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

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка вебзастосунку анонімного іміджборду

Виконав: студент IV курсу, групи СН-42
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

(підпис) Прибега О. С.
(прізвище та ініціали)

Керівник (підпис) Фриз М. Є.
(прізвище та ініціали)

Нормоконтроль (підпис) Липак Г.І.
(прізвище та ініціали)

Завідувач кафедри (підпис) Боднарчук І.О.
(прізвище та ініціали)

Рецензент (підпис) Осухівська Г. М.
(прізвище та ініціали)

Тернопіль
2026

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

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«_8_» _червня_ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)
за спеціальністю 122 Комп’ютерні науки
(шифр і назва спеціальності)
Студенту Прибезі Олександру Святославовичу
(прізвище, ім’я, по батькові)
1. Тема роботи Розробка вебзастосунку анонімного іміджборду

Керівник роботи Фриз Михайло Євгенович, к.т.н., доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом ректора від «*14*» *травня* 2026 року № *4/9-239*
2. Термін подання студентом завершеної роботи 22 червня 2026 р.
3. Вихідні дані до роботи

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ
1. Аналіз предметної області та архітектури системи
2. Проєктна частина та інструментальні засоби
3. Практична реалізація веб-сайту
4. Безпека життєдіяльності, основи охорони праці
Висновки
Перелік джерел

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці			

7. Дата видачі завдання 26 січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	26.01.2026	
2.	Підбір та опрацювання літературних джерел по темі кваліфікаційної роботи	27.01.2026-16.02.2026	
3.	Виконання дослідження згідно теми кваліфікаційної	17.02.2026-10.05.2026	
	роботи		
4.	Оформлення розділу «Аналіз предметної області	11.05.2026-17.05.2026	
	та архітектури системи»		
5.	Оформлення розділу «Проєктна частина та	18.05.2026-24.05.2026	
	інструментальні засоби»		
6.	Оформлення розділу «Практична реалізація	25.05.2026-31.05.2026	
	веб-сайту»		
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	01.06.2026-08.06.2026	
8.	Виконання завдання до підрозділу «Основи охорони	01.06.2026-08.06.2026	
	праці»		
9.	Оформлення кваліфікаційної роботи	09.06.2026-11.06.2026	
10.	Нормоконтроль	12.06.2026-15.06.2026	
11.	Перевірка на плагіат	16.06.2026	
12.	Попередній захист кваліфікаційної роботи	19.06.2026	
13.	Захист кваліфікаційної роботи	25.06.2026	

Студент _____ Прибега О. С.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Фриз М. Є.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка вебзастосунку анонімного імідж-борду // Кваліфікаційна робота освітнього ступеня «Бакалавр» // Прибега Олександр Святославович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СН-42 // Тернопіль, 2026 // С. 64, рис. – 11, табл. – 6, бібліогр. – 35.

Ключові слова: іміджборд, Node.js, Express.js, MySQL, вебзастосунок, клієнт-серверна архітектура, MVC, асинхронність.

Кваліфікаційна робота присвячена дослідженню та розробці програмного забезпечення для анонімного іміджборду. В першому розділі кваліфікаційної роботи описано актуальність створення анонімних дискусійних платформ, проаналізовано існуючі технологічні рішення для веброзробки та обґрунтовано вибір технологічного стека, що базується на середовищі Node.js, фреймворку Express.js та системі керування базами даних MySQL. Висвітлено переваги використання реляційних баз даних для забезпечення цілісності даних у ієрархічній структурі іміджборду. Розглянуто архітектурні патерни, що забезпечують високу швидкість обробки контенту.

В другому розділі кваліфікаційної роботи спроєктовано трирівневу клієнт-серверну архітектуру на основі патерну MVC. Досліджено структуру бази даних, розроблено схему взаємозв'язків між таблицями (дошки, треди, пости), а також визначено принципи маршрутизації API-запитів для забезпечення ефективної взаємодії клієнтської та серверної частин. Подано опис логіки роботи сервісного шару системи.

В третьому розділі кваліфікаційної роботи описано процес практичної реалізації системи. Проведено налаштування асинхронної обробки медіаконтенту, розроблено механізми безпеки та валідації вхідних даних. Проведено тестування стабільності API за допомогою інструментарію Postman.

Підтверджено ефективність та масштабованість розробленої архітектури для створення високонавантажених дискусійних вебплатформ.

Об’єкт дослідження: процес побудови високонавантажених вебзастосунків для інтерактивного обміну контентом.

Предмет дослідження: методи та інструменти розробки анонімних дискусійних платформ на основі JavaScript-технологій.

ANNOTATION

Development of an Anonymous Imageboard Web Application // Qualification work of the educational level «Bachelor» // Prybega Oleksandr Svyatoslavovych // Ternopil Ivan Pulyu National Technical University, Computer and Information Systems and Software Engineering Faculty, Computer Sciences Department, group SN-42 // Ternopil, 2026 // P. 64, fig. – 11, tabl. – 6, references – 35.

Keywords: imageboard, Node.js, Express.js, MySQL, web application, client-server architecture, MVC, asynchrony.

The qualification work is dedicated to the research and development of software for an anonymous imageboard. The first section of the qualification paper considers the relevance of creating anonymous discussion platforms, analyzes existing technological solutions for web development, and justifies the choice of a technology stack based on the Node.js environment, the Express.js framework, and the MySQL database management system. It highlights the advantages of using relational databases for ensuring data integrity within the hierarchical structure of an imageboard. Architectural patterns that ensure high content processing speed are examined.

