Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА ІНТЕРАКТИВНОГО БЛОГУ З ВИКОРИСТАННЯМ REACT, NODE.JS ТА MONGODB

DEVELOPMENT OF AN INTERACTIVE BLOG USING REACT, NODE.JS AND MONGODB

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-41
Гольонко Максим Анатолійович

Керівник:
Ліщина Наталія Миколаївна

Кваліфікаційну роботу
допущено до захисту
«10» 06 2025р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *бакалавр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення*»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«20» 12 2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Гольонку Максиму Анатолійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *«Розробка інтерактивного блогу з використанням React, Node.js та Mongodb»*
Керівник роботи: к.т.н., доцент Ліщина Н. М.
затверджені наказом закладу вищої освіти від *«19» грудня 2024 р. № 474/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи
«10» червня 2025 р.
3. Вихідні дані до роботи: *WebStorm, MongoDB Atlas, Postman Git, Cloudinary, Mantine методичні вказівки до виконання кваліфікаційної роботи бакалавра.*
4. Зміст розрахунково-пояснювальної записки: розробити архітектуру застосунку на основі клієнт-серверної моделі, реалізувати структуру бази даних, проєктувати й реалізувати структуру бази даних розробити повний функціонал створення, редагування, видалення постів.
5. Перелік графічного матеріалу: *15 рисунків, 4 таблиці, 2 додатки.*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Ліщина Н. М.*		
Специфікація вимог до розробленої системи	*Ліщина Н. М.*		
Розробка об'єкта проектування	*Ліщина Н. М.*		
Нормоконтроль	*Вознюк А. В.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	3,78 %		
Академічна доброчесність	*Ліщина Н. М.*		

7. Дата видачі завдання *«20» грудня 2024 р.*

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 04.02.2025 р.	виконано
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 01.03.2025 р.	виконано
3	Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 15.03.2025 р.	виконано
4	Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних	до 29.03.2025 р.	виконано
5	Практична реалізація об'єкта проектування та розробка бази даних	до 26.04.2025 р.	виконано
6	Тестування та налагодження об'єкта проектування	до 03.05.2025 р.	виконано
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру	до 10.06.2025 р.	виконано

Здобувач вищої освіти _____ Максим ГОЛЬОНКО

Керівник кваліфікаційної роботи _____ Наталія ЛІЩИНА

АНОТАЦІЯ

Гольонко М. А. Розробка інтерактивного блогу з використанням React, Node.js та MongoDB. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі здійснено аналіз сучасного стану проблеми і сформульовано завдання. В другому розділі проведено аналіз, визначення вимог до розроблюваного програмного забезпечення, проектування програмного забезпечення та вибір засобів, методів і алгоритмів вирішення поставленого завдання. У третьому розділі описана практична реалізація об'єкта проєктування, здійснено тестування та налагодження інформаційно-комп'ютерної системи, розроблено базу даних, доданий захист інформаційно-комп'ютерної системи та описана документація для розгорнання вебдодатку.

Ключові слова: інтерактивний блог, вебзастосунок, React, Node.js, MongoDB, клієнт-серверна архітектура, розробка програмного забезпечення, база даних, проектування, інформаційна безпека, документація, тестування, фронтенд, бекенд, REST API, хмарні сервіси.

ABSTRACT

Holonko M. Development of an Interactive Blog Using React, Node.js and MongoDB. Manuscript. Qualification work of the bachelor of the OP "Software Engineering" specialty "Software Engineering". Lutsk National Technical University. Lutsk, 2025.

The bachelor's qualification thesis consists of an introduction, three chapters, conclusions, a list of references, and appendices.

The first chapter presents, the analysis of the current state of the problem is carried out and formulates the objectives. The second chapter , an analysis, definition of requirements for the developed software, software design and selection of tools, methods and algorithms for solving the task. The third chapter describes the practical implementation of the design object, tests and debugging of the information and computer system are carried out, a database is developed, protection of the information and computer system is added and documentation for deploying the web application is described.

Keywords: interactive blog, web site, React, Node.js, MongoDB, client-server architecture, software development, database, design, information security, documentation, testing, frontend, backend, REST API, cloud services.

ЗМІСТ

ВСТУП

Актуальність теми. Сучасний розвиток інформаційних технологій сприяє швидкому зростанню популярності вебдодатків, особливо в сфері створення інтерактивного контенту. Блоги стали невід'ємною частиною онлайн-простору, надаючи користувачам можливість ділитися думками, ідеями та досвідом. Серед основних технологій, що використовуються для розробки вебдодатків, особливе місце займають React, Node.js та MongoDB. React, як бібліотека для створення інтерфейсів, забезпечує високу продуктивність та зручність у розробці, тоді як Node.js дозволяє створювати масштабовані серверні рішення. MongoDB, у свою чергу, є потужною NoSQL базою даних, що забезпечує гнучкість у зберіганні та обробці даних. Актуальність теми розробки інтерактивного блогу обумовлена зростаючими вимогами до інтерактивності та швидкості вебдодатків. Сучасні користувачі очікують миттєвого реагування на свої дії, а також можливості взаємодії з контентом у реальному часі.

Світові тенденції свідчать про зростання популярності односторінкових додатків, які забезпечують більш плавний користувацький досвід. Багато компаній вже впроваджують подібні рішення, однак існує потреба в подальшому вдосконаленні методів розробки та інтеграції технологій для створення більш ефективних та безпечних рішень.

Метою кваліфікаційної роботи бакалавра є розробка інтерактивного блогу, який використовує технології React, Node.js та MongoDB, що дозволить створити сучасний, функціональний та зручний у використанні вебдодаток. Область застосування даного програмного продукту охоплює як індивідуальних авторів контенту, так і компанії, що прагнуть розвивати свою онлайн-присутність

Об'єктом кваліфікаційної роботи бакалавра є розробка інтерактивного блогу.

Предметом кваліфікаційної роботи бакалавра є практична реалізація інтерактивного блогу, що базується на сучасних вебтехнологіях, зокрема на

JavaScript бібліотеці React, Node.js фреймворку Express.js та базі даних MongoDB.

Завданням кваліфікаційної роботи бакалавра є:

– провести аналіз предметної області та сучасних інструментів веброзробки з урахуванням специфіки реалізації;

– розробити архітектуру застосунку на основі клієнт-серверної моделі взаємодії, з виокремленням ролей frontend, backend та бази даних;

– реалізувати клієнтську частину застосунку з використанням бібліотеки React, забезпечивши адаптивність, інтуїтивність та високу швидкодію інтерфейсу;

– створити серверну частину на базі Node.js із застосуванням фреймворку Express.js для реалізації логіки обробки HTTP-запитів, маршрутизації та бізнес-логіки;

– спроєктувати й реалізувати структуру бази даних у середовищі MongoDB, орієнтуючись на гнучкість зберігання динамічної інформації, пов’язаної з користувачами, публікаціями та їхніми властивостями;

– реалізувати функціональність реєстрації, входу в систему, автентифікації та верифікації користувачів із підтримкою розмежування прав доступу;

– забезпечити можливість створення, редагування, перегляду та видалення статей із дотриманням правил авторства;

– розгорнути готовий застосунок на хмарному сервері, інтегрувавши всі компоненти системи для демонстрації його працездатності в реальному середовищі.

Робота апробована шляхом участі у X Міжнародній науково-практичній конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025)», м. Луцьк, 23-24 травня 2025 р. (додаток А).

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

У сучасному інформаційному просторі вебблоги є важливим інструментом для створення, публікації та поширення контенту. Їх використовують як індивідуальні автори, так і великі організації для донесення новин, ведення технічної документації, формування експертних спільнот тощо. Попит на інтерактивні, зручні та швидкі вебсайти стимулює розробників створювати нові рішення, які базуються на сучасних технологіях веброзробки. Традиційно, для створення блогових платформ використовуються готові системи управління контентом. Ці платформи мають довгу історію розвитку і багатий функціонал, але з точки зору кастомізації, продуктивності та інтеграції з сучасними frontend- і backend-інструментами вони часто виявляються обмеженими. У багатьох випадках внесення змін до їхньої архітектури потребує складного втручання у внутрішню логіку системи, що ускладнює розширення функціоналу або інтеграцію зі сторонніми сервісами.

Водночас, останніми роками спостерігається перехід від класичних багатосторінкових вебсайтів до односторінкових застосунків, які реалізуються з використанням JavaScript-бібліотек і фреймворків нового покоління. Однією з найпопулярніших технологій у цьому напрямі є React, який дозволяє створювати реактивні інтерфейси на основі компонентного підходу [1]. Його популярність пояснюється високою продуктивністю, активною спільнотою та можливістю гнучкого контролю над кожним елементом інтерфейсу. React активно використовується як у великих корпоративних проєктах, так і в невеликих особистих застосунках, що свідчить про його універсальність.

На стороні сервера помітне домінування Node.js як середовища виконання JavaScript, що дозволяє реалізовувати backend-логіку з використанням тієї ж мови програмування, що й на клієнті [2]. У поєднанні з Express.js –

мінімалістичним, але потужним фреймворком – Node.js, який дозволяє швидко створювати RESTful API [3], які легко масштабуються та інтегруються з frontend-частиною. Усі ці технології доповнює MongoDB документно-орієнтована база даних [4], яка зберігає інформацію у форматі BSON, сумісному з JSON. Така структура ідеально підходить для застосунків, де структура даних може змінюватися динамічно, і не потребує складного налаштування схем, як це притаманно реляційним базам даних.

Аналіз наукових джерел, технічної документації та практичних рішень свідчить, що поєднання React, Node.js і MongoDB є одним із найефективніших підходів до розробки сучасних вебзастосунків. Це підтверджується наявністю великої кількості навчальних матеріалів, відкритих проєктів, а також комерційних продуктів, створених за цим принципом. Серед переваг такого підходу слід відзначити високу швидкодію, можливість асинхронної обробки даних, масштабованість і зручність для розробників завдяки використанню єдиної мови на клієнті та сервері. Також важливим чинником є активна підтримка з боку спільноти, яка забезпечує швидке вирішення технічних проблем і регулярне оновлення бібліотек.

Для порівняльного аналізу були розглянуті існуючі популярні блогові додатки та порівняти їх з реалізованим застосунком. Найпоширенішими рішеннями у сфері створення блогів залишаються такі системи управління контентом CMS, як WordPress, Drupal, а також Medium – платформа із закритим кодом, орієнтована на публікацію статей.

WordPress є найпопулярнішою CMS у світі, яка охоплює понад 40% всіх сайтів в Інтернеті. Вона забезпечує широкий спектр можливостей завдяки тисячам плагінів та шаблонів, що дозволяє створювати блоги без глибоких технічних знань [5]. Однак, WordPress має низку обмежень: складна кастомізація, низька продуктивність при великому обсязі контенту без оптимізації, потенційні проблеми з безпекою через сторонні плагіни. Крім того, більшість функціональності реалізується за допомогою PHP, що ускладнює інтеграцію з сучасними JavaScript-фреймворками.

Drupal позиціонується як більш гнучка та масштабована альтернатива WordPress і активно використовується у корпоративному та урядовому секторі [6]. Вона надає гнучкі механізми контролю доступу, потужну систему таксономії та багатомовність із коробки. Проте, складність розробки, крута крива навчання та обмеженість у реалізації сучасних SPA-застосунків робить її менш привабливою для проєктів, орієнтованих на сучасний інтерфейс і динамічну взаємодію з користувачем

Medium приклад повністю хмарного рішення, яке не потребує розгортання, оновлень чи налаштувань [7]. Платформа забезпечує привабливий інтерфейс, адаптивність і простоту використання. Водночас вона є закритою системою без можливості повної кастомізації функціоналу або структури бази даних. Усі користувачі змушені дотримуватись заданої моделі контенту, що унеможливлює створення нестандартних функцій, як-от, наприклад, реалізація адміністративної панелі з розмежуванням прав доступу чи інтеграція зі сторонніми сервісами.

На відміну від перелічених рішень у межах кваліфікаційної роботи буде реалізований вебзастосунок, як повноцінний SPA з використанням React для клієнтської частини, що забезпечує швидке відображення контенту, безперервну взаємодію з сервером та зручність редагування без перезавантаження сторінки. Backend буде реалізований на Node.js із застосуванням Express.js, що дозволяє легко масштабувати систему, а також реалізувати бізнес-логіку згідно з вимогами користувача. Структура бази даних на основі MongoDB надає змогу зберігати гнучкі, динамічні структури об’єктів – користувачів, публікацій, коментарів, ролей тощо. Важливо, що застосунок може бути розгорнутий на будь-якому хмарному сервері, включно з безкоштовними рішеннями на кшталт Render та Vercel, що забезпечує повну автономність і контроль з боку розробника.

Таким чином, рішення буде поєднувати переваги високої кастомізованості, сучасного UX, масштабованості та технологічної гнучкості. Воно перевершує готові CMS у тих випадках, коли йдеться про створення нестандартних

інтерфейсів, потребу в власному API або необхідність повного контролю над логікою доступу та зберіганням даних.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

У межах кваліфікаційної роботи поставлено такі цілі, досягнення яких забезпечить реалізацію інтерактивного блогу:

– провести аналіз сучасних інструментів веброзробки та порівняти їх з альтернативними технологіями, з урахуванням специфіки реалізації;

– розробити архітектуру застосунку на основі клієнт-серверної моделі взаємодії, з виокремленням ролей frontend, backend та бази даних;

– реалізувати клієнтську частину застосунку з використанням бібліотеки React, забезпечивши адаптивність, інтуїтивність та високу швидкодію інтерфейсу;

– створити серверну частину на базі Node.js із застосуванням фреймворку Express.js для реалізації логіки обробки HTTP-запитів, маршрутизації та бізнес-логіки;

– спроєктувати й реалізувати структуру бази даних у середовищі MongoDB, орієнтуючись на гнучкість зберігання динамічної інформації, пов’язаної з користувачами, публікаціями та їхніми властивостями;

– реалізувати функціональність реєстрації, входу в систему, автентифікації та верифікації користувачів із підтримкою розмежування прав доступу;

– забезпечити можливість створення, редагування, перегляду та видалення статей із дотриманням правил авторства;

– розгорнути готовий застосунок на хмарному сервері, інтегрувавши всі компоненти системи для демонстрації його працездатності в реальному середовищі.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

2.1.1 Методи виявлення і аналізу вимог

Під час розробки інтерактивного блогу було прийнято рішення реалізувати систему за клієнт-серверною архітектурною моделлю. Такий вибір зумовлений низкою переваг, які ця модель забезпечує як з технічної, так і з функціональної точки зору. У контексті вебзастосунку, клієнт-серверна модель дозволяє чітко розмежувати обов'язки між фронтендом (клієнтською частиною) та бекендом (серверною частиною), що забезпечує модульність, масштабованість і зручність у розробці, супроводі та подальшій експлуатації системи.

Клієнт-серверна модель є оптимальною для таких застосунків, як блоги, з огляду на такі аргументи:

– клієнт відповідає лише за інтерфейс та обробку взаємодії, а сервер – за бізнес-логіку та управління даними. Це дозволяє вести незалежну розробку та тестування обох частин системи;

– серверна частина може бути масштабована окремо від клієнтської, що є критично важливим при зростанні кількості користувачів або обсягів трафіку;

– зберігання та обробка даних виконується на сервері, що забезпечує кращий контроль над безпекою, захистом персональних даних і контролем доступу;

– клієнт-серверна архітектура дозволяє створити багато різних інтерфейсів, які можуть одночасно використовувати один серверний API;

– сервер можна оптимізувати для обробки запитів, кешування результатів, обробки зображень та інших завдань, що зменшує навантаження на клієнтський пристрій.

Вибір REST (Representational State Transfer) API як основної архітектурної моделі взаємодії між клієнтською та серверною частинами проєкту зумовлений

низкою технічних, архітектурних і практичних переваг, що повністю відповідають вимогам до розроблюваного вебзастосунку інтерактивного блогу.

Вибір REST API як основної архітектурної моделі взаємодії між клієнтською та серверною частинами проєкту зумовлений низкою технічних, архітектурних і практичних переваг, що повністю відповідають вимогам до розроблюваного вебзастосунку – інтерактивного блогу. REST API базується на протоколі HTTP, що робить його універсальним і легко сумісним з будь-якими клієнтськими додатками, включно з браузерами та сторонніми сервісами. Це особливо важливо в контексті побудови масштабованої та легко розширюваної системи, де важлива взаємодія з потенційними зовнішніми інтерфейсами.

REST чітко структурує комунікацію за допомогою HTTP-методів: GET, POST, PUT, DELETE тощо. Це відповідає природі операцій, які виконує користувач у блозі: перегляд публікацій, створення нових статей, оновлення вмісту або його видалення. Така відповідність дозволяє будувати логічно зрозумілий і передбачуваний інтерфейс обміну даними, що суттєво спрощує як реалізацію, так і майбутню підтримку системи.

Крім того, REST-підхід забезпечує розділення відповідальностей між клієнтом і сервером, що є основним принципом при створенні SPA (Single Page Application) на базі React. У цьому випадку клієнт повністю відповідає за відображення інтерфейсу, тоді як сервер лише надає структуровані дані у форматі JSON. Це дозволяє досягнути високої продуктивності, а також забезпечити незалежність у розвитку клієнтської й серверної частин.

REST API також має високу гнучкість при реалізації механізмів автентифікації та авторизації, зокрема через використання JWT(JSON Web Token) токенів [8], що дозволяє реалізувати безпечну систему контролю доступу з можливістю масштабування до кількох ролей користувачів.

На завершення, REST є де-факто стандартом у розробці вебзастосунків, що забезпечує легку інтеграцію з більшістю фреймворків, бібліотек і засобів документування, а також полегшує співпрацю в команді завдяки зрозумілим принципам побудови запитів і відповідей.

У контексті інтерактивного блогу це забезпечує:

– передбачуваність взаємодії між частинами системи;

– простоту в обслуговуванні;

– зменшення залежностей між клієнтом і сервером;

– полегшену документацію та тестування.

