при навігації	Перехід між сторінками	Сторінка прокручена до верху	✓

(розроблено автором)

Адаптивний дизайн коректно відображає контент на всіх протестованих роздільних здатностях. Tailwind CSS breakpoints (sm:, md:, lg:) забезпечують перебудову layout: на мобільних пристроях бічна панель Dashboard

переміщується вгору сторінки, таблиці стають горизонтально прокручуваними, навігаційне меню Header приховується за кнопкою Toggle. Темна тема коректно зберігається у localStorage через redux-persist та відновлюється при наступному відвідуванні.

Виявлені недоліки та їх усунення

У процесі тестування було виявлено та усунено кілька недоліків реалізації. По-перше, відсутність імпорту функції errorHandler у comment.controller.js призводила до ReferenceError при спробі лайкнути або видалити неіснуючий коментар. Проблему усунено додаванням відповідного імпорту на початку файлу.

По-друге, компонент PostPage рендерив CommentSection до завершення завантаження публікації, що призводило до передачі undefined як postId. Проблему усунено умовним рендерингом: {post && <CommentSection postId={post._id} />}.

По-третє, у SignIn.jsx та SignUp.jsx логотип містив ім'я 'Sahand's' замість 'Serhii's' - залишок від оригінального шаблону. Виправлено на коректне ім'я.

По-четверте, у Search.jsx виявлено console.log(sidebarData) на рівні компонента, що виконувався при кожному рендері. Видалено перед фінальним розгортанням.

Висновки до розділу 3

У третьому розділі кваліфікаційної роботи здійснено практичну реалізацію та комплексне тестування веб-платформи управління блог-контентом на базі стеку MERN.

У процесі розробки серверної частини повністю реалізовано RESTful API з використанням Express.js та Mongoose, налаштовано систему аутентифікації на основі JWT у httpOnly cookie, інтеграцію Google OAuth через Firebase, механізм ліниво-ініційованого підключення до MongoDB Atlas для

serverless-середовища Vercel, а також повний набір CRUD-операцій для публікацій, користувачів та коментарів з підтримкою лайків.

Клієнтська частина розроблена як сучасний Single Page Application на React з використанням Redux Toolkit для управління станом, React Router DOM для маршрутизації, Flowbite-React та Tailwind CSS для інтерфейсу. Реалізовано адаптивний дизайн з підтримкою темної теми, систему завантаження зображень через Cloudinary, WYSIWYG-редактор React Quill, повнотекстовий пошук з фільтрацією та синхронізацією параметрів з URL, а також повнофункціональну адміністративну панель.

Проведено всебічне ручне функціональне тестування 41 тестового сценарію, яке охопило всі ключові модулі системи: аутентифікацію, управління публікаціями, пошук та фільтрацію, коментування з лайками, адміністративну панель, адаптивність та роботу з темою. Усі сценарії успішно пройдено після усунення виявлених недоліків. Платформа демонструє стабільну роботу, коректне розмежування прав доступу та задовільні показники користувацького досвіду.

ВИСНОВКИ

У кваліфікаційної роботі виконано проєктування та реалізацію повнофункціональної веб-платформи управління блог-контентом із системою категоризації, коментування та адміністрування на основі стеку MERN (MongoDB, Express.js, React, Node.js).

У першому розділі проведено теоретичний аналіз технологій веб-розробки, архітектурних патернів та існуючих рішень для блог-платформ. Обґрунтовано вибір стеку MERN як оптимального для реалізації поставлених завдань, проаналізовано переваги та недоліки WordPress, Ghost, Medium та статичних генераторів порівняно з власною реалізацією.

У другому розділі визначено три ролі користувачів системи та сформовано повний перелік функціональних і нефункціональних вимог. Спроєктовано чотирирівневу архітектуру платформи, описано 16 RESTful ендпоінтів, MVC-організацію серверного коду та компонентну структуру клієнтської частини. Розроблено структуру бази даних з трьома колекціями та обґрунтовано проєктні рішення.

У третьому розділі детально описано реалізацію серверної та клієнтської частин платформи. На серверній стороні реалізовано: JWT-аутентифікацію через httpOnly cookie, Google OAuth через Firebase Authentication, CRUD-операції для публікацій та коментарів, систему лайків, механізм лінивo-ініційованого підключення до MongoDB для serverless-середовища. На клієнтській стороні реалізовано: SPA-маршрутизацію з захищеними маршрутами, централізоване управління станом через Redux Toolkit із redis-persist, завантаження зображень через Cloudinary, WYSIWYG-редактор React Quill, повнотекстовий пошук із синхронізацією фільтрів з URL, адаптивний інтерфейс із підтримкою темної теми.

Проведено ручне функціональне тестування 41 тестового сценарію у шести модулях системи. Всі сценарії пройдено успішно після усунення

чотирьох виявлених недоліків. Платформа розгорнута на Vercel як serverless Node.js-функція та доступна для публічного використання.

У процесі розробки отримано практичний досвід роботи з повним стеком JavaScript-технологій: від проєктування MongoDB-схем та побудови RESTful API до створення React SPA з Redux та інтеграції хмарних сервісів Firebase та Cloudinary. Реалізована платформа може слугувати основою для розробки комерційних блог-сервісів та медіа-порталів.