In the second section of the qualification work, a three-tier client-server architecture based on the MVC pattern is designed. The database structure is investigated, a scheme of relationships between tables (boards, threads, posts) is developed, and the principles of routing API requests are defined to ensure effective interaction between the client and server parts. A description of the system's service layer logic is provided.

In the third section of the qualification work, the process of practical system implementation is described. Asynchronous media content processing is configured, and security and input data validation mechanisms are developed. The stability of the API is tested using the Postman toolkit. The effectiveness and scalability of the developed architecture for creating high-load discussion web platforms are confirmed.

Object of research: the process of building high-load web applications for interactive content exchange.

Subject of research: methods and tools for developing anonymous discussion platforms based on JavaScript technologies.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API (англ. Application Programming Interface) – інтерфейс прикладного програмування, набір готових інструментів для побудови програмного забезпечення.

CSS (англ. Cascading Style Sheets) – каскадні таблиці стилів, що використовуються для опису зовнішнього вигляду документів.

HTML (англ. HyperText Markup Language) – стандартна мова розмітки документів для вебпереглядачів.

HTTP (англ. HyperText Transfer Protocol) – протокол передачі гіпертексту, що використовується для обміну даними в мережі Інтернет.

JSON (англ. JavaScript Object Notation) – текстовий формат обміну даними, що базується на JavaScript.

MVC (англ. Model-View-Controller) – архітектурний патерн, що розділяє застосунок на три логічні частини: модель, представлення та контролер.

MySQL – система управління базами даних (СУБД) з відкритим кодом.

Node.js – програмна платформа, що базується на рушії V8, яка дозволяє виконувати JavaScript-код на сервері.

SQL (англ. Structured Query Language) – мова структурованих запитів, що використовується для створення, модифікації та керування даними в реляційних базах даних.

URL (англ. Uniform Resource Locator) – уніфікований локатор ресурсів, що вказує адресу сторінки або ресурсу в мережі.

БД – база даних, організована структура, призначена для зберігання, зміни та обробки взаємопов’язаної інформації.

СУБД – система управління базами даних, сукупність програмних засобів для створення та роботи з БД.

ЗМІСТ

ВСТУП

Стрімкий розвиток інтернет-технологій та попит на анонімні платформи для спілкування зумовлюють необхідність створення високоефективних вебзастосунків. Іміджборди вимагають особливих архітектурних підходів до обробки медіаконтенту та забезпечення стабільної швидкодії за умов високого навантаження. Використання асинхронних технологій, зокрема середовища Node.js, є оптимальним рішенням для вирішення цих завдань, що підкреслює актуальність теми дослідження.

Метою роботи є проєктування та програмна реалізація анонімного іміджборду із застосуванням трирівневої архітектури та патерну MVC. Для досягнення поставленої мети передбачено виконання таких завдань: аналіз технологічного стека (Node.js, Express.js, MySQL); проєктування архітектури системи; реалізація функціоналу для створення тредів, публікації контенту та управління правами доступу; налаштування системи тестування API.

Об'єкт дослідження: процес побудови високонавантажених вебзастосунків для інтерактивного обміну контентом.

Предмет дослідження: методи та інструменти розробки анонімних дискусійних платформ на основі JavaScript-технологій.

Розроблена модульна архітектура забезпечує легке масштабування проєкту та впровадження нових функцій, а застосування асинхронної моделі Node.js гарантує швидкий відгук системи та ефективну роботу з даними, що робить програмний продукт придатним для розгортання як повноцінного дискусійного майданчика.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА АРХІТЕКТУРИ СИСТЕМИ

1.1 Аналіз особливостей функціонування анонімних іміджбордів

Анонімний іміджборд – це специфічний різновид вебфоруму, ключовою особливістю якого є пріоритет візуального контенту та повна або часткова відсутність обов’язкової реєстрації користувачів [1]. На відміну від класичних форумів, де дискусія будується навколо профілів користувачів та їхньої історії активності, іміджборд зосереджений на миттєвому обміні інформацією, що забезпечує унікальний динамічний характер взаємодії [2].

Функціонування іміджборду базується на принципі ефемерності даних [3]. Основною одиницею контенту є «тред» (thread) – тематична гілка обговорення, яка розпочинається з публікації стартового повідомлення (OP-post). Особливістю цього формату є механізм «бампу» (bump – від англ. to bump, «штовхати»), який автоматично піднімає тред на початок списку при кожній новій відповіді. Така динаміка створює «природний відбір» тем: актуальні дискусії залишаються в полі зору, тоді як неактивні треди поступово зміщуються до кінця списку і згодом видаляються (процес «pruning») [1, 2].

Таблиця 1.1 – Порівняння класичного форуму та іміджборда

	Класичний форум	**Іміджборд**
Реєстрація	Обов'язкова	Відсутня / Опціональна
Ідентичність	Нікнейм, аватар, підпис	Анонім
Контент	Текстово-орієнтований	Зображення-орієнтований
Життєвий цикл тем	Статичний	Динамічний (через бамп)
Зберігання даних	Архівування	Тимчасове (видалення)

Такий підхід до організації контенту мінімізує соціальні бар'єри. Відсутність постійних облікових записів нівелює фактор «репутації» користувача, фокусуючи дискусію на змістовному наповненні повідомлення, а не на особистості автора. Це дозволяє здійснювати максимально відвертий обмін думками, проте накладає суттєві вимоги до архітектури системи [4], зокрема щодо модерації контенту, боротьби зі спамом та забезпечення високої швидкості обробки запитів [3].