2.1.2 Методи виявлення і аналізу вимог

Для визначення функціональних і нефункціональних вимог до системи було проведено гіпотетичний аналіз цільової аудиторії – потенційних користувачів новинного вебресурсу. Це дозволило сформувати загальне уявлення про потреби, мотивації та очікування користувачів, а також виявити типові сценарії використання системи.

Основними критеріями, за якими сегментувалася аудиторія, стали такі фактори:

– автор який є одночасно і читачем;

– базовий рівень володіння комп’ютером та інтернетом;

– написання, перегляд і керування публікаціями новинного характеру.

Профіль середнього користувача включає: молодих людей, переважно студентів або молодих спеціалістів, зацікавлених у створенні власного інформаційного контенту та обміні думками. Їх головними очікуваннями є:

– зручний, адаптивний і швидкий інтерфейс;

– можливість швидко публікувати тексти з медіаконтентом;

– можливість переглядати і фільтрувати новини інших користувачів;

– доступ до комунікації через коментарі.

У контексті інтерактивного блогу, де кожен користувач має можливість створювати публікації, редагувати та видаляти власні матеріали, а також переглядати контент інших авторів, REST API виявляється максимально зручним і логічним рішенням. Він дозволяє чітко структурувати доступ до ресурсів, такі як, пости, користувачі, категорії, застосовуючи стандартні HTTP-методи відповідно до CRUD-операцій.

У такій системі автор виконує більшість дій, притаманних як звичайному користувачеві, так і контент-криєтору. REST API забезпечує інтуїтивну й логічну маршрутизацію: /posts для отримання списку публікацій, /posts/:id для доступу до конкретної публікації, /auth/login для входу тощо. Адміністратор, у свою чергу, через ті ж самі маршрути отримує доступ до розширених можливостей – наприклад, редагування або видалення публікацій інших авторів, модерації контенту, або керування правами доступу.

REST-архітектура дає змогу легко реалізувати перевірку прав доступу на рівні кожного запиту: наприклад, API може повертати статус 403 Forbidden, якщо автор намагається видалити не свій пост. Використання токенів у заголовках запитів дозволяє аутентифікувати користувача та авторизувати його на основі ролі. Це особливо корисно в умовах клієнт-серверної взаємодії, коли сервер не зберігає стан користувача.

Також REST API легко розширювати: додавання нових ендпоінтів або розширення функціональності, наприклад, вподобань, коментарів, категорій не потребує суттєвої перебудови системи – нові ресурси просто додаються як нові маршрути. Цей підхід сприяє масштабованості проєкту та спрощує інтеграцію з іншими сервісами.

На основі проведеного аналізу цільової аудиторії, а також ураховуючи характер системи, що розробляється, було сформульовано набір функціональних і нефункціональних вимог до інформаційно-комп’ютерної системи.

Функціональні вимоги описують конкретні дії, які система повинна виконувати. Основні з них включають:

– користувачі мають змогу реєструватися, входити до системи та виходити з неї. На основі токенів JWT визначається рівень доступу до функціоналу;

– користувач може створювати новини, редагувати власні пости та видаляти їх;

– під час створення або редагування публікацій автор має можливість прикріплювати зображення, які зберігаються у хмарному сховищі Cloudinary;

– всі користувачі, незалежно від автентифікації, мають змогу переглядати новини;

– адміністратори мають повний доступ до керування всіма публікаціями та користувачами;

– можливість позначити публікацію як улюблену – для персоналізації взаємодії з контентом.

Нефункціональні вимоги визначають якісні характеристики роботи системи:

– система повинна швидко реагувати на запити, забезпечуючи мінімальну затримку при завантаженні сторінок та виконанні CRUD-операцій;

– архітектура має дозволяти легке масштабування, зокрема шляхом винесення медіафайлів у хмарне сховище та розділення фронтенду і бекенду;

– передбачене використання HTTPS для передачі даних, а також JWT для захисту сесій користувача;

– підтримка роботи на всіх сучасних веббраузерах і пристроях, адаптивний дизайн для мобільних платформ;

– інтерфейс повинен бути інтуїтивно зрозумілим, простим у навігації, з мінімальною кількістю кроків до виконання основних дій;

– система повинна бути стійкою до помилок, а також мати базові механізми валідації та обробки виключень на клієнті та сервері;

– повинна бути виключена можливість пошкодження чи втрати інформації в процесі взаємодії з базою даних.

Чітке визначення функціональних і нефункціональних вимог на початковому етапі проєктування дозволяє побудувати систему, яка забезпечують потрібний набір можливостей і високий рівень якості програмного продукту.

2.1.3 Моделювання в UML

Unified Modeling Language (UML) є стандартним інструментом для візуального моделювання програмних систем, що дозволяє описати структуру, поведінку та взаємодію компонентів на різних рівнях абстракції. У межах цієї

кваліфікаційної роботи UML-діаграми використовуються для візуалізації вимог, архітектури та логіки функціонування новинного блогу.

Діаграма класів, яка зображена на рисунку 2.1 відображає структуру системи у вигляді класів, їх атрибутів, методів та зв’язків.

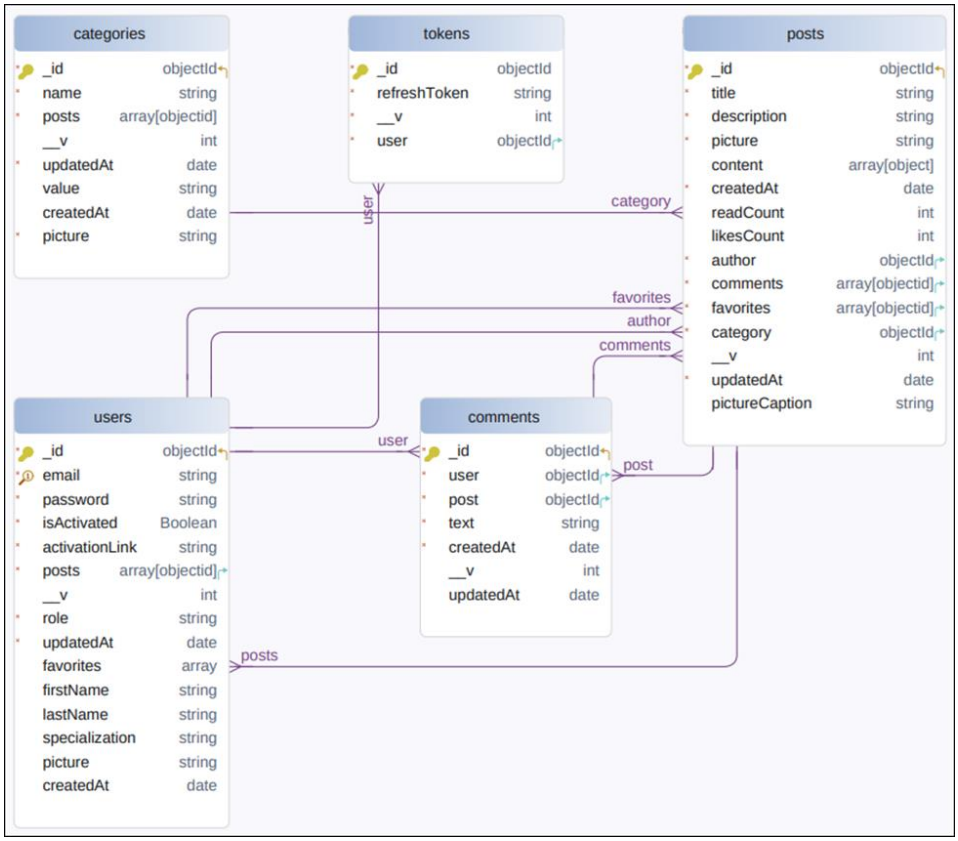

Рисунок 2.1 – Структура бази даних

Основні сутності:

– users, користувачі системи (автори/читачі), мають поля для автентифікації, профілю, улюблених постів тощо;

– posts, пости з полями для контенту, автора, категорій, коментарів, медіа тощо;

– comments, коментарі, які пов’язані з користувачами та постами;

– tokens, зберігають refresh-токени для реалізації авторизації через JWT;

– categories, категорії, до яких належать пости.

Типи зв’язків:

– один-до-багатьох, наприклад, один користувач → багато постів;

– багато-до-багатьох, наприклад, пости можуть бути в улюблених у багатьох користувачів і навпаки;

– один-до-одного: tokens.user, токен належить лише одному користувачеві.

Для візуалізації поведінки користувача та логіки взаємодії з системою під час роботи зі статтями було побудовано діаграму активностей, зображену на рисунку 2.2. Ця діаграма відображає послідовність дій, які виконує користувач при створенні, перегляді, редагуванні та видаленні публікацій. Вона охоплює як клієнтську частину (взаємодія з інтерфейсом), так і серверну логіку, включно з передачею запитів до API та обробкою даних на сервері. Структура діаграми дозволяє чітко простежити розгалуження в залежності від обраної дії, а також передбачає перевірку автентичності користувача, що є обов'язковим етапом доступу до функцій керування контентом.

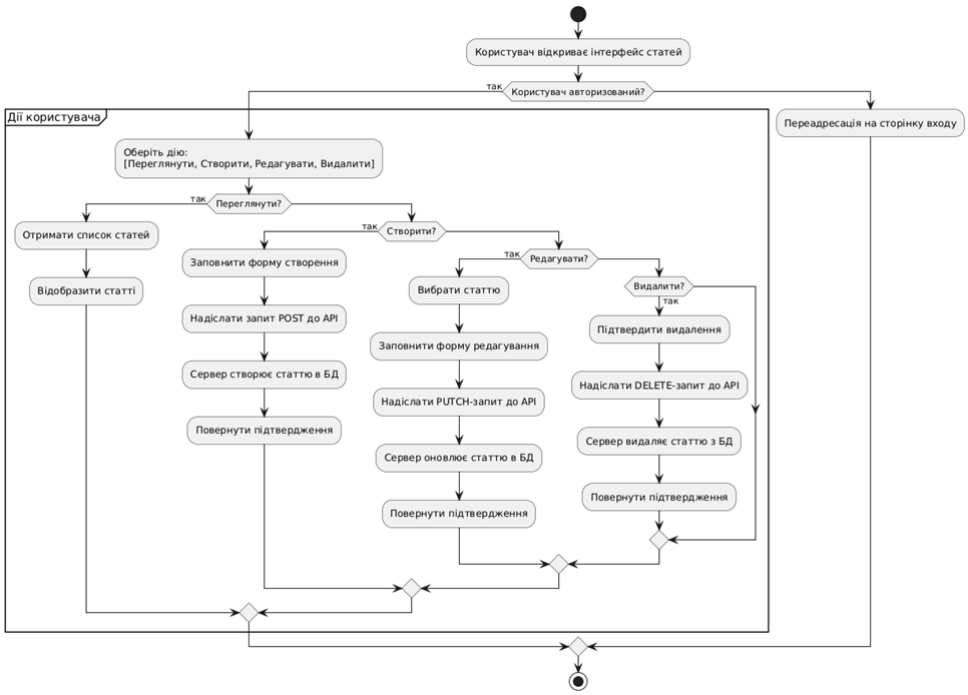

Рисунок 2.2 – Діаграма активностей

Для детального опису взаємодії між компонентами системи під час основних операцій зі статтями було побудовано діаграму послідовностей на рисунку 2.3.

Вона демонструє типову послідовність запитів і відповідей між користувачем, клієнтською частиною React, серверною логікою Express.js і базою даних MongoDB. Такий підхід дозволяє чітко простежити, як саме реалізується цикл запиту, які етапи обробки даних проходять на сервері, та як система реагує на дії користувача.

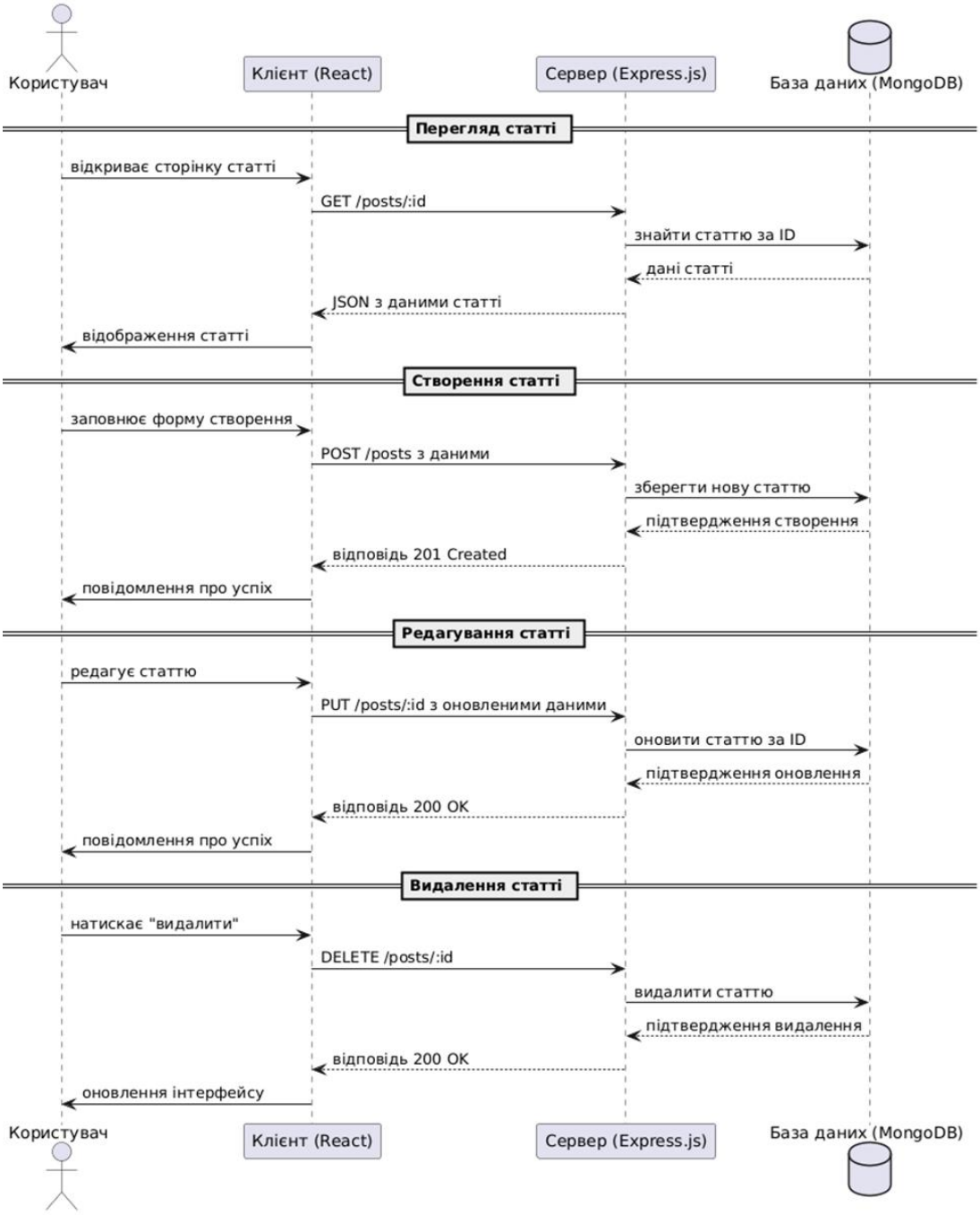

Рисунок 2.3 –Діаграма послідовностей

2.1.4 Специфікація функціональних можливостей системи

Функціональність автентифікації та авторизації є ключовою частиною системи, яка забезпечує безпечний вхід користувачів, а також контроль за доступом до окремих частин функціоналу залежно від ролі.

Процес починається з реєстрації користувача, під час якої вказуються email і пароль. Пароль перед збереженням у базу даних хешується за допомогою алгоритму bcrypt [9], що виключає можливість зберігання його у відкритому вигляді. Також при реєстрації до профілю користувача автоматично додається поле role зі значенням user, яке визначає роль за замовчуванням.

Після входу користувач отримує пару токенів: короткочасний accessToken, який використовується для автентифікації запитів, та refreshToken, що дозволяє оновити токени без повторного введення облікових даних. Токени реалізовано за допомогою стандарту JWT, що дозволяє вбудовувати в них інформацію про ідентифікатор користувача та його роль (role: user | admin). Ця інформація шифрується за допомогою секретного ключа та перевіряється сервером при кожному запиті до захищених маршрутів.

Авторизація реалізована на основі middleware, яке аналізує JWT-токен, що передається в заголовку Authorization, і надає або забороняє доступ до функціоналу залежно від ролі. Таким чином, користувачі з роллю 'admin' мають додаткові повноваження: перегляд, редагування та видалення всіх постів, а також управління вмістом системи. Звичайні користувачі з роллю user мають змогу створювати публікації, редагувати або видаляти лише власні матеріали.

Усі маршрути, що потребують автентифікації, є захищеними, і доступ до них можливий лише за умови наявності дійсного accessToken. Крім того, вся комунікація між клієнтом і сервером здійснюється через HTTPS, що гарантує шифрування даних і захист від атак типу «man-in-the-middle».

Функціональність створення, читання, оновлення та видалення новинних статей є основою інтерактивного блогу, що розробляється. Вона реалізована як на клієнтській, так і на серверній частині застосунку та забезпечує повний життєвий цикл обробки контенту.

Кожна новинна стаття складається зі структурованих полів: заголовка, короткого опису, основного вмісту редагованого через Tiptap-редактор, зображень, автора та дати створення. Для обробки запитів використовуються RESTful маршрути:

– POST /posts для створення нової публікації, цей маршрут доступний лише автентифікованим користувачам, після створення стаття автоматично зв’язується з автором через його id;

– GET /posts для отримання списку всіх публікацій із можливістю пагінації, фільтрації за категоріями чи автором;

– GET /posts/:id для перегляд повної інформації про окрему статтю за її ідентифікатором;

– PUT /posts/:id для оновлення змісту публікації, цй маршрут доступний лише автору публікації або адміністратору, перед оновленням система перевіряє, чи збігається id користувача в токені з id автора статті;

– DELETE /posts/:id для видалення публікації, правила доступу аналогічні до оновлення: автор або адміністратор.