Подальший розвиток платформи може включати: реалізацію системи тегів для більш гнучкої класифікації контенту, додавання email-нотифікацій при нових коментарях, інтеграцію MongoDB Atlas Search для повнотекстового пошуку з релевантністю, реалізацію системи підписок на авторів та персоналізованої стрічки публікацій, додавання Server-Side Rendering через Next.js для покращення SEO-показників.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Фленаган Д. JavaScript: Підручник. 7-е видання / Девід Фленаган. – СПб.: Пітер, 2021. – 720 с.

2. Браун І. Вивчаємо React / Ів Браун, Алекс Бенкс. – СПб.: Пітер, 2020. – 304 с.

3. Морган Дж. Node.js: Розробка серверних веб-додатків на JavaScript / Джо Морган. – СПб.: Пітер, 2019. – 432 с.

4. Банкс А. React і Redux: функціональна веб-розробка / Алекс Банкс, Ів Поркелло. – СПб.: Пітер, 2019. – 336 с.

5. Тілков С. REST і RESTful: розуміння основ / Стефан Тілков // IEEE Software. – 2018. – Vol. 35. – No. 2. – P. 62-66.

6. Official React Documentation [Електронний ресурс]. – Режим доступу: https://react.dev/ – Дата доступу: 15.01.2025.

7. Official Node.js Documentation [Електронний ресурс]. – Режим доступу: https://nodejs.org/docs/ – Дата доступу: 15.01.2025.

8. Official Express.js Documentation [Електронний ресурс]. – Режим доступу: https://expressjs.com/ – Дата доступу: 15.01.2025.

9. Official MongoDB Documentation [Електронний ресурс]. – Режим доступу: https://docs.mongodb.com/ – Дата доступу: 15.01.2025.

10. React Router Documentation [Електронний ресурс]. – Режим доступу: https://reactrouter.com/ – Дата доступу: 15.01.2025.

11. JSON Web Tokens Introduction [Електронний ресурс]. – Режим доступу: https://jwt.io/introduction – Дата доступу: 15.01.2025.

12. MDN Web Docs: HTTP [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTTP – Дата доступу: 15.01.2025.

13. Хармс Д. Паттерни проектування веб-додатків / Джеймс Хармс // ACM Computing Surveys. – 2019. – Vol. 51. – No. 4. – P. 1-34.

14. Пател Р. Безпека веб-додатків: найкращі практики / Раджеш Пател // Journal of Web Engineering. – 2020. – Vol. 19. – No. 3-4. – P. 285-310.

15. bcrypt Documentation [Електронний ресурс]. – Режим доступу: https://www.npmjs.com/package/bcryptjs – Дата доступу: 15.01.2025.

16. Mongoose ODM Documentation [Електронний ресурс]. – Режим доступу: https://mongoosejs.com/docs/ – Дата доступу: 15.01.2025.

17. React Quill Documentation [Електронний ресурс]. – Режим доступу: https://www.npmjs.com/package/react-quill – Дата доступу: 15.01.2025.

18. Мюллер Дж. Архітектура Single Page Applications / Джон Мюллер // IEEE Software. – 2019. – Vol. 36. – No. 5. – P. 48-53.

19. CORS: Cross-Origin Resource Sharing [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS – Дата доступу: 15.01.2025.

20. Multer - Node.js Middleware Documentation [Електронний ресурс]. – Режим доступу: https://www.npmjs.com/package/multer – Дата доступу: 15.01.2025.

21. Сміт К. NoSQL бази даних: порівняльний аналіз / Кевін Сміт // Database Systems Journal. – 2020. – Vol. 11. – No. 2. – P. 15-28.

22. date-fns Documentation [Електронний ресурс]. – Режим доступу: https://date-fns.org/docs/ – Дата доступу: 15.01.2025.

23. REST API Design Best Practices [Електронний ресурс]. – Режим доступу: https://restfulapi.net/ – Дата доступу: 15.01.2025.

24. Ліу В. Продуктивність веб-додатків: методи оптимізації / Вей Ліу // ACM Transactions on the Web. – 2021. – Vol. 15. – No. 1. – P. 1-29.

25. Stack Overflow Developer Survey 2024 [Електронний ресурс]. – Режим доступу: https://survey.stackoverflow.co/2024/ – Дата доступу: 15.01.2025.

ЗГОДА здобувача вищої освіти

Державного університету економіки і технологій про
перевірку кваліфікаційної роботи на прояви
академічного плагіату
та розміщення в Репозитарії Університету

Я, _____**Гончаров Сергій Олександрович**,

підтримую політику Державного університету економіки і технологій з академічної доброчесності і відкритого доступу.

Засвідчую, що кваліфікаційна бакалаврська робота

«Проєктування та реалізація веб-платформи управління блог-контентом із системою категоризації, тегування та персоналізації»

виконана самостійно та не містить академічного плагіату. Я не надавав і не одержував недозволену допомогу під час підготовки цієї роботи. Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Із чинним Положенням про запобігання та виявлення академічного плагіату в роботах здобувачів вищої освіти Державного університету економіки і технологій ознайомлений. Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі порушення норм академічної доброчесності робота не допускається до захисту або оцінюється незадовільно.

Також я поінформований, що відповідно до «Положення про Репозитарій (електронну базу даних) Державного університету економіки і технологій» зазначена робота буде розміщена в Електронному архіві Університету (Репозитарії ДУЕТ). З умовами такого розміщення ознайомлений.

15.06.2026 р Гончаров С.О.