Для ефективної навігації в межах великого обсягу неструктурованих даних, в іміджбордах застосовується багаторівнева ієрархія [5]. Вона дозволяє користувачам швидко сегментувати контент за тематичними інтересами, що є критично важливим для утримання уваги аудиторії в умовах високої динаміки оновлення стрічки.

Першим рівнем організації є дошки (boards) – тематичні розділи, що позначаються унікальними ідентифікаторами (наприклад, /tech/ для технологічних обговорень, /art/ для графіки або /news/ для новин). Такий підхід забезпечує горизонтальну масштабованість системи: при зростанні аудиторії адміністрація може легко створювати нові дошки без зміни базової архітектури сайту, що дозволяє користувачам ефективно фільтрувати контент за власними вподобаннями [6].

Одиницею дискусії всередині кожної дошки виступає тред (thread). Тред ініціюється користувачем (OP або Original Poster), який створює стартовий пост (OP-post), що містить основне повідомлення та, як правило, прикріплений медіафайл. Подальша взаємодія відбувається через додавання відповідей, що формують лінійну, проте в контексті загальної структури дошки – деревоподібну послідовність повідомлень.

На рисунку 1.1 наведено діаграму ієрархічної структури даних, що реалізується в межах розроблюваного іміджборду. Система базується на дворівневому групуванні: верхній рівень представлений тематичними дошками (наприклад, /news/ та /art/), які виступають контейнерами для окремих обговорень – тредів. Кожен тред, своєю чергою, є послідовним ланцюгом постів,

де перший пост (OP) визначає тему дискусії, а наступні утворюють лінійну хронологію відповідей. Така структура забезпечує логічне розмежування контенту та дозволяє користувачам ефективно орієнтуватися у великих обсягах даних, що надходять у режимі реального часу.

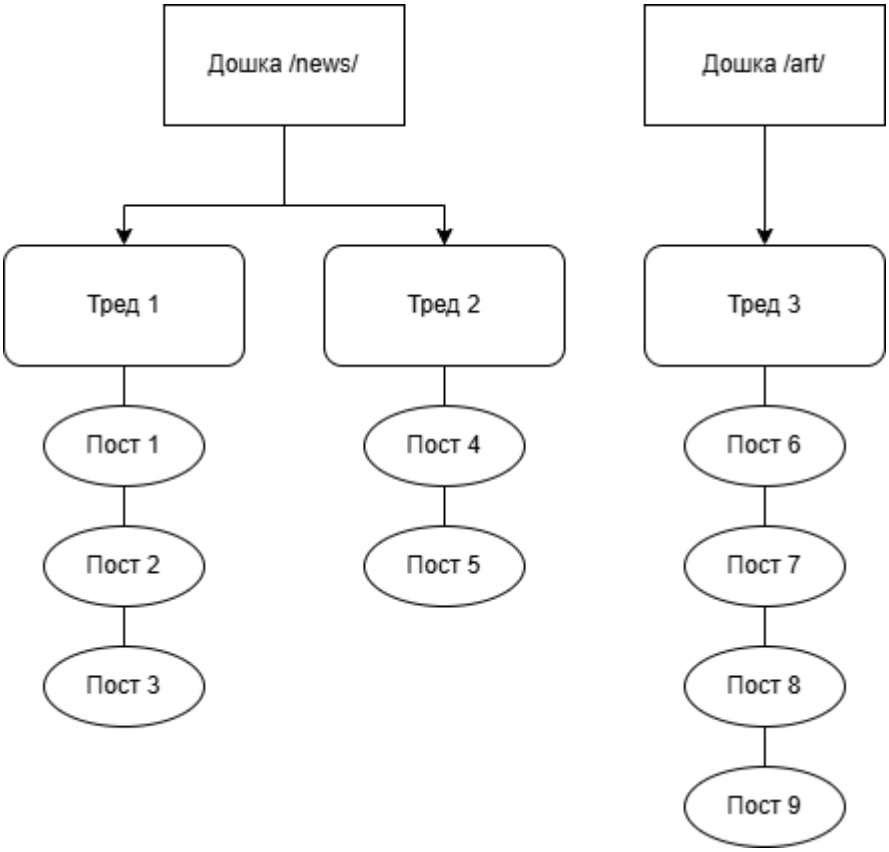

Рисунок 1.1 – Ієрархічна модель організації контенту

«Вибрана структура іміджборду має виражений децентралізований характер на рівні окремих дощок, що дозволяє легко масштабувати проєкт шляхом додавання нових тематичних розділів. З точки зору реалізації бази даних, кожен пост є окремою сутністю, яка має прив’язку до конкретного треду, а тред – до конкретної дошки. Така модель є оптимальною для реляційних баз даних [7, 8], оскільки вона мінімізує надмірність даних (data redundancy) та дозволяє реалізувати швидкі запити на вибірку за ідентифікатором треду, що є критично важливим для швидкості завантаження сторінок при високому навантаженні на вебзастосунок [9]».

Ключовим алгоритмічним елементом підтримки активності тредів є механізм «бампу» (bump). Це правило, за яким будь-яка нова відповідь у треді автоматично переміщує його на першу позицію в списку активних тем дошки. Цей процес забезпечує природну селекцію контенту: теми, що викликають жваве обговорення, постійно залишаються в топі (актуальними), тоді як треди, що не отримують нових відповідей протягом певного часу, поступово зміщуються до кінця списку. Коли тред досягає нижньої межі сторінок, він підлягає видаленню (процес, відомий як «pruning» або «топлення»).