На рівні бази даних для зберігання публікацій використовується колекція posts, у якій кожен документ має посилання на документ користувача. Це дозволяє ефективно отримувати як особисті пости користувача, так і пости інших авторів.

Завдяки такій CRUD-архітектурі користувачі можуть повноцінно взаємодіяти з контентом: вести блог, оновлювати новини, реагувати на поточні події. Адміністратори мають повний контроль над усім контентом, що забезпечує ефективну модерацію й контроль якості публікацій.

Функціональність завантаження медіафайлів є важливою складовою сучасного блогу, оскільки дозволяє авторам створювати візуально насичені пости з ілюстраціями, знімками екрана, інфографікою або іншими графічними матеріалами. Це підвищує привабливість і зручність сприйняття контенту користувачами.

У реалізації цієї функції було обрано хмарний сервіс Cloudinary, що спеціалізується [10] на зберіганні, обробці та оптимізації зображень. Завдяки інтеграції з Cloudinary користувачі можуть завантажувати зображення безпосередньо з браузера. На клієнтській частині (React) після обрання зображення виконується його попередній перегляд, а сам файл відправляється у вигляді FormData на бекенд.

На сервері медіафайл обробляється через middleware multer, після чого надсилається до Cloudinary через офіційний SDK. У відповідь від Cloudinary надходить унікальне посилання secure_url, яке зберігається в базі даних разом з іншими полями поста або вставляється вмістом у редактор через кастомний плагін.

Цей підхід має кілька суттєвих переваг:

– Cloudinary автоматично оптимізує зображення для різних пристроїв і забезпечує швидке завантаження за допомогою глобальної мережі доставки контенту;

– зображення не зберігаються локально на сервері Render, що вирішує проблему нестійкого файлового збереження в хмарних середовищах;

– Cloudinary дозволяє обробляти великі обсяги медіа без погіршення продуктивності основного сервера;

– можна вказати додаткові параметри зображення, такі як розмір, формат, якість без повторного завантаження, просто змінюючи URL.

Підтримка зображень реалізована як у формі створення постів, так і в інтерактивному редакторі вмісту. Це дає можливість додавати зображення не лише як окремі об'єкти, а й вбудовувати їх у текстові блоки поста.

Забезпечення безпеки даних і контроль доступу до ресурсів є критично важливими аспектами при розробці сучасного вебзастосунку. У реалізованій системі для захисту автентифікованих маршрутів використовується технологія JSON Web Token (JWT), яка є сучасним та надійним механізмом передачі ідентифікаційної інформації між клієнтом і сервером.

Принцип роботи JWT полягає в наступному: після успішної автентифікації введення правильних логіна і пароля, сервер створює токен, що містить закодовану інформацію про користувача, зокрема його id, email та role. Цей токен підписується секретним ключем і передається клієнту, який зберігає його локально, зазвичай у localStorage або cookie. Надалі клієнт додає цей токен до кожного запиту у заголовок Authorization, що дозволяє серверу ідентифікувати користувача без потреби повторної автентифікації.

На сервері реалізовано middleware, яке перевіряє валідність токена. Якщо токен дійсний і підпис збігається, користувач отримує доступ до запитуваного ресурсу. Якщо ж токен відсутній, протермінований або підроблений – сервер повертає помилку доступу 401 Unauthorized або 403 Forbidden, запобігаючи несанкціонованим діям.

JWT-токени мають обмежений строк дії, після чого користувачеві необхідно повторно пройти автентифікацію. Такий механізм забезпечує додатковий рівень захисту, зменшуючи ризик використання вкрадених токенів.

Переваги використання JWT у системі:

– немає потреби зберігати сесію на сервері;

– у токен можна вбудувати будь-які атрибути користувача, включно з роллю;

– підходить для розподілених систем і мікросервісної архітектури;

– підписані токени захищені від змін, а при використанні HTTPS передача є шифрованою.

Завдяки JWT було реалізовано надійну систему контролю доступу до функціоналу платформи, що є основою для забезпечення конфіденційності, цілісності та доступності даних.

Для забезпечення захищеного віддаленого доступу до системи у проєкті використано протокол HTTPS, розширення стандартного HTTP, яке підтримує шифрування з використанням TLS. Завдяки цьому всі дані, що передаються між клієнтом і сервером, зашифровані, що унеможливлює їх перехоплення або модифікацію зловмисниками.

Сайт розгорнуто на платформі Vercel клієнтську частину та Render серверну частину, які автоматично надають підтримку HTTPS за допомогою безкоштовних сертифікатів від Let's Encrypt. Це дозволяє користувачам безпечно взаємодіяти з системою, вводити особисті дані, здійснювати автентифікацію, публікувати пости, переглядати контент та виконувати інші дії без ризику втрати конфіденційної інформації.

Наявність HTTPS також позитивно впливає на індексацію ресурсу пошуковими системами, а сучасні браузери відображають попередження при спробі доступу до сайтів без захищеного з’єднання, що робить його критично необхідним компонентом для довіри та безпеки.

На сервері реалізовано додаткові заходи безпеки:

– обов’язкове перенаправлення всіх HTTP-запитів на HTTPS;

– використання заголовків безпеки, таких як Strict-Transport-Security для примусового використання HTTPS;

– заборона CORS-запитів з ненадійних джерел.

Завдяки HTTPS система гарантує автентичність сервера, захист від атак та збереження цілісності даних, що передаються мережею.

2.1.5 UX/UI-дизайн та інтерфейси

Розробка UX/UI-дизайну відіграє ключову роль у забезпеченні зручності користування, естетичної привабливості й загальної якості взаємодії користувача з системою. У межах даного проєкту особливу увагу було приділено побудові логічної, інтуїтивно зрозумілої структури інтерфейсів, адаптивному розташуванню елементів та чіткому візуальному поділу.

Загалом UX-дизайн був спрямований на створення таких сценаріїв взаємодії, які б мінімізували кількість дій для досягнення цілі, запобігали помилкам користувачів і надавали чіткий зворотний зв’язок. Інтерфейс системи проектувався згідно з принципами human-centered design, а також враховував сучасні практики з області когнітивної ергономіки, щоб уникати перевантаження інформацією.

UI-дизайн системи реалізовано з використанням бібліотеки компонентів Mantine, сучасного open-source фреймворку на базі React, що підтримує гнучку кастомізацію, темну тему, адаптивність і високу продуктивність.

Основні переваги вибору Mantine:

– підтримка готових доступних компонентів;

– повна підтримка стилізації через tailwindcss;

– чудова інтеграція з редактором Tiptap, який було використано для створення rich text-редактора публікацій;

– можливість реалізації кастомних UI-елементів з урахуванням потреб проєкту.

Типографіка обрана з урахуванням зручності сприйняття тексту на різних пристроях. Всі компоненти протестовано на відповідність адаптивності, дизайн коректно працює як на десктопах, так і на мобільних пристроях.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

У процесі розробки інтерактивного блогу було проаналізовано низку сучасних інструментів та технологій, які застосовуються у створенні вебзастосунків. Було обрано стек MERN (MongoDB, Express.js, React, Node.js) як один із найпоширеніших і найбільш підтримуваних у світовій практиці. Його гнучкість, розширюваність та активна спільнота роблять його оптимальним вибором для реалізації масштабованого та динамічного вебдодатку.

На клієнтській стороні використовується React – бібліотека JavaScript, яка забезпечує побудову інтерфейсу користувача з високим рівнем реактивності. React дозволяє ефективно керувати станом додатку, повторно використовувати компоненти, а також інтегрувати сторонні рішення для управління редактором тексту, завантаження зображень і обробки запитів до API.

Node.js було обрано як серверну платформу завдяки її здатності обробляти велику кількість одночасних запитів у неблокуючому режимі. Разом з Express.js

ця технологія дозволяє швидко реалізувати REST API, управляти сесіями користувачів, маршрутизацією, авторизацією та взаємодією з базою даних.

Перевага Node.js – використання єдиної мови JavaScript/TypeScript [11] як на клієнті, так і на сервері, що значно спрощує логіку взаємодії між фронтендом і бекендом, дозволяє повторно використовувати деякі утиліти або логіку валідації. Це робить його особливо привабливим для невеликих і середніх команд, які хочуть підтримувати єдину мовну екосистему у всьому стеку. У випадку блогу, де основне навантаження пов’язане з обробкою CRUD-операцій, авторизацією, фільтрацією, сортуванням і взаємодією з базою даних, Node.js демонструє високу продуктивність і легкість у розгортанні. Проте варто зазначити, що через свою асинхронну природу і відсутність усталеної структури, Node.js може створити труднощі для новачків або в командній розробці, якщо не встановлено чітких стандартів і підходів до архітектури.

MongoDB – це документно-орієнтована NoSQL база даних, яка зберігає дані у форматі BSON (розширення JSON). Основна перевага MongoDB полягає у її гнучкій схемі: на відміну від реляційних СКБД, тут немає потреби попередньо визначати структуру таблиці – документи можуть містити різні набори полів, що особливо зручно для динамічних або швидко змінюваних даних. У випадку блогової платформи це означає можливість зберігати пости з довільною структурою – наприклад, стаття з галереєю зображень, вбудованими відео, таблицями, цитатами тощо. MongoDB дозволяє інкапсулювати всі дані посту в одному документі, включаючи коментарі, теги, автора, мітки часу тощо, без складних зв’язків між таблицями. Це спрощує запити і підвищує продуктивність на читання. Вона також добре масштабується горизонтально, що корисно при збільшенні навантаження. Проте гнучкість MongoDB може стати недоліком у великих системах, де потрібна сувора нормалізація даних або жорсткі зв’язки між сутностями, наприклад, користувачами, ролями, правами доступу. Також відсутність транзакційного контролю на рівні складних міждокументних операцій може створювати ризики для цілісності даних у деяких сценаріях.

Для візуалізації логіки системи використовуються UML-діаграми: діаграми варіантів використання, діаграма класів, блок-схеми обробки даних. Основні алгоритми включають логіку, реалізацію механізму додавання в обране, контроль доступу до адміністративних функцій.

Усі ці засоби дозволяють забезпечити побудову сучасного, адаптивного, масштабованого й безпечного блогу, що відповідає як технічним, так і користувацьким вимогам, перевірки автентичності, обробку запитів CRUD до статей, реалізацію механізму додавання в обране, контроль доступу до адміністративних функцій.

Для реалізації функціональності інтерактивного блогу було розроблено алгоритми і логічні моделі, що забезпечують ефективну обробку користувацьких дій, взаємодію з базою даних і підтримку інтерактивного інтерфейсу. Алгоритми було реалізовано з урахуванням вимог до масштабованості, швидкодії та безпеки.

Однією з ключових моделей є модель користувача, яка описує основні атрибути та містить методи для автентифікації й авторизації. У моделі передбачено зберігання паролів у хешованому вигляді за допомогою алгоритму bcrypt, що підвищує рівень безпеки.

Алгоритм реєстрації користувача передбачає перевірку унікальності email, хешування пароля, створення токена доступу й запису користувача до бази даних. У випадку успішної реєстрації клієнт отримує токен, який використовується для подальших автентифікованих запитів.

Алгоритм автентифікації включає перевірку валідності введених даних, пошук користувача в базі, порівняння пароля з використанням bcrypt.compare та генерацію JWT-токена із заданим часом життя. Цей токен надсилається у відповідь на клієнтську сторону й зберігається у локальному сховищі та куках.

Для реалізації додавання поста до «вподованого» використовується наступний алгоритм: при натисканні кнопки «додати в вподованого» на клієнті надсилається PUT-запит на сервер, який перевіряє автентифікацію користувача, перевіряє наявність поста у списку обраного й додає або видаляє відповідний

ідентифікатор у масив favorites. Після цього оновлений список зберігається в базі даних.

Ще один важливий алгоритм – фільтрація та пошук постів. При запиті клієнта сервер виконує пошук у базі за текстовими індексами (наприклад, по полях title, content, tags) з використанням регулярних виразів або MongoDB $text-пошуку. Для забезпечення продуктивності результати обмежуються, сортуються та розбиваються на сторінки (пагінація), реалізована через skip() і limit().

Модель поста включає інформацію про автора, заголовок, текст, дату створення, список коментарів, категорію, позначку «обране», а також посилання на зображення. Для зручності серіалізації на клієнт передаються DTO-об’єкти, які приховують зайві поля. наприклад, хеш пароля чи службову інформацію MongoDB.

Окрім моделей, використовуються також middleware-алгоритми, які реалізують перевірку токенів, контроль доступу до маршрутів, обробку помилок і логування подій. Вони дозволяють ізолювати функціональну логіку від перевірочних дій, зберігаючи чисту архітектуру додатку.

На клієнтській стороні застосовуються алгоритми керування станом через react hooks та бібліотеки Redux Toolkit.

Таким чином, реалізовані алгоритми й моделі створюють надійну основу для функціонування блогу, забезпечуючи інтерактивність, динамічність, стабільність і безпеку системи.

У ході моделювання та проєктування системи інтерактивного блогу були створені структурні та поведінкові схеми, що визначають архітектуру вебзастосунку, взаємодію між компонентами, а також потоки даних і сценарії використання. Це дозволило формалізувати вимоги до системи, уникнути логічних помилок на етапі реалізації та забезпечити масштабованість і розширюваність проєкту.

У процесі роботи були створені UML-діаграми, які візуалізують основні аспекти архітектури:

– діаграма варіантів використання відображає основних акторів і їхню взаємодію із системою, зокрема, описано сценарії реєстрації, входу в систему, створення постів, додавання до обраного;

– діаграма класів демонструє структуру об'єктів: моделі користувача, поста, коментаря, категорії, ролі, у діаграмі відображено зв’язки між сутностями, типи атрибутів;

– діаграми активності та блок-схеми ілюструють логіку обробки даних, наприклад: реєстрація користувача, додавання поста до обраного, перевірка прав доступу, оновлення профілю, обробка запитів на фільтрацію.

Проєктування frontend-частини виконувалося відповідно до архітектурного підходу Feature-Sliced Design, який передбачає поділ застосунку на шари shared, entities, features, widgets, pages, app відповідно до функціональних ролей. Це забезпечує ізольованість компонентів, повторне використання логіки, полегшує масштабування та супровід коду.

Для backend було спроєктовано багаторівневу архітектуру із розділенням відповідальностей на контролери, сервіси, моделі, middleware та DTO-об’єкти. Такий підхід дозволив реалізувати чітку логіку маршрутизації, автентифікації, валідації, обробки помилок і взаємодії з базою даних MongoDB.

Результатом проєктування стало створення системи, яка:

– підтримує багаторазове використання компонентів і сервісів;

– має модульну структуру, зручно масштабовану;

– забезпечує безпечну обробку користувацьких даних;

– дозволяє легко додавати нові функції без порушення існуючої логіки;

– відповідає принципам чистої архітектури та SOLID-підходу.

Таким чином, результати моделювання й проєктування стали основою для успішної реалізації програмного продукту, який має чітку логічну структуру, продуману архітектуру й відповідає технічним вимогам до сучасних вебзастосунків.

РОЗДІЛ 3
РОЗРОБКА ІНТЕРАКТИВНОГО БЛОГУ З ВИКОРИСТАННЯМ REACT, NODE.JS ТА MONGODB

3.1 Практична реалізація об’єкта проєктування

3.1.1 Підготовка середовища розробки

На першому етапі розробки було створено зручне та стабільне середовище, яке дозволяє ефективно організувати процес написання, тестування та деплоювання коду. Для цього було здійснено встановлення базових інструментів:

– Node.js як рушій для виконання JavaScript на сервері;

– MongoDB документоорієнтована база даних, встановлена локально для тестування та з доступом до хмарної версії;

– менеджер пакетів npm для встановлення залежностей проєкту;

– Vercel хостинг для фронтенд-частини на React;

– Render хостинг для серверної частини на Node.js, з підтримкою MongoDB та змінних середовища.

Після встановлення інструментів було ініціалізовано репозиторій на GitHub, що дозволило зручно контролювати версії коду та співпрацювати [12]. Для автоматизації процесу розгортання було налаштовано CI/CD-процес із використанням Git, що забезпечило миттєве оновлення застосунку після комітів у основну гілку.

3.1.2 Реалізація серверної частини

Серверна частина блогу була реалізована з використанням Node.js та фреймворку Express.js, що забезпечують високу продуктивність, масштабованість і зручну побудову REST API. Проєкт має модульну структуру, що дозволяє легко розширювати функціональність і підтримувати чистоту коду.

Було створено основні директорії:

– controllers – містить логіку обробки запитів;

– models – описує схеми MongoDB за допомогою бібліотеки Mongoose;

– routes – визначає API-маршрути;

– middlewares – включає проміжне ПЗ для авторизації, обробки помилок, перевіром доступу;

– services – інкапсулює бізнес-логіку, зокрема обробку користувачів, постів, коментарів.

Після налаштування структури було реалізовано підключення до бази даних MongoDB із використанням Mongoose.

У файлі index.ts реалізовано ініціалізацію серверу та з'єднання з базою приклад на лістингу 3.1.