Описаний механізм бампу забезпечує саморегуляцію контенту, дозволяючи системі автоматично підтримувати актуальність інформаційного потоку без постійного втручання адміністраторів, а також гарантує ефективний захист від переповнення сервера, оскільки автоматичне видалення неактивних тредів звільняє дисковий простір та оптимізує навантаження на базу даних [10], що є критично важливим для стабільності вебзастосунку при опрацюванні великих обсягів медіаконтенту.

Додатково, для забезпечення гнучкості адміністративного управління, система підтримує функціонал «саги» (автоматичного обмеження кількості повідомлень у треді) та інструменти примусового «закриття» обговорень, що унеможливлює подальшу публікацію відповідей у конкретній гілці. Така висока динаміка взаємодії, що передбачає постійне оновлення стрічки, автоматичне сортування за алгоритмом бампу та миттєве відображення нових постів, висуває підвищені вимоги до клієнт-серверної архітектури застосунку [11, 12]. Саме тому вибір ефективної моделі обробки даних, здатної забезпечити високу швидкість виконання асинхронних запитів [13], стає фундаментальним етапом проєктування, що визначає продуктивність системи в умовах активного користувацького трафіку.

1.2 Порівняльний аналіз існуючих рішень та визначення конкурентних переваг розроблюваної системи

Для визначення вектора розвитку та технічних особливостей розроблюваного вебзастосунку було проведено аналіз популярних іміджбордів: 2ch, 4chan та Кропивач. Незважаючи на їхню популярність, більшість із них базуються на застарілих програмних рішеннях, що обмежує їхню масштабованість, швидкість опрацювання запитів та можливості інтеграції сучасних інтерфейсних рішень [14]. Порівняльна характеристика розроблюваної системи та існуючих аналогів представлена у таблиці 1.2.

Таблиця 1.2 – Порівняння розроблюваної системи та існуючих аналогів

	Традиційні іміджборди(4chan, 2ch)	**Розроблювана система**
Технологічний стек	Застарілі движки (PHP/Perl)	Сучасний стек (Node.js, Express)
Асинхронність	Обмежена (вимагає оновлення сторінки)	Висока (WebSockets/AJAX)
Локалізація	Російськомовний/Англомовний сегмент	Україномовний фокус
Робота з медіа	Базова (статичні зображення)	Кастомна бібліотека стікерів, GIF, емодзі
Інтерфейс	Консервативний (мінімалізм 2000-х)	Адаптивний та інтерактивний

На основі даних, наведених у таблиці 1.2, можна виділити ключові переваги розроблюваної системи, що забезпечують її актуальність та технологічну перевагу над застарілими аналогами:

- Використання Node.js та фреймворку Express дозволяє відійти від традиційної моделі обробки запитів, притаманної PHP-рішенням [15, 16].

Завдяки подієво-орієнтованій архітектурі, система здатна ефективно обробляти тисячі одночасних з’єднань, забезпечуючи реальну асинхронність у передачі даних [17]. Користувач отримує оновлений контент без необхідності оновлення сторінки, що кардинально покращує динаміку спілкування та зменшує навантаження на мережевий канал [18].

- На відміну від 2ch або 4chan, що орієнтовані на іншомовні аудиторії, розроблюваний проєкт є повністю україномовним. Це не лише зручність навігації, а й стратегічний крок для згуртування української інтернет-спільноти. Створення локалізованого майданчика сприяє розвитку національного сегмента мережі, дозволяючи користувачам обговорювати культурні та соціальні питання у рідному мовному середовищі, що стає вагомим внеском у розвиток вітчизняного цифрового простору [19].

- Замість використання стандартного мінімалізму, характерного для іміджбордів 2000-х років, наш застосунок інтегрує сучасну бібліотеку кастомних стікерів, емодзі та GIF-анімацій. Можливість користувачів самостійно завантажувати власні графічні елементи до загальної бібліотеки значно спрощує процес створення унікального контенту [20]. Це усуває потребу в локальному зберіганні великих баз мемів на пристроях користувачів, оскільки вся необхідна графічна база доступна безпосередньо через інтерфейс застосунку.

- На відміну від застарілих вебсайтів, що потребують значного масштабування на мобільних пристроях, розроблюваний інтерфейс є повністю адаптивним. Застосування сучасних стандартів CSS та JS дозволяє платформі коректно відображатися на будь-яких пристроях – від смартфонів до широкоформатних моніторів [21, 22]. Це робить користування платформою зручним незалежно від того, звідки користувач отримує доступ до сервісу.

1.3 Опис клієнт-серверної моделі сайту

В основу програмної реалізації іміджборду покладено трирівневу клієнт-серверну архітектуру [23]. Цей підхід є галузевим стандартом, оскільки забезпечує чітке розмежування відповідальності між інтерфейсом користувача (презентаційний рівень), обробкою бізнес-логіки (рівень застосунку) та зберіганням даних (рівень даних).

Клієнтська частина (фронтенд) виконує роль візуального представлення системи, реалізованого через веббраузер [24]. Вона відповідає за відображення списку дощок, тредів та окремих постів, а також за забезпечення інтерактивності (наприклад, надсилання нових повідомлень, завантаження медіафайлів та перемикання між сторінками). Взаємодія з сервером відбувається через асинхронні запити, що дозволяє користувачу отримувати оновлення контенту без необхідності перезавантаження всієї сторінки, забезпечуючи плавність роботи інтерфейсу.