Лістинг 3.1 – Корінний файл ініціалізації сервера

```
const PORT =  process.env.PORT_SERVER || 5000;
const app: Application = express();
app.use(rateLimit({windowMs: 15 * 60 * 1000, limit: 100, standardHeaders:
'draft-8', legacyHeaders: false,}));
app.use(express.json());
app.use(mongoSanitize());
app.use(cookieParser());
app.use(cors({
      credentials: true,
      methods: ['GET', 'POST', 'DELETE', 'PUT', 'PATCH'],
      origin: process.env.ALLOWED_ORIGINS?.split(',') || [],
  }));
app.use('/static', (req, res, next) => {
    res.header('Access-Control-Allow-Origin', '*');
    res.header('Access-Control-Allow-Headers', 'Origin, X-Requested-With,
Content-Type, Accept');
    next();
});
app.use('/static', express.static(path.join(__dirname, 'static')));
app.use(fileUpload());
app.use('/api', authRouter);
app.use('/api', postsRouter);
app.use('/api', categoryRouter);
app.use('/api', usersRouter);
app.use(errorMiddleware);

const start = async () => {
    try {
        if (!process.env.DB_URL) {
            throw new Error('DB_URL is not defined in the environment
variables');
        }
```

```
        await mongoose.connect(process.env.DB_URL);
        app.listen(PORT, () => {
            console.log(`Server started on PORT = ${PORT}`)
        });
    } catch (e) {
        console.error(e);
    }
};
start();
```

кінець лістингу 3.1

Створено основні моделі:

– user, модель, яка містить інформацію про користувача: ім’я, email, пароль (хешований через bcrypt), аватар тощо;

– post, модель, яка зберігає назву, вміст, зображення, теги, автора, дату створення та оновлення;

– comment, модель, яка включає текст коментаря, користувача, дату, пост, до якого його прикріплено;

– category, модель яка включає в себе назву самої категорії та пости всі пости, які створені з використанням цієї категорії.

Приклад моделі Post буде зображено на лістингу 3.2.

Лістинг 3.2 – Схема моделі Post

```
import { Schema, model, Document, Types } from 'mongoose';
export interface IPost extends Document {
    _id: Types.ObjectId;
    author: Types.ObjectId;
    title: string;
    description: string;
    content: object[];
    picture: string;
    pictureCaption: string;
    createdAt: string;
    updatedAt: string;
    isFavorite: boolean;
    favoritesCount: number;
    comments: Types.ObjectId[],
    favorites: Types.ObjectId[],
    category: Types.ObjectId;
}
const PostSchema = new Schema<IPost>({
```

```
    title: { type: String, required: true },
    description: { type: String, required: true },
    picture: { type: String, required: true },
    pictureCaption: { type: String },
    content: { type: [Object], required: true },
    author: { type: Schema.Types.ObjectId, ref: 'User', required: true },
    comments: [{ type: Schema.Types.ObjectId, ref: 'Comment' }],
    favorites: [{ type: Schema.Types.ObjectId, ref: 'User' }],
    category: { type: Schema.Types.ObjectId, ref: 'Category' },
}, {
    timestamps: true,
    versionKey: false
});

PostSchema.virtual('isFavorite').get(function (this: IPost, userId:
string) {
    return this.favorites.some(fav => fav.toString() === userId);
});
PostSchema.virtual('favoritesCount').get(function (this: IPost) {
    return this.favorites.length;
});
PostSchema.set('toObject', { virtuals: true });
PostSchema.set('toJSON', { virtuals: true });

export default model<IPost>('Post', PostSchema);
```

кінець лістингу 3.2

Модель Post є центральним елементом системи, що описує публікації у блозі. Вона поєднує текстовий та мультимедійний контент, інформацію про автора, категорію, коментарі, обрані пости, а також реалізує віртуальні властивості для підрахунку вподобань.

Призначення моделі:

збереження та виведення інформації про дописи блогу;

– підтримка мультимедійного формату статей із редактором на базі Tiptap;

– взаємодія з користувачами (обране, коментарі);

– категоризація контенту;

– підтримка віртуальних властивостей без дублювання даних у базі.

Основні поля:

– title: string – заголовок публікації;

– description: string – короткий опис, що відображається у списку публікацій;

– picture: string – обкладинка, завантажена через Cloudinary;

– pictureCaption: string – підпис до зображення;

– content: object[] – вміст статті у вигляді масиву блоків редактора;

– author: ObjectId – посилання на користувача, що створив публікацію;

– comments: ObjectId[] – список коментарів до публікації;

– favorites: ObjectId[] – список користувачів, які додали публікацію у вибране;

– category: ObjectId – категорія, до якої належить публікація;

– createdAt, updatedAt: string – автоматично додані поля дати створення та оновлення документа.

Віртуальні поля:

– isFavorite: boolean – визначає, чи є пост у вибраному для конкретного користувача;

– favoritesCount: number – загальна кількість користувачів, які додали пост у вибране.

Використання Tiptap. Контент зберігається у форматі JSON-блоків, що забезпечує гнучкість візуального представлення статей на фронтенді.

Віртуальні поля дозволяють уникати збереження похідних значень у БД, зберігаючи при цьому можливість доступу до них при серіалізації об’єкта.

Масив коментарів реалізує зв’язок «один до багатьох» між публікацією та коментарями, що дозволяє ефективно відображати та модерацію дискусій.

Поле category забезпечує належність допису до певної категорії, що використовується для тематичної фільтрації контенту.

Модель User реалізує збереження інформації про користувачів системи, зокрема їх персональні дані, роль, активність, а також взаємодію з постами. Вона є базовим елементом системи автентифікації, авторизації та персоналізації.

Призначення моделі:

– забезпечення ідентифікації користувачів через електронну пошту та пароль;

– підтримка ролей (звичайний користувач або адміністратор);

– зв’язок користувача з його постами;

– збереження постів, доданих у вибране;

– підтримка активації облікового запису через email.

Призначення моделі:

– забезпечення ідентифікації користувачів через електронну пошту та пароль;

– email: string – електронна пошта, унікальний ідентифікатор користувача;

– password: string – хешований пароль користувача;

– role: 'user' | 'admin' – роль користувача в системі, що визначає права доступу;

– firstName, lastName: string – ім’я та прізвище користувача;

– picture: string – URL аватарки користувача, завантаженої з Cloudinary;

– specialization: string – поле, що зберігає спеціалізацію користувача;

– isActivated: boolean – ознака того, чи було підтверджено електронну пошту;

– activationLink?: string – унікальне посилання для активації облікового запису;

– posts: ObjectId[] – масив постів, створених користувачем;

– favorites: ObjectId[] – масив постів, які користувач додав у вибране;

– createdAt, updatedAt: string – автоматично додані поля з часом створення й оновлення користувача.

Реалізація модулі User буде зображена на лістингу 3.3.

Лістинг 3.3 – Схема моделі User

```
import { Schema, model, Document, Types } from 'mongoose';
import { IPost } from "./post-model";

export interface IUser extends Document {
    _id: Schema.Types.ObjectId;
```

```
    email: string;
    role: 'user' | 'admin';
    firstName: string;
    lastName: string;
    picture: string;
    specialization: string;
    password: string;
    isActivated: boolean;
    activationLink?: string;
    createdAt: string;
    updatedAt: string;
    posts: Types.Array<IPost>;
    favorites: Types.Array<Schema.Types.ObjectId>;
}

const UserSchema = new Schema<IUser>({
    email: { type: String, unique: true, required: true },
    password: { type: String, required: true },
    role: { type: String, enum: ['user', 'admin'], default: 'user' },
    firstName: { type: String },
    lastName: { type: String },
    picture: { type: String },
    specialization: { type: String },
    isActivated: { type: Boolean, default: false },
    activationLink: { type: String },
    posts: [{ type: Schema.Types.ObjectId, ref: 'Post' }],
    favorites: [{ type: Schema.Types.ObjectId, ref: 'Post' }]
}, {
    timestamps: true,
    versionKey: false
});

export default model<IUser>('User', UserSchema);
```

кінець лістингу 3.3

Система ролей: реалізовано два рівні прав доступу – user і admin. Це дозволяє адміністратору модерувати контент, а звичайному користувачеві – створювати пости та взаємодіяти з ними.

Вибране: за допомогою поля favorites реалізовано логіку збереження улюблених постів, що дозволяє користувачу персоналізувати свій досвід.

Зв’язок з постами: двосторонній зв’язок із моделлю Post через поле posts на стороні користувача і author на стороні посту.

Активація користувача: поле isActivated забезпечує перевірку електронної пошти під час реєстрації, а activationLink – технічну реалізацію цього процесу.

Масштабованість: структура моделі дозволяє легко розширювати її додатковими полями.

Модель Category відповідає за класифікацію постів у системі. Вона дозволяє об’єднувати пости за тематиками, полегшуючи їх фільтрацію, навігацію та аналітику. Відповідно до обраної архітектури, категорії є окремими документами в базі даних MongoDB, пов’язаними з постами через зовнішній ключ.

Призначення моделі:

– створення структурованої ієрархії або списку категорій;

– забезпечення зв’язку між постами та відповідною категорією;

– підтримка візуального представлення категорій за допомогою зображення;

– реалізація механізмів фільтрації постів за категоріями.

Основні поля:

– name: string – назва категорії, яку бачить користувач, наприклад, «Наука», «Технології», «Мистецтво»;

– value: string – технічне текстове значення, що може використовуватись як ключ при фільтрації, наприклад, science, tech, art;

– posts: ObjectId[] – список постів, що належать до цієї категорії;

– picture: string – url-адреса ілюстрації для категорії (завантажується, наприклад, через Cloudinary);

– createdAt, updatedAt: string – дата створення та оновлення категорії додаються автоматично завдяки опції timestamps у схемі.

Реалізація моделі Category буде зображена на лістингу 3.4.

Лістинг 3.4 – Схема моделі Category

```
import { Schema, model, Document, Types } from 'mongoose';
import {IPost} from "./post-model";

export interface ICategory extends Document {
    _id: Types.ObjectId;
    name: string;
    value: string;
```

```
    posts: Types.Array<IPost>;
    picture: string
}

const CategorySchema = new Schema<ICategory>({
    name: { type: String, required: true,  unique: true },
    value: { type: String },
    posts: [{ type: Schema.Types.ObjectId, ref: 'Post' }],
    picture: { type: String  },
}, {
    timestamps: true,
    versionKey: false
});

export default model<ICategory>('Category', CategorySchema);
```

кінець лістингу 3.4

Унікальність назви, поле name має обмеження unique: true, що запобігає створенню категорій з однаковими назвами.

Зв’язок з постами: категорія може містити багато постів, що реалізується через масив posts. Це дозволяє швидко отримувати всі публікації в межах однієї тематики.

Використання зображення: завдяки полю picture кожна категорія може мати власну візуальну ідентичність, що особливо корисно в UI/UX-інтерфейсі при відображенні списку категорій.

Підтримка фільтрації: завдяки парі полів name і value, система може ефективно реалізувати фільтрацію на фронтенді та бекенді, при цьому зберігаючи розділення між відображуваним текстом і технічним ідентифікатором.

Модель Comment реалізує механізм коментування постів у системі. Вона дозволяє користувачам залишати текстові відгуки або доповнення до публікацій, що сприяє зворотному зв’язку, обговоренню та залученню до платформи.

Призначення моделі:

– зберігання коментарів, залишених користувачами до постів;

– встановлення зв’язку між коментарем, постом та автором коментаря;

– підтримка хронологічного порядку коментарів;

– розширення соціального функціонала блогу.

Основні поля:

– user: ObjectId – ідентифікатор користувача, який залишив коментар;

– post: ObjectId – ідентифікатор поста, до якого належить коментар;

– text: string – основний вміст коментаря (текст);

– createdAt, updatedAt: string – дата створення та оновлення коментаря.

Реалізація моделі Comment буде зображена на лістингу 3.5.

Лістинг 3.5 – Схема моделі Comment

```
import { Schema, model, Document, Types } from 'mongoose';

export interface IComment extends Document {
    _id: Types.ObjectId;
    user: Types.ObjectId;
    post: Types.ObjectId;
    text: string;
}

const CommentSchema = new Schema<IComment>({
    user: { type: Schema.Types.ObjectId, ref: 'User', required: true },
    post: { type: Schema.Types.ObjectId, ref: 'Post', required: true },
    text: { type: String, required: true },
}, {
    timestamps: true,
    versionKey: false
});

export default model<IComment>('Comment', CommentSchema);
```

кінець лістингу 3.5

Двосторонній зв’язок: коментарі зв’язані як з користувачем, так і з конкретним постом, що дозволяє легко фільтрувати та групувати дані, наприклад, отримувати всі коментарі користувача або всі коментарі до поста.

Автоматичне сортування, завдяки полю createdAt коментарі можуть відображатися у хронологічному порядку, звсюди можливістю реалізації пагінації або стрічки оновлень.

Обов’язковість полів, всі ключові поля автор, пост, текст є обов’язковими required: true, що гарантує цілісність записів у базі даних.

Модель Token, яка зображена на лістингу 3.6 реалізує механізм refresh-токенів, необхідних для збереження безперервної автентифікації користувача без потреби повторного входу після закінчення дії access токена.

Модель відповідає за зберігання refresh токена, який видається користувачу разом з access токеном при вході в систему. Коли access токен закінчує дію, refresh токен використовується для генерації нового access токена. Це підвищує безпеку системи, оскільки дозволяє обмежити час життя access токена без шкоди для UX.

Лістинг 3.6 – Схема моделі Token

```
import { Schema, model, Document, Types } from 'mongoose';

export interface IToken extends Document {
    user: Types.ObjectId;
    refreshToken: string;
}

const TokenSchema = new Schema<IToken>({
    user: { type: Schema.Types.ObjectId, ref: 'User', required: true },
    refreshToken: { type: String, required: true }
});

export default model<IToken>('Token', TokenSchema);
```

кінець лістингу 3.6

Поля:

– user: посилання на користувача, якому належить токен, посилання на модель User;

– refreshToken: сам токен, що зберігається в базі даних у зашифрованому вигляді.

Принцип роботи refresh-токена:

1) після входу користувача генерується пара access + refresh токенів;
2) access токен має короткий строк дії 2 дні, а refresh довший 30 днів;
3) refresh токен зберігається в базі у моделі Token;

4) коли access токен протерміновується, користувач автоматично надсилає refresh токен і отримує нову пару токенів.

При використанні refresh токенів важливо дотримуватися безпеки, а саме:

– шифрувати або зберігати з урахуванням безпеки;

– відкликати (видаляти з БД) при виході користувача;

– перегенеровувати при зміні паролю чи даних користувача.

Таким чином, моделі даних у цьому проєкті охоплюють усі основні об'єкти доменної області: користувачів, пости, коментарі, категорії, токени.

Вони побудовані з використанням ODM-бібліотеки Mongoose для MongoDB, що дозволяє:

– створювати гнучкі схеми зі зв'язками;

– інтегрувати валідацію;

– налаштовувати віртуальні поля;

– підтримувати референції між сутностями;

– використовувати timestamps і налаштовувати поведінку серіалізації.

API реалізовано за принципами REST, де кожен ендпоінт відповідає за конкретну дію над ресурсами. Основні маршрути згруповані за функціональністю: /api/auth, /api/posts, /api/users, /api/category.

Маршрути по шляху /api/auth, наведені в таблиці 3.1.

Таблиця 3.1 – Основні методи по маршруту /api/auth

Тип	Шлях	Опис
POST	/api/auth/register	Реєстрація користувача.
POST	/auth/login	Автентифікація користувача та видача токенів
POST	/auth/logout	Вихід з системи, видалення refresh токена
GET	/auth/authData	Отримання списку всіх нагород
POST	/auth/sendMail	Відправка листа для активації профілю
GET	/auth/activate/:activationLink	Активації профілю
POST	/auth/refresh	Оновлення access токена

Маршрути по шляху /api/users та /api/profile, наведені в таблиці 3.2.

Таблиця 3.2 – Основні методи для взаємодії з користувачами

Тип	Шлях	Опис
GET	/api/users	Отримання всіх користувачів
GET	/api/profile/:id	Отримання даних певного користувача
PATCH	/api/profile/:id	Оновлення особистої інформації
DELETE	/api/profile/:id	Видалення профілю;
PATCH	/api/user/change-password	Зміна пароля

Маршрути по шляху /api/posts наведені в таблиці 3.3.

Таблиця 3.3 – Основні методи для взаємодії з постами

Тип	Шлях.	Опис
GET	/api/posts	Отримання списку всіх постів
POST	/api/posts	Створення нового поста
GET	/api/posts/:id	Отримання одного поста
PUTCH	/api/posts/:id	Оновлення поста
DELETE	/api/user/change-password	Видалення поста
GET	/api/posts/:postId/comments	Отримання всіх коментарів до поста
GET	/api/posts/:postId/comments/:commentId	Отримання даних певного коментаря для певного поста
POST	/api//posts/:postId/comments	Створення коментаря
PUT	/api/posts/:postId/comments/:commentId	Оновлення коментаря
DELETE	/api/posts/comments/delete	Видалення коментаря для певного поста
GET	/api/posts/favorites/allByUser	Отримати всі улюблені пости користувача
POST	/api/posts/favorites	Додати пост в улюблені.

Маршрути по шляху /api/category наведені в таблиці 3.4.

Таблиця 3.4 – Основні методи для взаємодії з користувачами

Тип	Шлях	Опис
GET	/api/category	Отримання списку всіх категорій
GET	/api/category/:id	Отримання даних певної категорії;
POST	/api/category	Створення нової категорі
PATCH	/api/category	Оновлення категорії
DELETE	/api/category/:id	Видалення категорії.

Для захисту API застосовано JWT. Після логіну користувач отримує accessToken, який зберігається в клієнта у localStotage і refreshToken, який зберігається в cookie і базі даних.

Middleware authMiddleware, який зображений на лістингу 3.7 перевіряє валідність accessToken.

Лістинг 3.7 – Middleware authMiddleware

```
import ApiError from '../exceptions/api-error';
import tokenService from '../service/token-service';
import {Response, Request, NextFunction} from "express";

export default function authMiddleware(req: Request, res: Response, next:
NextFunction): void {
    try {
        const authorizationHeader = req.headers.authorization;
        if (!authorizationHeader) {
            return next(ApiError.UnauthorizedError());
        }
        const accessToken = authorizationHeader.split(' ')[1];
        if (!accessToken) {
            return next(ApiError.UnauthorizedError());
        }
  const userData = tokenService.validateAccessToken(accessToken);
        if (!userData) {
            return next(ApiError.UnauthorizedError());
        }
        if (typeof userData !== 'string') {
            //@ts-ignore
            req.user = userData;
```

```
        }
        next();
    } catch (e) {
        return next(ApiError.UnauthorizedError());
    }
}
```

кінець лістингу 3.7

Для зберігання зображень використано хмарне сховище Cloudinary. Користувачі можуть додавати зображення до профілю чи постів. Для цього файли з клієнта надсилаються на бекенд через multipart/form-data, обробляються за допомогою file-service, який представлений в додатку Б та бібліотеки multer, а потім завантажуються у Cloudinary, приклад створення конфігу Cloudinary буде зображено на лістингу 3.8.

Лістинг 3.8 – Створення конфігу Cloudinary

```
import { v2 as cloudinary } from 'cloudinary';
cloudinary.config({
    cloud_name: process.env.CLOUDINARY_CLOUD_NAME!,
    api_key: process.env.CLOUDINARY_API_KEY!,

    api_secret: process.env.CLOUDINARY_API_SECRET!,
});
export default cloudinary;
```

кінець лістингу 3.8

3.1.3 Реалізація клієнської частини

Клієнську частину застосунку реалізовано на основі бібліотеки React із використанням Feature-Sliced Design – архітектурного підходу, що дозволяє масштабувати фронтенд-проєкти завдяки чіткому розділенню відповідальності. В основі структури – поділ на логічні шари:

– pages/ – сторінки з маршрутизацією, які складаються з готових віджетів і фіч;

– widgets/ – незалежні блоки інтерфейсу, що поєднують декілька фіч;

– features/ – окремі функціональні можливості, наприклад, changePassword, deletePost, які мають власні компоненти, логіку, store;

– entities/ – сутності предметної області, наприклад, user, post, comment, які містять UI-компоненти, API та моделі;

– shared/ – спільні компоненти, стилі, типи, утиліти, константи тощо.

Такий підхід полегшує повторне використання коду, його тестування й підтримку, особливо при розширенні застосунку.

Для навігації між сторінками використано react-router-dom v6. Основна конфігурація маршрутизатора знаходиться у src/app/providers/router/ui/AppRouter/AppRouter.tsx Усі маршрути розділено за публічними та захищеними. Компонент RequireAuth, який зображено на лістингу 3.9 виконує перевірку авторизації на основі access токена й перенаправляє на сторінку входу в разі його відсутності.

Лістинг 3.9 – Логіка компонент RequireAuth

```
const useAuthNavigation = (middleware: Middleware[], authData: any,
location: any) => {
     if (!authData) {
          return null;
     }
     if (middleware.includes(Middleware.AUTH)) {
          if (!authData) { return <Navigate replace state={ { from:
location } } to={ getLoginRoutePath() } />; }
          if (!middleware.includes(Middleware.NO_VERIFY) && authData &&
!authData.isActivated) {          return <Navigate replace state={ {
from: location } } to={ getVerifyRoutePath() } />; }
     }
     if (middleware.includes(Middleware.AUTH) &&
middleware.includes(Middleware.ADMIN)) {
          if (authData.role !== "admin") { return <Navigate replace to={
getMainRoutePath() } />; }
     }

     if ((middleware.includes(Middleware.NO_AUTH) && authData) ||
(middleware.includes(Middleware.NO_VERIFY) && authData?.isActivated)) {
          return <Navigate replace state={ { from: location } } to={
getMainRoutePath() } />;
     }
     return null;
};
export const RequireAuth = ({ children, middleware }: { children:
ReactNode, middleware: Middleware[] }) => {
     const authData = useSelector(authSelectors.getUserAuthData);
     const location = useLocation();
```

```
     const navigation = useAuthNavigation(middleware, authData,
location);
     if (navigation) {
          return navigation;
     }
     return children;
};
```

кінець лістингу 3.9

Головна сторінка, ця сторінка показує перелік доступних публікацій. Компоненти RecentPosts і AllPosts отримує список з бекенду, сортує та відображає у вигляді карток з назвою, датою, автором і коротким описом.

Загалом розмітка головної сторінки буде зображена на лістингу 3.10.

Лістинг 3.10 – Розмітка головної сторінки

```
const MainPage = () => {
     const dispatch = useAppDispatch();
     const recentPosts = useSelector(mainPageSelectors.getRecentPosts);
     const allPosts = useSelector(mainPageSelectors.getAllPosts);
     const isLoading = useSelector(mainPageSelectors.getIsLoading);
     const error = useSelector(mainPageSelectors.getError);

     useEffect(() => {
          dispatch(mainPageActions.getPosts());
          dispatch(mainPageActions.getRecentPosts());
     }, [dispatch]);

     if (isLoading) {
          return (
               <Page className={ cls.PostsByUserPage }>
                    <Container>
                         <PageLoader />
                    </Container>
               </Page>
          );
     }
     if (error) {
          return (
               <Page className={ cls.PostsByUserPage }>
                    <Container>    <p>{ error.message }</p> </Container>
               </Page>
          );
     }
     return (
```

```
          <Page className={ cls.MainPage }>
               <Container>
                    <div className={ cls.MainPage }>
                         { recentPosts && <RecentPosts recentPosts={
recentPosts } /> }
                         { allPosts && <AllPosts allPosts={ allPosts } />
}
                    </div>
               </Container>
          </Page>
     );
};
export default MainPage;
```

кінець лістингу 3.10

Компонент PostPage відображає повну статтю з заголовком, контентом у rich text форматі, зображенням і коментарями. Для відображення rich text використовуються можливості редактора Tiptap, підключеного в режимі лише читання.

Сторінки LoginPage та RegisterPage реалізовані для логіну й реєстрації. Дані надсилаються на /api/auth/login або /api/auth/register, після чого токени зберігаються у cookie та localStorage. Після входу користувач перенаправляється на головну сторінку.

Сторінка профілю дозволяє користувачеві переглядати й редагувати власну інформацію, а також керувати власними публікаціями. Доступ до цієї сторінки мають лише авторизовані користувачі, і вона реалізована як окрема сторінка в pages/profile.

Компонент ProfilePage використовує API-запит для отримання поточної інформації про користувача.

Основний функціонал сторінки:

– відображення особистих даних ім’я, email, дата реєстрації, аватар;

– редагування профілю – кнопка відкриває модальне вікно або окрему форму, яка дозволяє змінити ім’я, email або завантажити нове зображення для аватару.

Інтерфейс реалізовано за допомогою Tailwind CSS v3, що забезпечує швидку розробку адаптивного дизайну. Кожен компонент адаптується під мобільні, планшетні й десктопні пристрої. Tailwind дозволяє писати чистий, декларативний код стилів без CSS-файлів, прикладом може бути кнопка зображена на лістингу 3.11.

Лістинг 3.11 – Приклад використання tailwindcss

```
<MaybeLink
     className="text-center border-2 border-solid border-black rounded-
full py-1 px-2.5 font-medium mt-5 mx-auto w-fit"
     url={ getPostsRoutePath() }
>
     Всі пости
</MaybeLink>
```

кінець лістингу 3.11

Форма створення/редагування статті з Tiptap. Для створення та редагування статей реалізовано розширену форму з полями: назва, опис, зображення, rich text контент.

Основу контенту формує Tiptap Editor, який дозволяє:

– вставляти зображення;

– форматувати текст, як заголовки, списки, цитати;

– вставляти відео, за допомогою YouTube-плагіну.

Конфігурація редактора зберігається у shared/ui/Editor/ui/Editor.tsx і включає кастомні екстеншени для кращої взаємодії з бекендом.

Для взаємодії з API використовується бібліотека Axios. У shared/api/api.ts реалізовано налаштування інстансу, який автоматично додає accessToken до заголовків запитів та слідкує чи accessToken залишається валідним, якщо токен не актуальний, то відправляється автоматично запит до API, що дозволяє підтримувати сесію користувача, лістинг.

Редактор статей реалізований як окремий компонент Editor, що інтегрує rich text функціональність на основі бібліотеки Tiptap – розширюваного редактора, створеного поверх ProseMirror. Цей редактор забезпечує зручне

форматування контенту з можливістю інтерактивної вставки зображень, відео та інших елементів.

Основні функціональні можливості:

– форматування тексту жирний, курсив, заголовки, цитати, списки, посилання;

– вставка зображень;

– вставка відео з YouTube, яка реалізована спеціальна кнопка, яка приймає посилання й вбудовує відео у вигляді iframe;

– візуальне форматування панель інструментів з іконка;

– підтримка undo/redo;

Tiptap конфігурується через розширення, що додають підтримку:

– StarterKit базові можливості;

– Image вставка зображень;

– YouTube підтримка YouTube.

Після редагування контент у форматі JSON зберігається в полі content моделі Post, що дозволяє точно відтворювати структуру при відображенні.

3.1.4 Хостинг і деплой

Розгортання вебзастосунку здійснювалося на популярних платформах хмарного хостингу Vercel для клієнтської частини та Render для серверної. Це дозволило забезпечити безперебійну роботу застосунку, автоматизацію CI/CD, а також зручну інтеграцію з системою керування версіями Git.

Vercel – це платформа для хостингу React-додатків, яка забезпечує автоматичний деплой при кожному коміті у головну гілку репозиторію [13]. Проєкт було ініціалізовано з GitHub-репозиторію, після чого Vercel автоматично визначив React як фреймворк, провів білд і опублікував застосунок за наданим URL.

Особливості налаштування:

– вибір папки збірки – dist або build, залежно від конфігурації Vite;

– встановлення змінних середовища, наприклад, API_URL;

– можливість підключення власного домену або використання піддомену Vercel.

Для бекенд-серверу на Node.js з Express використано платформу Render, яка підтримує автоматичне збирання, деплой і масштабування [14]. Сервер розгорнуто з GitHub-репозиторію за допомогою Web Service Render.

Налаштування включало:

– встановлення змінних середовища: MONGO_URI, JWT_SECRET, CLOUDINARY_API_KEY тощо;

– вказання стартової команди, наприклад, npm run builld;

– ввімкнення автоматичного деплою з гілки main;

– підтримка SSL/HTTPS на рівні Render.

Для забезпечення безпеки додатку було реалізовано:

– CORS (Cross-Origin Resource Sharing) дозволено запити з домену фронтенду за допомогою middleware cors, конфігурованого через origin;

– HTTPS, Render і Vercel автоматично забезпечують безпечне з’єднання;

– заголовки безпеки, використано middleware helmet, який додає заголовки Content-Security-Policy, X-Content-Type-Options, X-Frame-Options тощо.

3.2 Тестування та налагодження інформаційно-комп’ютерної системи

У процесі розробки системи особливу увагу приділено перевірці коректності її функціонування та виявленню можливих помилок. Тестування здійснювалося як вручну, так і з використанням автоматизованих підходів. Основними цілями були виявлення помилок логіки, перевірка стабільності API, а також забезпечення належного UX під час взаємодії користувача з клієнтською частиною.

У ході роботи застосовано такі методи тестування:

– використовувалося для перевірки окремих функцій backend-частини, зокрема валидації даних, логіки авторизації та роботи з базою даних, для цього використовувався Postman у поєднанні з mock-даними;

– інтеграційне тестування перевіряло взаємодію між модулями системи, наприклад, створення поста з автентифікацією користувача та подальшою перевіркою відображення на фронтенді;

– проводилося вручну через Postman для кожного endpoint'a з позитивними й негативними кейсами;

– реалізоване на клієнтській частині для перевірки правильного відображення інтерфейсу в різних браузерах і на різних пристроях.

Для ефективного виявлення і виправлення помилок була впроваджена система логування та моніторингу. Основна увага приділялася обробці критичних помилок і помилок авторизації.

Реалізовано єдину систему обробки помилок за допомогою власного міддлава на Express-сервері, яка забезпечує уніфіковану відповідь на всі винятки. На клієнті такі помилки обробляються глобальним error handler'ом із повідомленням користувачу.

У процесі ручного та інтеграційного тестування було виявлено і усунуто ряд проблем:

– помилки при обробці зображень через Cloudinary;

– випадки з некоректними токенами, що вимагало реалізації механізму оновлення через refreshToken;

– баги в логіці авторизації та доступу до захищених маршрутів, які були усунуті шляхом покращення middleware авторизації та перевірки ролей.

Для повноцінної роботи застосунку необхідне середовище з такими мінімальними характеристиками.

Для серверу:

– Node.js версії 18 або новішої;

– MongoDB хмарна або локальна інстанція;

– інтернет-з’єднання для доступу до Cloudinary та зовнішніх API.

Для клієнту:

– сучасний браузер із підтримкою ES6+;

– розширення Web API, для fetch, localStorage, cookies;

– пристрій з екраном роздільної здатності від 320px.

Завдяки гнучкій архітектурі застосунок легко масштабувати, деплоїти на різні середовища та адаптувати під різні пристрої користувачів.

3.3 Розробка бази даних

У рамках реалізації серверної частини інформаційної системи було обрано базу даних MongoDB як гнучке документно-орієнтоване сховище, яке дозволяє ефективно працювати з напівструктурованими даними та масштабувати систему відповідно до потреб. З'єднання з базою даних здійснюється за допомогою офіційної бібліотеки mongoose, яка надає обгортку над MongoDB API, спрощуючи операції зі зберігання, пошуку та валідації даних.

Для підключення до бази даних необхідно попередньо створити .env файл з відповідним рядком підключення, зображено на лістингу 3.11.

Лістинг 3.11 – Змінна для підключення до бази даних

```
DB_URL=mongodb+srv://<user>:<password>@<cluster>/<dbname>?retryWrites=true&w=majority
```

кінець лістингу 3.11

У програмному коді серверної частини конфігурація підключення реалізована в точці входу index.ts, як зображено на лістингу 3.12.

Лістинг 3.12 – Підключення до бази даних

```
import mongoose from 'mongoose';
import dotenv from 'dotenv';
dotenv.config();
```

```
const start = async () => {
    try {
        if (!process.env.DB_URL) {
            throw new Error('DB_URL is not defined in the environment
variables');
        }

        await mongoose.connect(process.env.DB_URL);

        app.listen(PORT, () => {
            console.log(`Server started on PORT = ${PORT}`)
        });

    } catch (e) {
        console.error(e);
    }
};

start();
```

кінець лістингу 3.12

Під час ініціалізації додатка здійснюється спроба встановити з’єднання з базою даних, і в разі успіху виводиться повідомлення про підключення. Усі моделі даних, користувачі, пости, коментарі, улюблені тощо, описуються за допомогою схем Mongoose у відповідній директорії models/, що відповідає архітектурі проєкту.

У разі виникнення помилки підключення система безпечно завершує ініціалізацію або інформує про проблему без виведення критичних відомостей, що відповідає вимогам безпеки.

Базу даних розміщено на хмарному сервісі MongoDB Atlas, який забезпечує високу доступність, резервне копіювання, масштабованість і сертифіковану безпеку на основі стандартів індустрії. Крім того, підключення до бази здійснюється лише з дозволених IP-адрес, що додає додатковий рівень захисту.

Таким чином, модуль підключення до бази даних є критичним компонентом серверної частини, що забезпечує надійний доступ до централізованого сховища даних, підтримує відповідність вимогам безпеки та зручності масштабування.

3.4 Захист інформаційно-комп’ютерної системи

Інформаційно-комп’ютерна система розроблялася з урахуванням актуальних викликів у сфері кібербезпеки, тому ключовими аспектами проєктування стали: надійна автентифікація та авторизація, захист від атак, безпечний хостинг та обмеження, що запобігають навмисному або випадковому компрометуванню даних.

Для автентифікації користувачів використано JSON Web Token сучасний відкритий стандарт RFC 7519, що дозволяє безпечно передавати дані між сторонами у вигляді об’єкта JSON. При успішному вході користувач отримує access- та refresh-токени, які зберігаються у браузері access у localStorage,, refresh у httpOnly cookie.

Користувацькі паролі зберігаються у базі у вигляді хешу, згенерованого за допомогою алгоритму bcrypt, який забезпчує надійне хешування з додаванням «солі».

Для контролю доступу реалізовано middleware-функції authMiddleware та adminMiddleware, які перевіряють валідність токенів і роль користувача user або admin.

На сервері використано захист від типових атак:

– Cross-Site Request Forger для протидії використовується токенізація запитів або реалізація строгої політики SameSite для cookies, наприклад Set-Cookie: refreshToken=...; HttpOnly; Secure; SameSite=Strict;

– NoSQL усі вхідні запити до бази перевіряються через валідаційну бібліотеку express-validator і ніколи не використовуються напряму у запитах без перевірки типу та структури [15];

– Ddos-атаки, на сервері встанлено middleware для обмеження кількості запитів з одного IP, за допомогою бібліотеки express-rate-limit [16], що дозволяє уникнути перевантаження, приклад middleware буде зображений на лістингу 3.13.

Лістинг 3.13 – Middleware для обмеження кількості запитів з одного IP

```
app.use(
     rateLimit(
     {
         windowMs: 15 * 60 * 1000,
         limit: 100,
         standardHeaders: 'draft-8',
         legacyHeaders: false,

     }
));
```

кінець лістингу 3.13

Вся передача даних між клієнтом і сервером здійснюється через HTTPS, сертифікати автоматично налаштовуються хостингом Vercel і Render підтримують Let’s Encry. Крім цього, впроваджено обмеження на розмір вхідних запитів, щоб уникнути атак через великі тіла запитів. Це забезпечує надійне шифрування з використанням TLS, згідно з математичною моделлю.

3.5 Розробка документації для програмного продукту

Для забезпечення ефективного використання розробленого програмного продукту – вебзастосунку блог створено технічну документацію, яка включає системні вимоги, інструкції з інсталяції, розгортання на сервері, підключення зовнішніх сервісів і довідкову інформацію для користувачів.

Для локального розгортання серверної та клієнтської частини проєкту необхідні наступні системні ресурси:

– операційна система: Linux Ubuntu 20.04+, macOS або Windows 10+;

– оперптивна пам’ять від 2 ГБ;

– процесор: 2 ядра з підтримкою x86-64;

– Node.js версія 18 або новіша;

– MongoDB версія 6.0 або хмарний екземпляр Atlas;

– інтернет-з’єднання для завантаження залежностей, інтеграції з Cloudinary, SMTP.

Інсталяція і налаштування системи починається з клонування репозиторіїв:

– git clone https://github.com/username/blog-client.git Blog/client;

– git clone https://github.com/username/blog-server.git Blog/server.