Серверна частина (бекенд), побудована на базі середовища Node.js, виконує ключові функції управління даними та забезпечення безпеки [26]. Сервер виступає посередником між клієнтом та системою керування базами даних (MySQL). Він отримує HTTP-запити від клієнта, здійснює їхню валідацію, формує та виконує відповідні запити до бази даних, а також обробляє файлові операції, такі як збереження зображень та генерація їхніх прев'ю.

Деталізуючи представлений на рисунку 1.2 принцип функціонування, важливо зазначити, що вся взаємодія в системі базується на моделі Request-Response (запит-відповідь). Цей процес ініціюється виключно на стороні клієнта: браузер користувача формує запит до визначеного API-ендпоінту сервера. Наприклад, при спробі перегляду дошки клієнт надсилає запит, який сервер інтерпретує, перетворює на SQL-запит до MySQL, отримує масив даних і повертає його назад у форматі JSON. Вибір саме цього формату зумовлений його легкістю та зручністю парсингу мовою JavaScript, що є оптимальним для сучасних вебзастосунків [25].

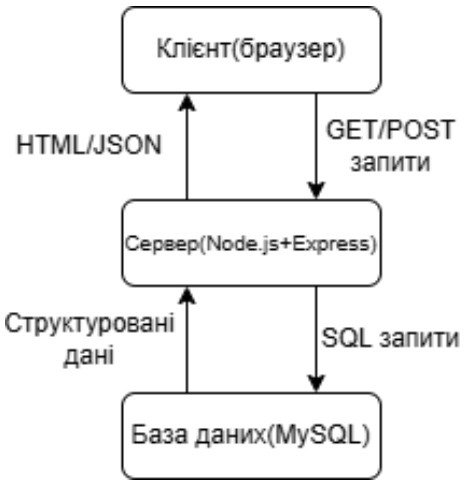

Рисунок 1.2 – Схема клієнт-серверної архітектури

Для забезпечення ефективного функціонування цієї моделі реалізовано чіткий розподіл функцій між компонентами системи:

- Клієнтський рівень відповідає за відображення контенту через маніпуляції з об'єктною моделлю документа (DOM) та збір вхідних даних від користувача. Завдяки застосуванню сучасних підходів до асинхронної обробки даних [24], клієнтська частина мінімізує потребу у повному перезавантаженні сторінки, що створює для користувача ефект швидкого, «живого» інтерфейсу.
- Серверний рівень (Node.js + Express) виконує роль інтелектуального посередника, який не лише забезпечує логіку передачі даних, а й гарантує безпеку та цілісність системи [27]. Сервер здійснює валідацію вхідних даних для запобігання SQL-ін’єкціям [31], контролює права доступу користувачів та керує файловою системою під час завантаження медіаконтенту. Використання Node.js дозволяє уніфікувати мову програмування – JavaScript – для фронтенду та бекенду, що значно спрощує подальшу підтримку та еволюцію проєкту.
- Рівень даних (MySQL) забезпечує надійне зберігання реляційних зв’язків між ключовими елементами іміджборду: дошками, тредів та постами. Використання транзакційних механізмів гарантує цілісність даних [28], а оптимізація індексів дозволяє системі забезпечувати швидкий доступ до

інформації навіть за умов високого навантаження та великих обсягів медіафайлів [29].

Такий підхід забезпечує модульність: кожна частина системи може бути вдосконалена або масштабована незалежно від іншої. Наприклад, при зростанні аудиторії можна інтегрувати додаткові сервіси для обробки зображень, не змінюючи при цьому основну архітектуру API чи структуру бази даних [30].

1.4 Проєктування архітектури сервера

Для забезпечення високої продуктивності та надійності вебзастосунку було обрано трирівневу архітектуру сервера [23]. Це рішення є не просто формальним поділом коду на окремі файли, а стратегічним підходом до реалізації принципу «розділення обов'язків» (Separation of Concerns) [4, 23]. Рівень представлення, реалізований на стороні клієнта, комунікує з бекендом виключно через RESTful API [11]. Завдяки цьому повністю абстрагується клієнтська логіка від внутрішніх процесів сервера: сервер застосунків не перевантажується завданнями з рендерингу інтерфейсу, як це відбувається у традиційних монолітних системах, і фокусується виключно на бізнес-логіці та маршрутизації даних. Це є фундаментальною умовою для забезпечення майбутньої масштабованості системи в умовах динамічного зростання кількості користувачів [13].

Центральним інтелектуальним елементом цієї архітектури виступає сервер застосунків, побудований на базі середовища Node.js та фреймворку Express [15, 16]. Його головна перевага в контексті іміджборду полягає у використанні неблокуючої подієво-орієнтованої моделі [27]. Коли система одночасно отримує сотні запитів на завантаження медіафайлів та читання різних тредів, Node.js ефективно керує чергами завдяки асинхронній обробці, не чекаючи завершення однієї операції для початку іншої [17]. Фреймворк Express при цьому виконує функцію гнучкого маршрутизатора, який миттєво розподіляє вхідний трафік до

відповідних контролерів, паралельно керуючи сесіями та взаємодією з файловою системою для збереження графічного контенту [26].