Налаштування серверної частини Blog/server. Перейти до каталогу Blog/server та встановити залежності за допомогою npm install. npm install – це команда, яка використовується в Node.js-проєктах для встановлення залежностей, бібліотек або модулів, вказаних у файлі package.json.

Для локального запуску потрібно створити файл .env та скопіювати всміст .env.example та наповнити змінними, далі ми його скопіюємо для .env файлу для Render:

– PORT_SERVER=5000;

– DB_URL=mongodb+srv://user:pass@cluster.mongodb.net/db;

– JWT_ACCESS_SECRET=secret-key-access;

– JWT_REFRESH_SECRET=secret-key-refresh;

– SMTP_HOST=smtp.gmail.com;

– SMTP_PORT=587;

– SMTP_USER=example@gmail.com;

– SMTP_PASSWORD=password;

– API_URL=https://blog-server.example.com;

– CLIENT_URL=https://blog-client.example.com;

– NODE_ENV=production;

– ALLOWED_ORIGINS=https://blog-client.example.com;

– CLOUDINARY_CLOUD_NAME=your-cloud-name;

– CLOUDINARY_API_KEY=your-api-key;

– CLOUDINARY_API_SECRET=your-api-secret;

Для отримання DB_URL потрібно перейти до https://cloud.mongodb.com/ та ввійти в свій аккаунт або зареєструватися.

Потім потім потрібно створити проєкт натиснувши на кнопку New Project як зображено на рисунку 3.1.

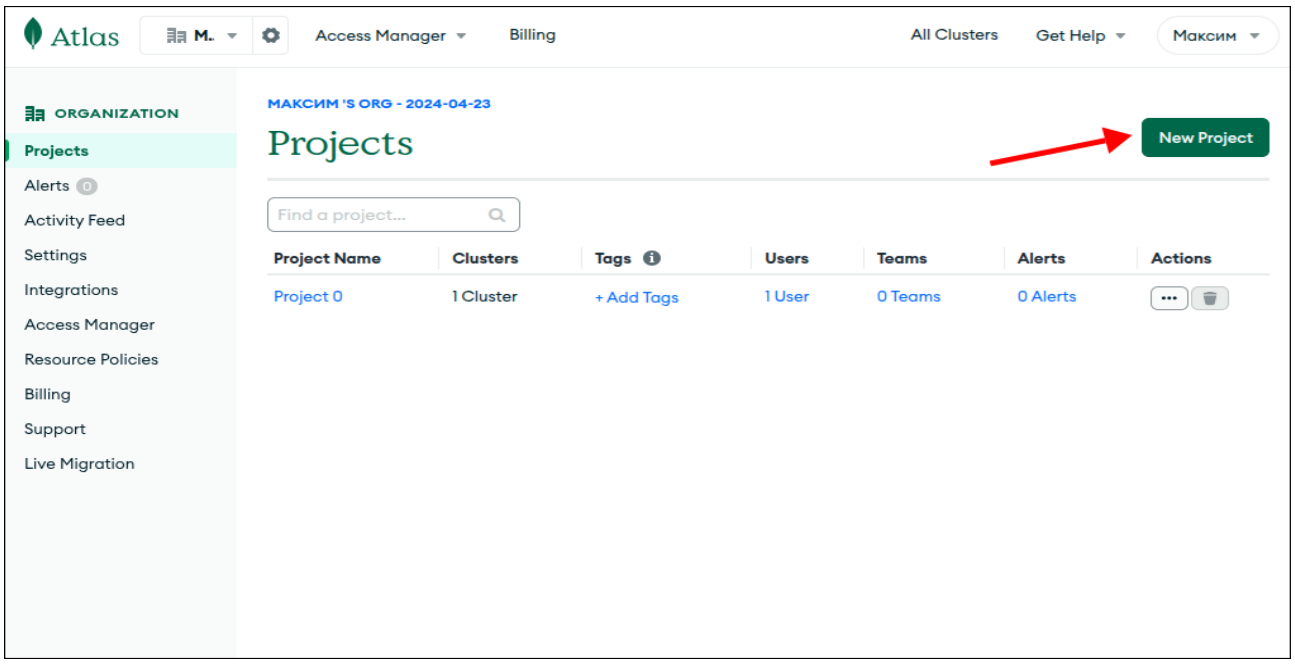

Рисунок 3.1 – Кнопка для створення проєкту

Заповнюєм назву проєкта, як зображено на рисунку 3.2.

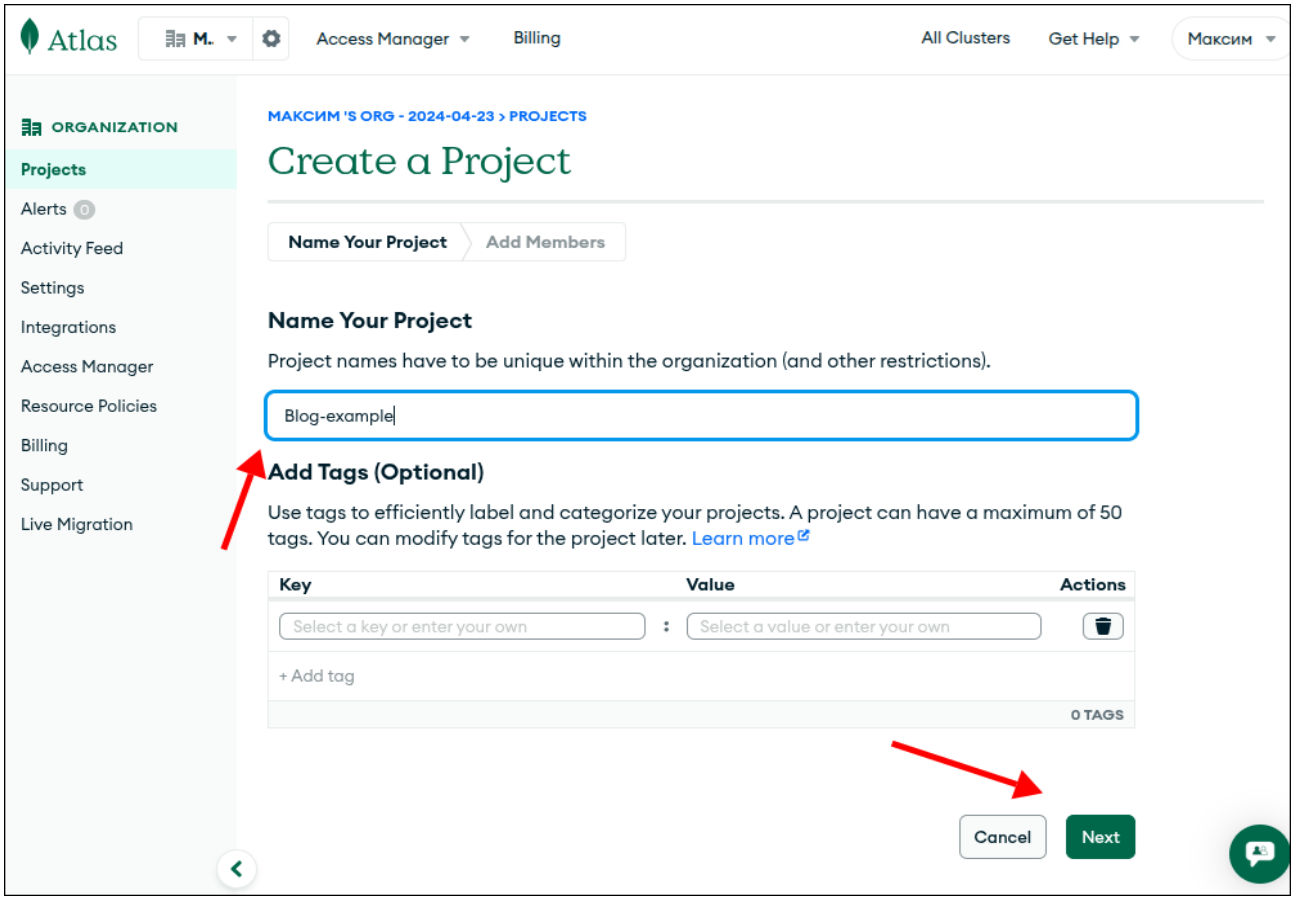

Рисунок 3.2 – Форма для вводу назви проєкта

Далі дотримуватися інструкції для створення проекту описаних далі в тексті. Додаєм спочатку себе як власником проєкту, як зображено на рисунку 3.3.

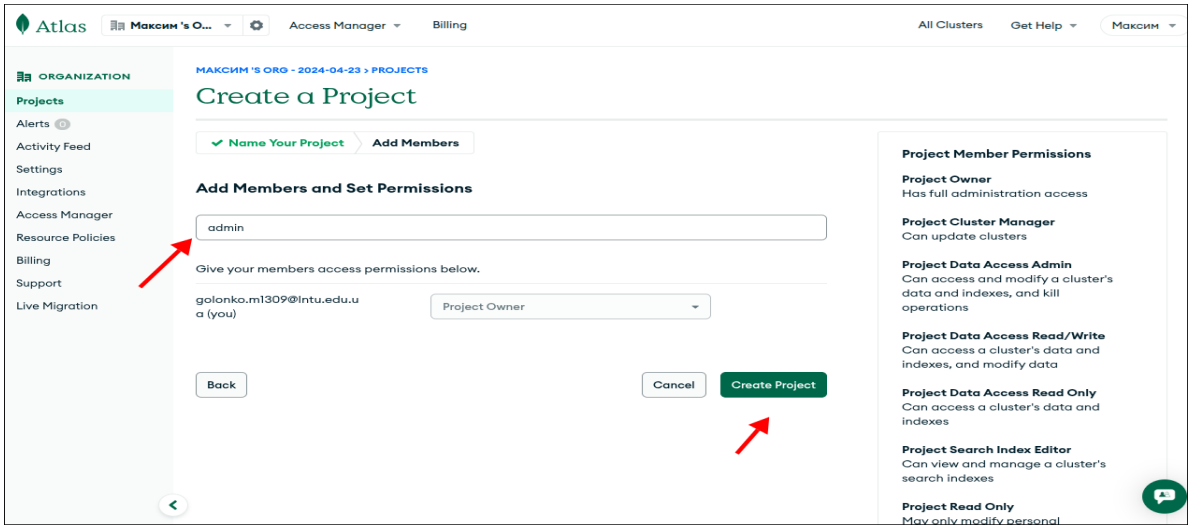

Рисунок 3.3 – Форма для вводу власника проєкту

Потім створюємо кластер вибираючи безкошковну версію або можете придбати екземпляр з більшими ресурсами та вводимо назву кластеру, деталі зображено на рисунку 3.4.

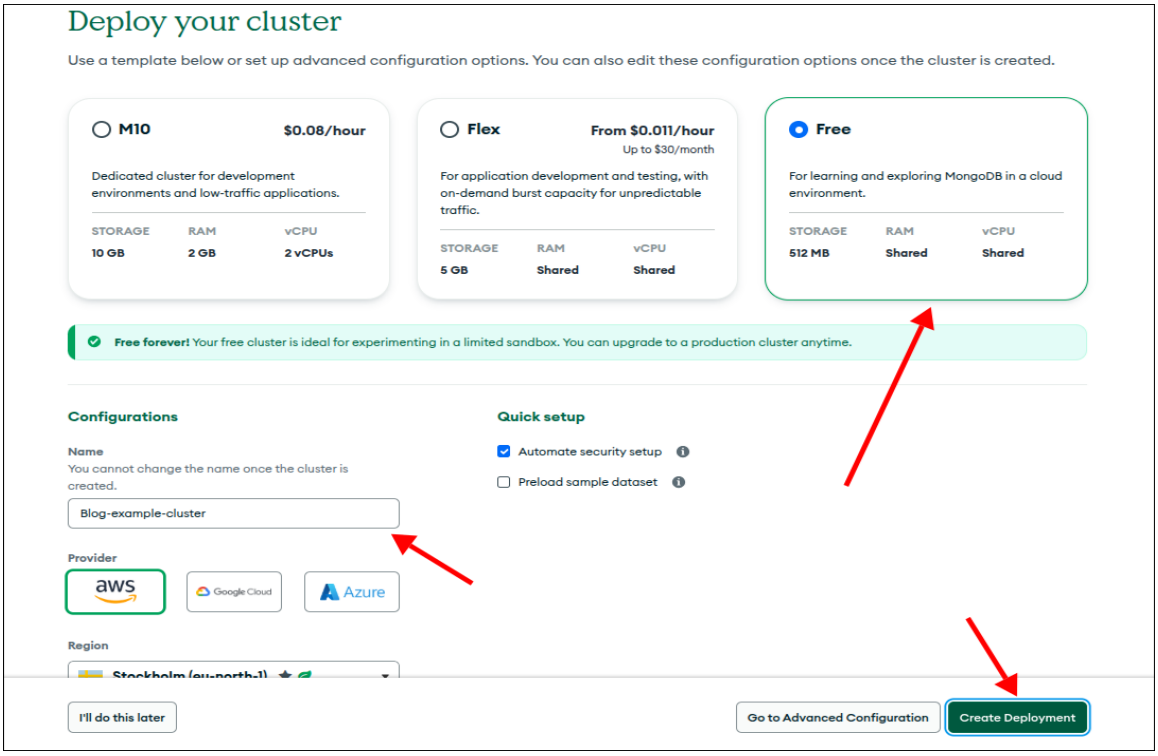

Рисунок 3.4 –Вибір кластеру на заповнення назви

Наступним кроком буде створення адміністратора для кластеру, так як він перший користувач, то він буде і адміном, вводимо пароль назву користувача та пароль, його бажано запам’ятати, тому, що він нам потрібен для підключенння до бази даних, створення користувача зображено на рисунку 3.5.

Далі вибираєм підключення до кластеру за допомогою Drivers, як зображено на рисунку 3.6.

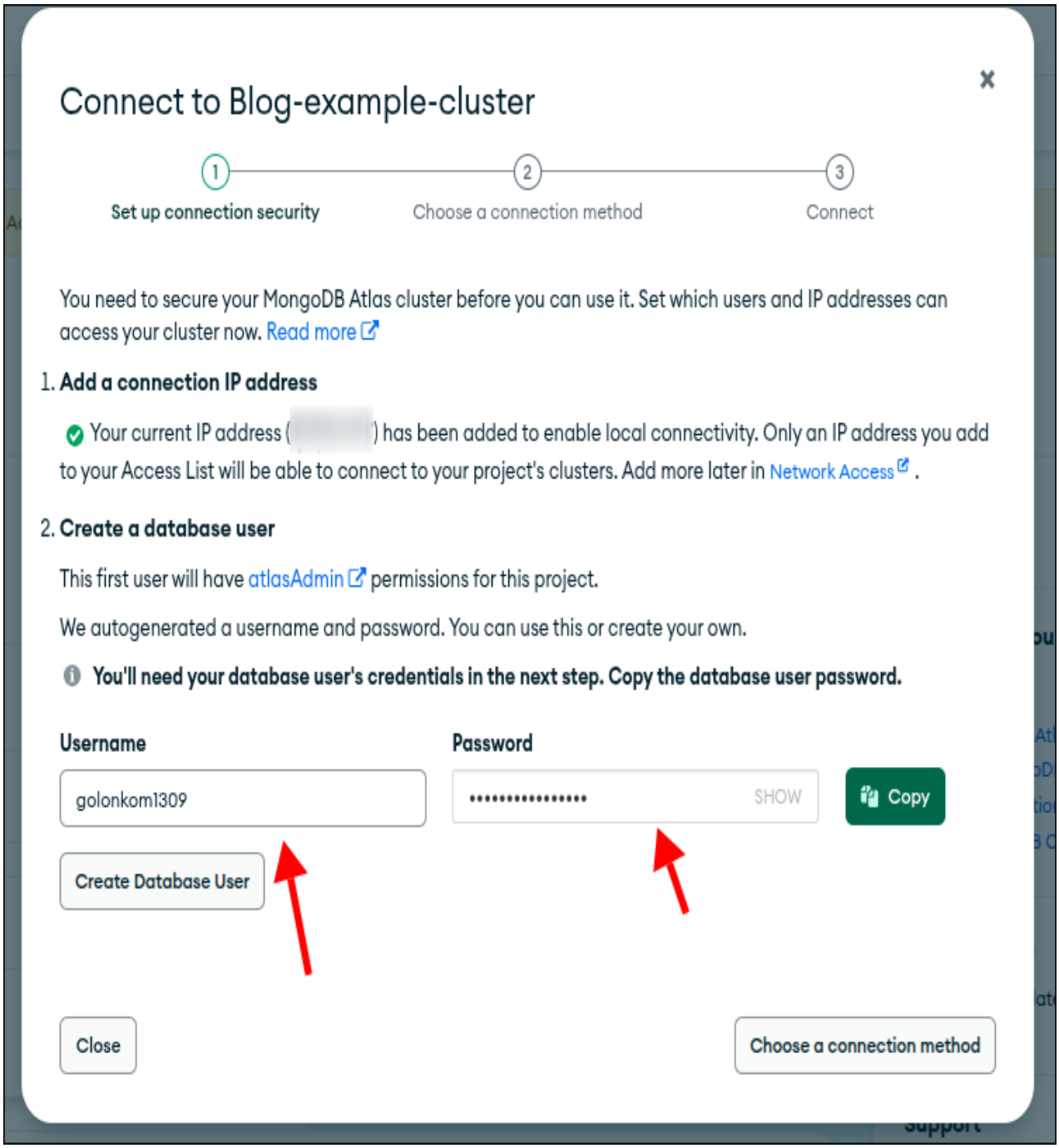

Рисунок 3.5 – Форма для створення адміна для кластеру

Далі вибираєм підключення до кластеру за допомогою Drivers, як зображено на рисунку 3.6.

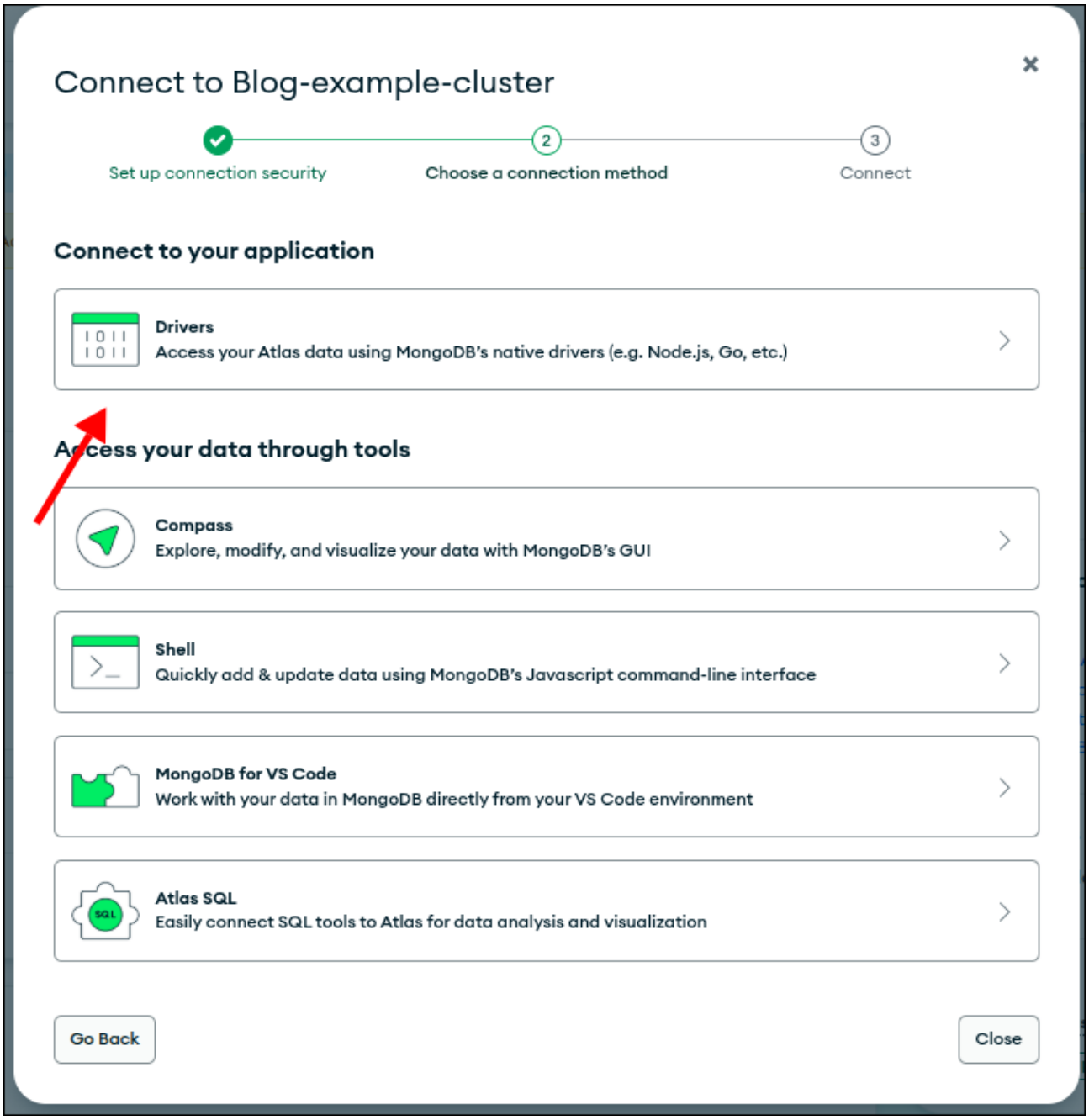

Рисунок 3.6 – Методи якими можна підключитися до кластеру

Копіюємо рядок з кодом для підключання до бази, для зручності можна показати пароль щоб відразу свопіювати рядок з кодом підключення або потім замінити на пароль свого адміністратора, як спопіювати рядок з кодом

підключення зображено на рисунку 3.7, потім цей рядок додаємо до нашого .env файлу, як значення DB_URL.

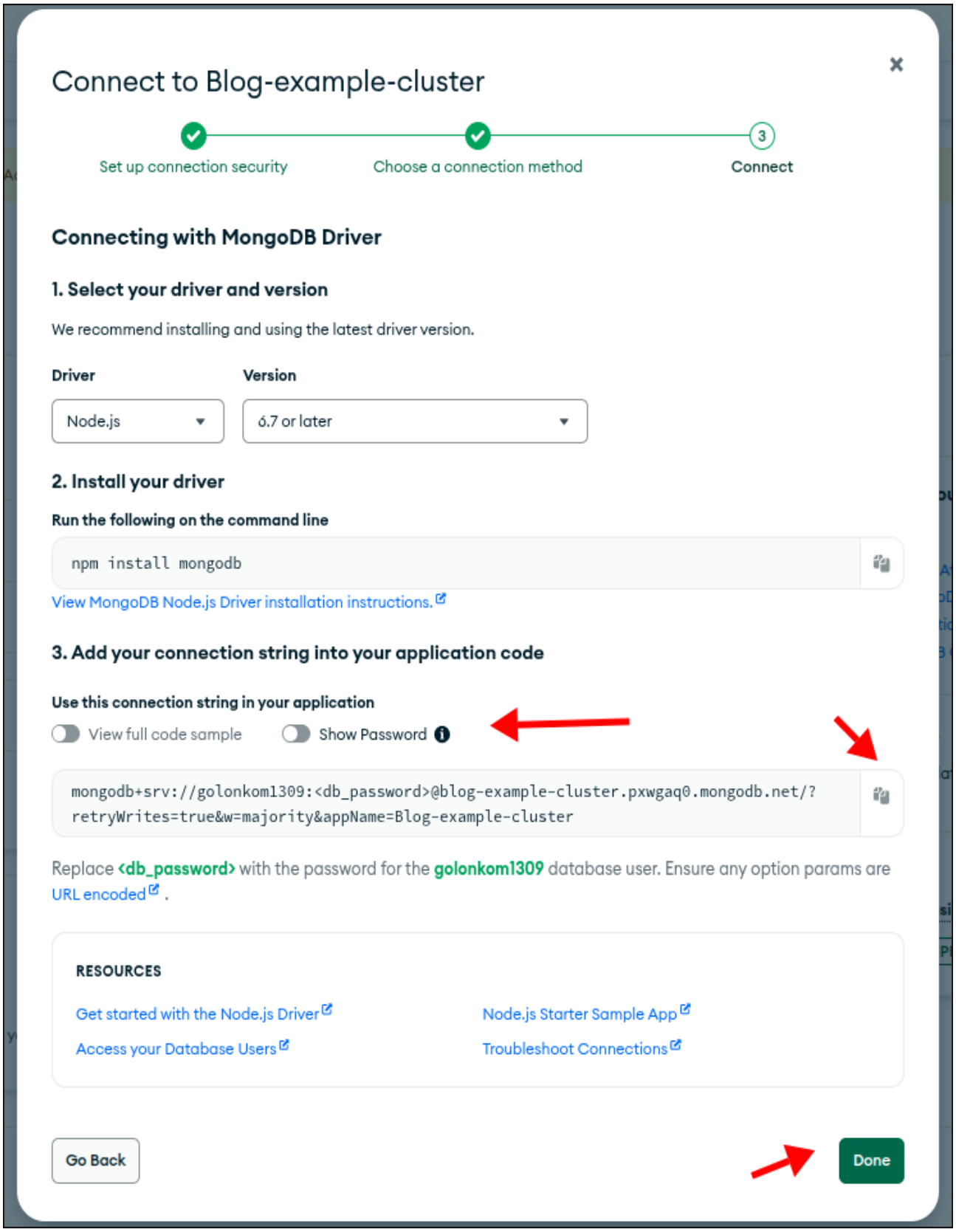

Рисунок 3.7 – Згенероний код для підключення до бази даних

Налаштування клієнської частини Blog/client. Перейти до каталогу Blog/client та встановити залежності за допомогою npm install. Для локального запуску потрібно виконати команду npm run start:vite:dev.

Потім додаєм наші каталоги client та server до GitHub.

Переходимо на render.com та авторизовуємося. Далі створюємо новий деплой і вибираємо тип web service, як зображено на рисунку 3.8 і додаємо наш репозиторійз сервером з GitHub.

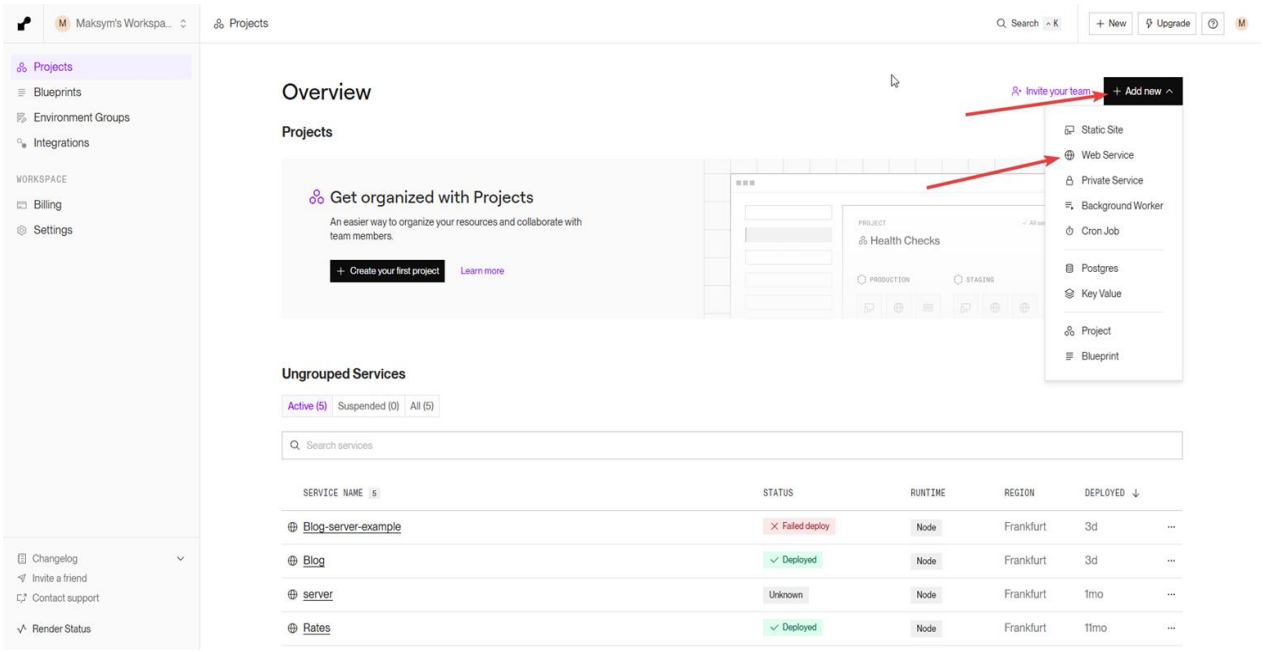

Рисунок 3.8 – Додаємо новий деплой

Після цього гортаєм нижче до Build Command який заповнюємо NODE_ENV=development npm install && npm run build та Start Command, який заповнюємо NODE_ENV=production && npm start та вибираємо безкоштовну версію екземпляру для деплоя, як зображено на рисунку 3.9. NODE_ENV=development встановлює змінну середовища NODE_ENV у значення development. Це важливо, бо деякі пакети або фреймворки поводяться по-різному залежно від режиму – наприклад, в development режимі вони включають додаткову діагностику, повідомлення про помилки, гаряче оновлення тощо.

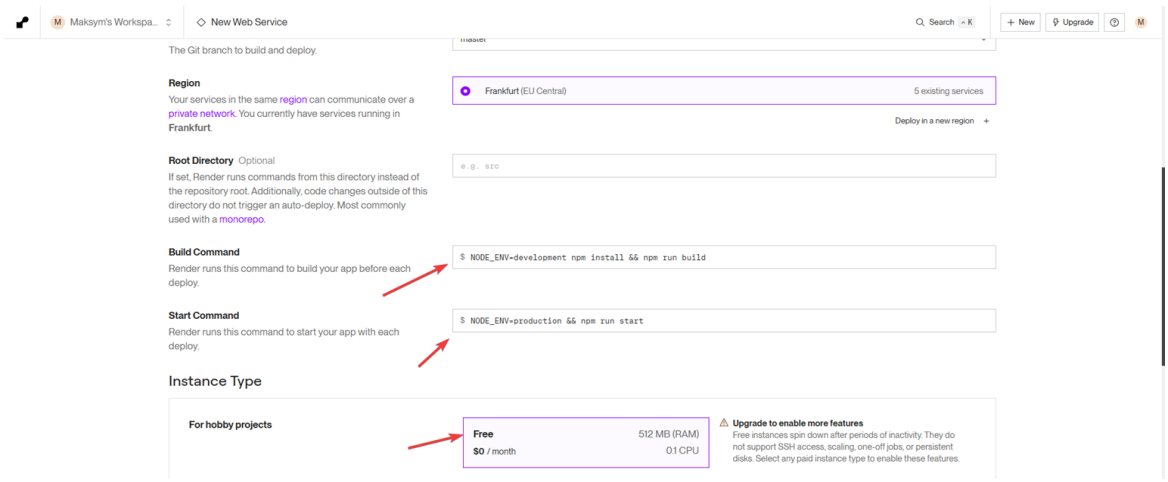

Рисунок 3.9 – Додавання параметрів та вибір деплоя

Заповнюєм .env файл, копіюючи туди вміст нашого локального .env з каталогу server, як зображено на рисунку 3.10.

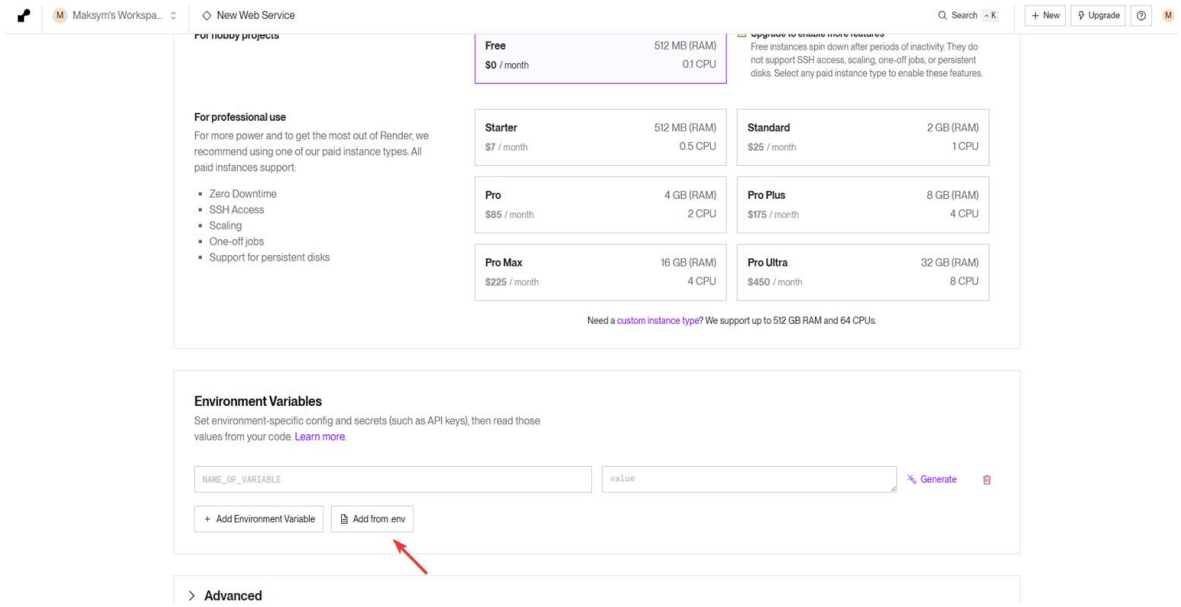

Рисунок 3.10 – Кнопка для додавання змінних до файлу .env

Тепер в нас є розгорнутий сервер, але потрібно ще розгорнути клієнтську частину на vercel для того, щоб додати до ALLOWED_ORIGINS адресу нашого клієнту, щоб в нього була змога взаємодіяти з сервером.

Тепер переходимо до vercel.com авторизовуємся та додаємо новий проект та імпортуєм з github та додаєм до .env змінну NODE_ENV_API значенням якої буде посилання на деплой нашого серверу та запускаєм деплой, після пару

хвилин у вас вже є розгорнутий сайт, приколад головної сторінки зображено на рисунку 3.11. На кінець копіюєм посилання до вашого деплою, переходим до деплою на render.com та вставляєм наше посилання до змігглої ALLOWED_ORIGINS.