Третім критичним компонентом є рівень даних, представлений реляційною базою MySQL, який спроєктовано як повністю ізольовану частину системи [8, 28]. Така архітектура кардинально підвищує загальний рівень безпеки: сервер застосунків виступає в ролі єдиного суворого «фільтра». Жоден клієнтський запит не має прямого доступу до бази даних. Уся вхідна інформація проходить через механізми валідації, очистки та параметризації на рівні Node.js, що зводить до мінімуму ризик виникнення SQL-ін'єкцій [31]. Крім того, подібна модульність дозволяє системі легко масштабуватися: за потреби, обробку зображень або саму базу даних можна без проблем мігрувати на окремі фізичні або віртуальні сервери, не порушуючи цілісності коду бекенду [30].

Для детальної візуалізації динаміки взаємодії між компонентами трирівневої архітектури сервера розроблено діаграму послідовності, представлену на Рисунку 1.3. На схемі продемонстровано наскрізний життєвий цикл асинхронного POST-запиту користувача на створення нового повідомлення з прикріпленим медіафайлом.

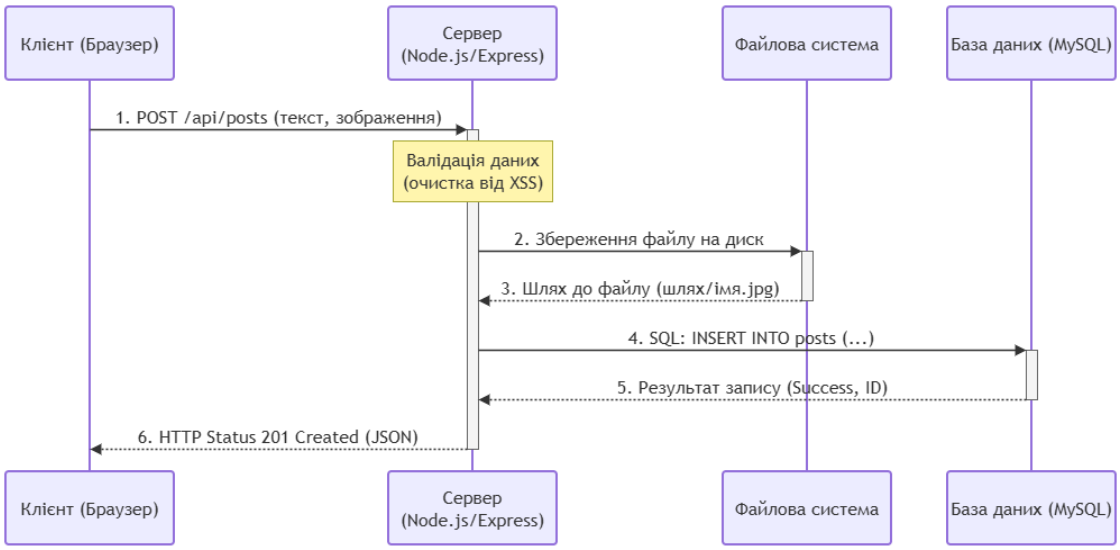

Рисунок 1.3 – Діаграма послідовності процесу публікації контенту

Процес чітко ілюструє бар'єрну функцію сервера застосунків Node.js/Express. Після отримання запиту від клієнтської частини, сервер виконує

внутрішній цикл валідації та санітарії вхідних даних, нейтралізуючи потенційні загрози безпеці [31]. Лише після успішної перевірки та ізольованого збереження файлу у файловій системі, сервер формує параметризований SQL-запит до бази даних MySQL. Таким чином, діаграма підтверджує практичну реалізацію принципу розділення обов'язків та захищеності шару даних від прямого зовнішнього впливу, що детально описано у поточному розділі.

1.5 Моделювання системи

1.5.1 Діаграма прецедентів використання

Для формалізації функціональних вимог до розроблюваного вебзастосунку та визначення меж взаємодії користувачів із системою застосовано методологію об'єктно-орієнтованого моделювання за допомогою мови UML [4, 19]. Ключовим інструментом на етапі аналізу вимог є діаграма прецедентів використання (Use Case Diagram), яка дозволяє чітко розмежувати ролі в системі та визначити набір функцій, доступних кожному типу користувачів [23].

У розроблюваному анонімному іміджборді виділено три основні ролі (актори): анонімний користувач, модератор та адміністратор. Кожна роль має власний рівень доступу до функціоналу платформи, що забезпечує баланс між свободою спілкування та контролем за дотриманням правил ресурсу [1, 3].

Анонімний користувач є базовим актором системи, для якого не передбачено процедури обов'язкової реєстрації [2]. Він має доступ до прецедентів перегляду тематичних дощок та активних тредів, створення нових гілок обговорення (OP-постів) та публікації відповідей. Особливістю цієї ролі є можливість взаємодії з кастомною бібліотекою мультимедіа, що включає використання вбудованих стікерів та GIF-анімацій під час формування повідомлень [20].

Модератор – це роль із розширеними повноваженнями, головним завданням якої є контроль за контентом. Модератор не керує структурою сайту,

але має доступ до інструментів оперативного реагування: прецедентів видалення окремих постів або цілих тредів, що порушують правила, а також блокування правопорушників за допомогою інструментів обмеження доступу (бан за IP-адресою або клієнтським хешем) [31].

Адміністратор володіє повним спектром прав у системі та керує її глобальною конфігурацією. Окрім можливостей модерації, адміністратор взаємодіє з прецедентами створення та редагування тематичних дощок, глобального налаштування лімітів системи (наприклад, параметрів «саги»), а також управління централізованою бібліотекою стікерів та призначення або відкликання прав модераторів [3, 27].

Графічне відображення архітектурної моделі функціональних можливостей, що ілюструє ієрархію прав доступу та взаємозв'язки між акторами та прецедентами, наведено на рисунку 1.4.

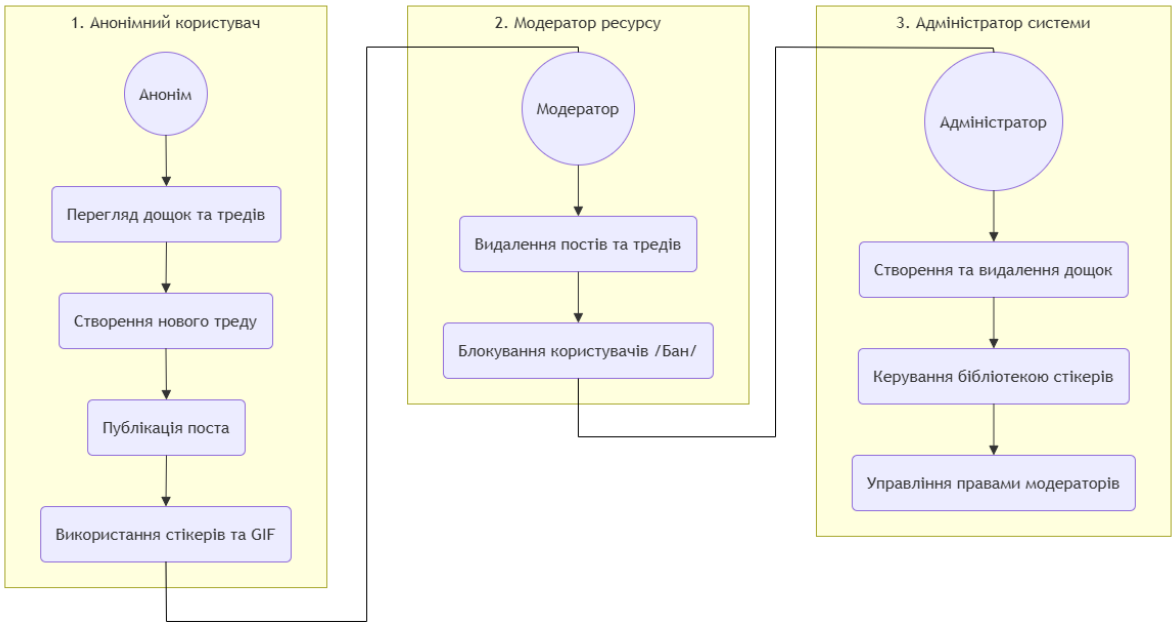

Рисунок 1.4 – Діаграма прецедентів використання розроблюваного вебзастосунку

Аналіз, представлений на рисунку 1.4, демонструє, як базові функції спілкування повністю відкриті для анонімних користувачів, тоді як специфічні інструменти адміністрування та модерації контенту ізольовані на рівні вищих ролей. Така структура діаграми не лише наочно показує розподіл обов'язків, але

й слугує фундаментальним документом для проектування системи безпеки, гарантуючи стабільне управління ресурсом та захист від несанкціонованого втручання [23, 31].

1.5.2 Діаграма розгортання

Для візуалізації фізичної топології системи, визначення конфігурації апаратних засобів та розподілу програмних компонентів по реальних вузлах мережі було розроблено діаграму розгортання (Deployment Diagram). Цей етап моделювання є завершальним у межах першого розділу, оскільки він переводить теоретичну трирівневу архітектуру у площину практичної реалізації та розгортання застосунку на реальних серверах [4, 11]. Діаграма дозволяє оцінити вимоги до середовища виконання, взаємозв'язки між фізичними пристроями та протоколи, що забезпечують їхню комунікацію [23].

Фізична структура розроблюваного іміджборду передбачає чіткий поділ на три основні обчислювальні вузли. Першим вузлом є клієнтський пристрій користувача, яким може виступати персональний комп'ютер або мобільний телефон. На цьому пристрої розгортається лише середовище виконання фронтенду – сучасний веббраузер, який ініціює з'єднання [21, 24]. Комунікація між клієнтським вузлом та сервером застосунків здійснюється через захищені протоколи передачі даних HTTP або HTTPS, що гарантує безпеку користувацьких запитів у мережі Інтернет [31].

Другим та центральним вузлом топології є сервер застосунків, який функціонує під управлінням операційної системи сімейства Linux на базі віртуального або фізичного виділеного сервера (VPS/VDS). Всередині цього вузла розгортається програмне середовище Node.js разом із фреймворком Express, які безпосередньо обробляють логіку іміджборду [15, 16, 26]. Також цей вузол містить локальну або підключену файлову систему, яка виділена під статичне сховище для завантажених користувачами медіафайлів, стікерів та GIF-

анімацій [20]. Сервер застосунків виступає головною точкою входу для зовнішнього трафіку.

Третім ізольованим вузлом є сервер баз даних, на якому функціонує система керування базами даних MySQL [8]. Він може бути розташований як на тому самому фізичному сервері, так і на окремому виділеному дата-сервері для підвищення безпеки та продуктивності [28]. Взаємодія між сервером застосунків Node.js та сервером баз даних MySQL відбувається всередині внутрішньої ізольованої мережі за допомогою захищеного протоколу TCP/IP із використанням спеціалізованого драйвера доступу [29]. Таке рішення унеможливлює пряме підключення до бази даних із зовнішньої мережі Інтернет, забезпечуючи максимальний захист інформації [31].

Візуалізацію фізичної топології системи, яка наочно відображає розподіл компонентів по реальних вузлах мережі, наведено на рисунку 1.5.