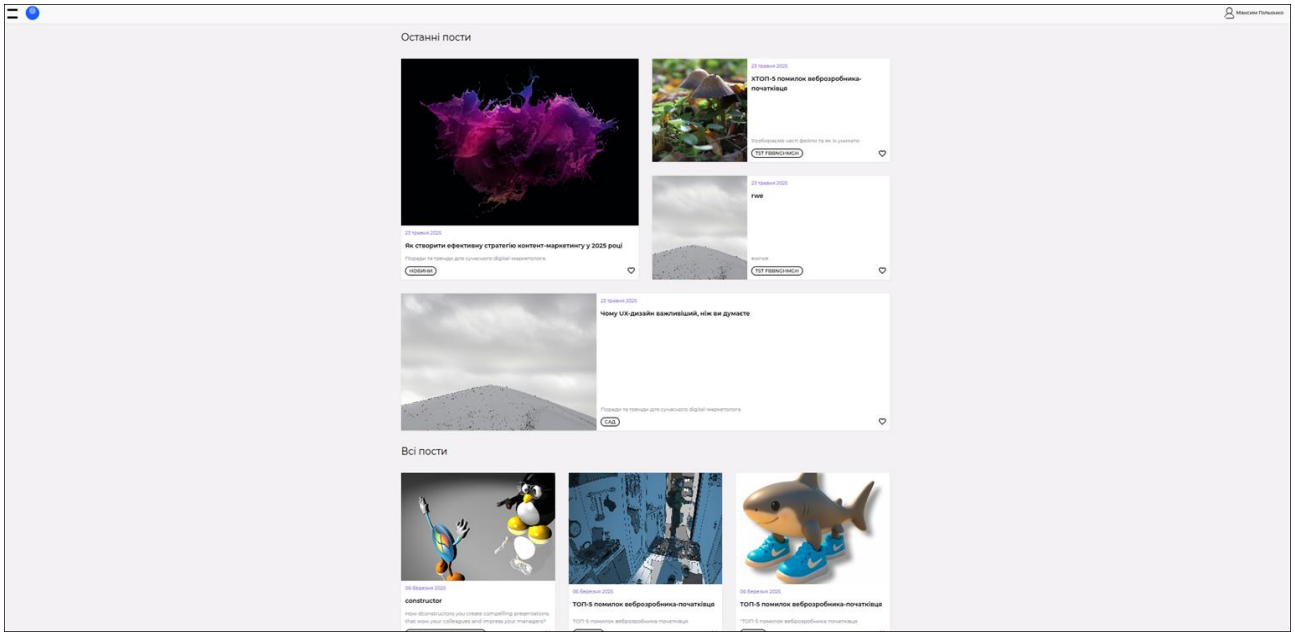

Рисунок 3.11 – Зображення головної сторінки

Також показано на рисунку 3.12, як виглядає детальна сторінка перегляду поста.

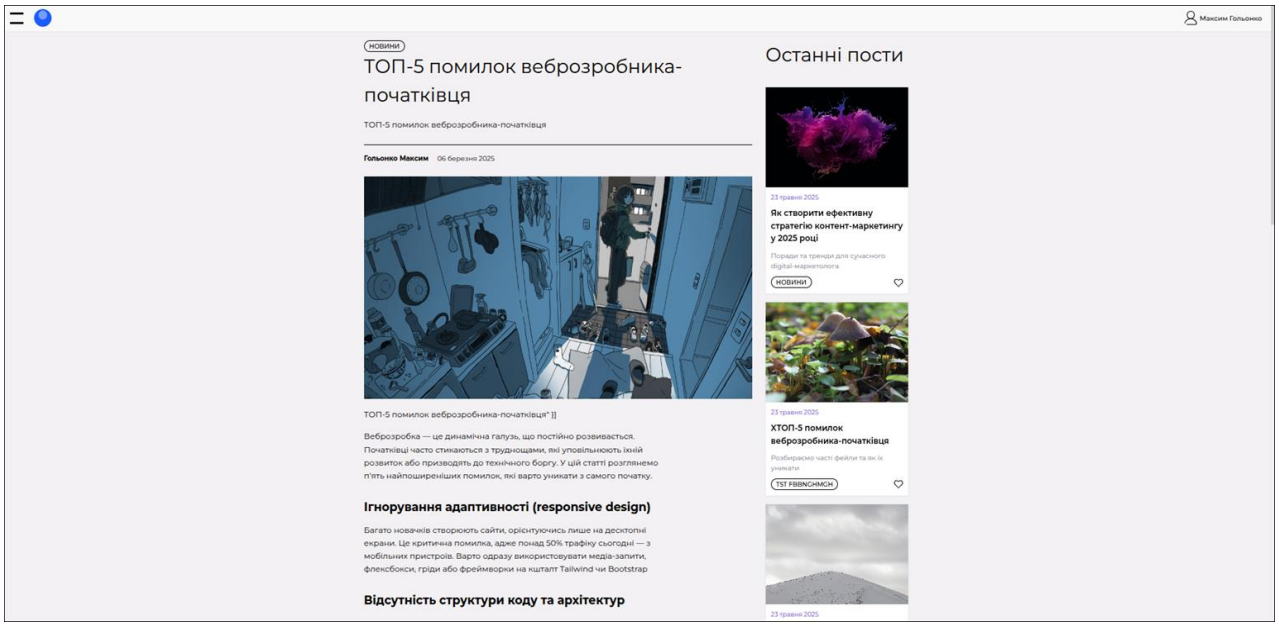

Рисунок 3.12 – Зображення сторінки перегляду посту

ВИСНОВКИ

У результаті кваліфікаційної роботи бакалавра, спрямованої на створення інтерактивного блогу з використанням сучасного технологічного стеку. Було успішно досягнуто:

– проведено детальний аналіз предметної області, фреймворків та бібліотек для frontend і backend розробки, обґрунтовано обрано стек: React, як найгнучкіша бібліотека для побудови динамічних UI, Node.js для асинхронної серверної логіки та MongoDB, як гнучка документо-орієнтована БД;

– розроблена архітектура на основі клієнт-серверною моделі, клієнт реалізований на React, сервер на Node.js з Express.js, база даних MongoDB, взаємодія відбувається через REST API;

– реалізована клієнська частина у вигляді SPA-додатоку на React, із використанням архітектури Feature-Sliced Design, що забезпечує масштабованість. UI реалізований з використанням Tailwind CSS та Mantine, забезпечено адаптивність, інтуїтивність інтерфейсу, високу швидкодію завдяки client-side rendering;

– розроблена серверна частинсу, як RESTful API з маршрутизацією, обробкою запитів і middleware-логікою, для валідації використано бібліотеки express-validator і кастомні middleware, реалізовано модульну структуру проєкту у вигляді DTO, controller, service, model, middleware, router;

– спроєктувано і розроблено схеми даних для користувачів, постів, улюбленого контенту, коментарів тощо, враховано принципи нормалізації, індексації та референцій між колекціями;

– реалізовано функціональ реєстрації, логіну, зберігання JWT токенів, а також middleware для перевірки ролей та застосовано хешування паролів за допомогою бібліотеки bcrypt;

– реалізовано функціонал для корисувачів, а саме створювання, редагування, перегляд та видалення постів, реалізовано перевірку авторства перед редагуванням або видаленням;

– клієнтська частина розгорнута на Vercel, серверна частиан на Render, для зберігання зображень, була інтегрована система Cloudinary, яка працює в онлайн-середовищі.

Подальші шляхи вдосконалення передбачають розширення функціональності, наприклад, впровадження push-сповіщень, розумного пошуку, масштабування до мікросервісної архітектури, а також перенесення серверної частини в контейнеризоване середовище для покращення розгортання та підтримки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. React. URL: https://react.dev/ (дата звернення: 08.03.2025).

2. Index. Node.js v24.1.0 Documentation. Node.js – Run JavaScript Everywhere. URL: https://nodejs.org/docs/latest/api/ (дата звернення: 08.03.2025).

3. Учасники проектів Вікімедіа. REST – Вікіпедія. URL: https://uk.wikipedia.org/wiki/REST (дата звернення: 08.03.2025).

4. Team M. D. MongoDB Documentation. MongoDB: The World's Leading Modern Database. MongoDB. URL: https://www.mongodb.com/docs/ (дата звернення: 08.03.2025).

5. WordPress. Your Way. WordPress.com. URL: https://wordpress.com/ (дата звернення: 08.03.2025).

6. Drupal community in Ukraine. Drupal UA. URL: https://drupal.org.ua/en (дата звернення: 08.03.2025).

7. Medium. Read and write stories. URL: https://medium.com/ (дата звернення: 14.04.2025).

8. RFC 7519. JSON Web Token (JWT). IETF Datatracker. URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 14.04.2025).

9. GitHub – bcrypt for NodeJs. URL: https://github.com/kelektiv/node.bcrypt.js (дата звернення: 14.04.2025).

10. Cloudinary Image & Video Management – Documentation Home. Cloudinary. Image and Video Upload, Storage, Optimization and CDN. URL: https://cloudinary.com/documentation (дата звернення: 14.04.2025).

11. JavaScript With Syntax For Types. TypeScript: JavaScript With Syntax For Types. URL: https://www.typescriptlang.org/ (date of access: 18.04.2025).

12. GitHub. Build and ship software on a single, collaborative platform. GitHub. URL: https://github.com/ (дата звернення: 18.04.2025).

13. Vercel Documentation. Vercel: Build and deploy the best web experiences with the Frontend Cloud. URL: https://vercel.com/docs (дата звернення: 20.04.2025).

14. Docs + Quickstarts. Render. Cloud Application Platform. Render. URL: https://render.com/docs (дата звернення: 23.05.2025).

15. GitHub. An express.js middleware for validator.js. URL: https://github.com/express-validator/express-validator (дата звернення: 23.05.2025).

16. GitHub. Basic rate-limiting middleware for the Express web server. URL: https://github.com/express-rate-limit/express-rate-limit (дата звернення: 24.05.2025).

ДОДАТКИ

Додаток А – Апробація

Міністерство освіти і науки України
Волинська обласна рада
Луцька міська рада
Луцький національний технічний університет
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
Національний університет «Львівська політехніка»
Військовий інститут телекомунікацій та інформатизації імені Героїв Крут (м. Київ)
Тернопільський національний педагогічний університет імені В. Гнатюка
Рівненський державний гуманітарний університет
Навчально-науковий інститут «Українська інженерно-педагогічна академія» Харківського національного університету ім. В.Н. Каразіна
Люблінська політехніка (Польща)
Фрайберзька гірнича академія (Німеччина)
Гронінгенський університет (Нідерланди)
Кавказький університет (Грузія)
Політехнічний університет Браганси (Португалія)

ТЕЗИ ДОПОВІДЕЙ
X Міжнародної науково-практичної конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025)

23-24 травня 2025 р.

Луцьк 2025

СЕКЦІЯ 4. ПРИКЛАДНІ ЗАСОБИ ПРОГРАМУВАННЯ І ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

На основі даного аналізу UML-діаграм основних етапів розробки мобільного додатку можна зробити наступний висновок. Інструментарій Expo суттєво зменшує час на розробку мобільного додатку, через своє функціональне та гнучке середовище. Expo зменшує поріг входу у мобільну розробку, роблячи її доступною навіть для новачків, тоді як традиційний підхід передбачає знання нативних інструментів і суттєво більшу технічну підготовку.

Список використаних джерел

1. Managed workflow. Expo Documentation. URL: https://docs.expo.dev/bare/overview/#new-workflows (date of access: 9.05.2025).
2. Bare workflow. Expo Documentation. URL: https://docs.expo.dev/versions/latest/config/metro/#bare-workflow-setup (date of access: 10.05.2025).

УДК 004.4

АКТУАЛЬНІ ПРАКТИКИ СТВОРЕННЯ ВЕБПЛАТФОРМ З ВИКОРИСТАННЯМ REACT ТА СУЧАСНОГО СТЕКУ ТЕХНОЛОГІЙ

Гольонко Максим Анатолійович
Луцький національний технічний університет, здобувач групи ІПЗ-41, golonko.m1309@lntu.edu.ua
Ліщина Наталія Миколаївна
Луцький національний технічний університет, к.т.н., доцент, завідувач кафедри інженерії програмного забезпечення, n.lishchyna@lutsk-ntu.com.ua

CURRENT PRACTICES OF CREATING WEB PLATFORMS USING REACT AND MODERN TECHNOLOGY STACK

Maksym Holonko
Lutsk National Technical University, Student of the Group IPZ-41, golonko.m1309@lntu.edu.ua
Nataliia Lishchyna
Lutsk National Technical University, Ph.D., Associate Professor, Head of the Department of Software Engineering, n.lishchyna@lutsk-ntu.com.ua

The theses highlight the experience of using React together with Node.js in the development of web platforms, compare the features of implementing client-server interaction for storing data in MongoDB.

***Keywords**: react, node.js, mongodb.*

Сучасні вебплатформи дедалі частіше будуються за архітектурною моделлю односторінкових вебдодатків, де головну роль відіграє клієнтська частина застосунку. Одним із провідних рішень для реалізації таких інтерфейсів є React – JavaScript-бібліотека, створена для побудови гнучких і масштабованих інтерфейсів з високою швидкістю реакції на зміну стану застосунку [1]. React дозволяє розробляти вебплатформи, які працюють як повноцінні застосунки з динамічним оновленням вмісту без повного перезавантаження сторінки.

Побудова вебблогу на React зазвичай передбачає реалізацію трьох основних зон логіки: перегляд постів, керування власними публікаціями та керування всіма матеріалами. Кожна зона організована як набір компонентів, які використовують backend API [2] . Компоненти React мають власний локальний стан, або ж отримують глобальні дані через контекст чи менеджери стану. Це дозволяє в реальному часі перемальовувати лише ті ділянки сторінки, де змінився контент – наприклад, новий коментар з'являється без оновлення всієї статті.

Особливістю React є компонентний підхід до побудови інтерфейсу, де кожен елемент блогу реалізується як окремий компонент. Це дозволяє повторно використовувати код, ізолювати логіку та оновлювати окремі частини без повного ререндеру. Компоненти керуються внутрішнім або глобальним станом, що забезпечує миттєву реакцію інтерфейсу на зміни, наприклад, оновлення списку публікацій після додавання нової статті. Завдяки віртуальному DOM [3], оновлюються лише змінені елементи, що підвищує продуктивність, особливо для динамічних блогів із частою взаємодією з контентом.

React відзначається також розширюваністю. Велика екосистема підтримує інтеграцію з інструментами маршрутизації керування станом за допомогою Redux або аналогів, асинхронного обміну даними та формування запитів до API. Це дозволяє ефективно реалізовувати клієнт-серверну взаємодію, де React відповідає за ініціацію запитів і відображення результатів, отриманих від сервера.

У типовій архітектурі платформи React працює у зв'язці з Node.js як серверною платформою. Node.js, використовуючи Express.js, забезпечує обробку запитів, маршрутизацію, авторизацію та взаємодію з базою даних. У такій моделі React виступає як фронтенд, що взаємодіє з RESTful API, створеним на базі Node.js. Це дає змогу повністю розділити клієнтську та серверну логіку, дотримуючись принципів розділення відповідальностей.

Для збереження даних використовується MongoDB – документоорієнтована база даних, яка природно інтегрується з React-застосунками через формат JSON. Дані, що надходять з клієнта, легко перетворюються у BSON-документи, які зберігаються у колекціях MongoDB. Використання бібліотеки Mongoose у Node.js дозволяє описати структуру даних, накласти правила валідації та формувати запити до бази з мінімальними затратами. MongoDB зберігає усі структурні дані вебблогу: користувачів, статті, коментарі, позначки «обране», метаінформацію про авторство, дати, теги тощо. Формат зберігання BSON дозволяє вкладені структури, які природно відповідають об'єктам у JavaScript, що спрощує передачу даних з Node.js до React.

На рівні серверної логіки використання маршрутизації за допомогою Express.js дозволяє чітко розділити запити за методами GET, POST, PUT, DELETE, а бібліотека Mongoose – створити модельні об'єкти для MongoDB та легко зв'язати між собою сутності: наприклад, користувач → стаття, статті → коментарі. Це забезпечує просту та прозору логіку запитів між frontend і backend частинами. У межах практичної реалізації

застосунку типу блог, де React відповідає за динамічне відображення контенту, а Node.js із MongoDB – за зберігання даних та реалізацію API. Реалізовано повний цикл обробки статей: створення, редагування, видалення та отримання через API. Такий підхід підтвердив ефективність React як основного засобу створення сучасного frontend для вебплатформ у поєднанні з гнучкими backend-рішеннями на базі JavaScript.

Отже, React у поєднанні з Node.js та MongoDB становить ядро сучасної вебплатформи, що дозволяє реалізувати високодинамічні застосунки зі складною логікою, повністю керованим станом та широкими можливостями для масштабування і подальшої розробки.

Список використаних джерел

1. React. *React*. URL: https://react.dev/.
2. API - MDN Web Docs Glossary: Definitions of Web-related terms | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Glossary/API.
3. DOM - MDN Web Docs Glossary: Definitions of Web-related terms | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Glossary/DOM.

Додаток Б – реалізація file-service

```
import { UploadedFile } from 'express-fileupload';
import fs from 'fs';
import path from 'path';
import { v4 as uuidv4 } from 'uuid';
import cloudinary from './cloudinary'; // новий імпорт
import os from 'os';

interface ContentBlock {
    [key: string]: any;
}

class FileService {
    async saveFile(file: UploadedFile): Promise<string | void> {
        try {
            const isProduction = process.env.NODE_ENV === 'production';

            if (isProduction) {
                const tempFilePath = path.join(os.tmpdir(), uuidv4() +
path.extname(file.name));
                await file.mv(tempFilePath);

                const result = await
cloudinary.uploader.upload(tempFilePath, {
                    folder: 'blog-uploads',
                });

                fs.unlinkSync(tempFilePath);
                return result.secure_url;
            } else {
                const fileExtension = path.extname(file.name);
                const fileName = uuidv4() + fileExtension;
                const filePath = path.resolve('static', fileName);

                if (!fs.existsSync(path.resolve('static'))) {
                    fs.mkdirSync(path.resolve('static'), { recursive:
true });
                }

                await file.mv(filePath);
                return `${process.env.API_URL}/static/${fileName}`;
            }
        } catch (e) {
            console.error('Error saving file:', e);
        }
    }

    async processContentBlocks(blocks: ContentBlock[]):
Promise<ContentBlock[]> {
        return Promise.all(
            blocks.map(async (block: ContentBlock) => {
```

```
                if (block.type === 'image' &&
block.attrs?.src?.startsWith('data:image')) {
                    const base64Data = block.attrs.src.split(',')[1];
                    const buffer = Buffer.from(base64Data, 'base64');
                    const file: UploadedFile = {
                        name: 'image.png',
                        data: buffer,
                        size: buffer.length,
                        mimetype: 'image/png',
                        mv: (path: string) => {
                            return new Promise<void>((resolve, reject) =>
{
                                fs.writeFile(path, buffer, (err: any) =>
{
                                    if (err) reject(err);
                                    else resolve();
                                });
                            });
                        }
                    } as UploadedFile;

                    const savedUrl = await this.saveFile(file);
                    if (savedUrl) {
                        block.attrs.src = savedUrl;
                    }
                }
                return block;
            })
        );
    }
}

export default new FileService();
```