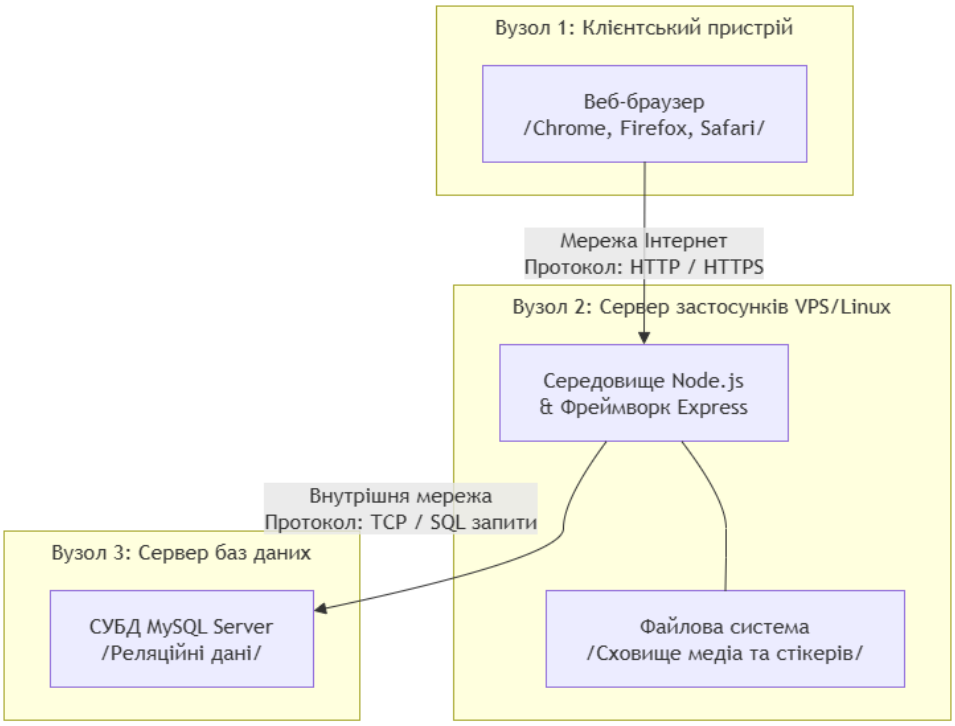

Рисунок 1.5 – Діаграма розгортання компонентів веб-застосунку

На Рисунку 1.5 наочно продемонстровано фізичне розміщення елементів системи та середовища їхнього виконання. Вертикальна структура схеми відображає шлях проходження даних від інтерфейсу користувача до кінцевого сховища. Чітке розмежування середовища виконання Node.js та бази даних

MySQL на рівні вузлів підтверджує архітектурну стійкість проєкту до навантажень та створює надійний периметр безпеки, обмежуючи доступ до даних лише через перевірені внутрішні мережеві запити [27, 30].

РОЗДІЛ 2. ПРОЄКТНА ЧАСТИНА ТА ІНСТРУМЕНТАЛЬНІ ЗАСОБИ

2.1 Обґрунтування вибору серверної платформи та мови програмування

При розробці будь-якого сучасного вебсайту дуже важливо правильно обрати інструменти для написання серверної частини (бекенду). Від цього залежить, наскільки швидко сайт буде працювати, як легко його буде оновлювати в майбутньому та скільки користувачів він зможе витримати одночасно. Для нашого анонімного іміджборду було обрано мову програмування JavaScript та платформу Node.js [15].

Головний плюс такого вибору полягає в тому, що ми використовуємо одну й ту саму мову програмування як для інтерфейсу сайту (фронтенд), так і для сервера. Це дуже зручно, адже розробнику не потрібно постійно перемикатися, наприклад, між PHP на сервері та JavaScript у браузері. Такий підхід значно економить час і зменшує кількість помилок [20].

Сама по собі платформа Node.js – це спеціальне середовище, яке дозволяє виконувати код JavaScript на сервері. Її головна фішка полягає в асинхронній роботі. Наприклад, класичні сервери на базі мов PHP чи Java створюють окремий процес для кожного користувача [17]. Якщо на сайт зайде 1000 людей, сервер може просто зависнути від нестачі пам'яті. Node.js працює інакше: він приймає всі запити в одному потоці і просто відправляє їх виконуватися у фоновому режимі [13].

Для іміджборду, де користувачі постійно оновлюють сторінку, відправляють нові пости та завантажують картинки, така швидкість і легкість платформи Node.js є ідеальним рішенням.

Для обґрунтування доцільності обраного стека було проведено порівняльний аналіз із іншими популярними серверними технологіями. Результати порівняння наведено у таблиці 2.1.

Таблиця 2.1 – Порівняння популярних серверних технологій

	Node.js (JavaScript)	**PHP (Apache)**	**Python (Django)**
Швидкість обробки запитів	Дуже висока (асинхронна модель)	Середня (синхронна модель)	Середня
Використання пам'яті	Низьке	Високе при великих навантаженнях	Середнє
Мова для фронтенду і бекенду	Одна мова (JavaScript)	Різні мови	Різні мови
Складність розробки	Середня	Низька	Висока

Як видно з даних, наведених у Таблиці 2.1, платформа Node.js значно виграє за рахунок своєї асинхронної моделі та економії ресурсів оперативної пам'яті [17]. Крім того, можливість використовувати єдину мову JavaScript для всієї розробки робить цей варіант найбільш вигідним та сучасним для нашого проєкту.

2.2 Аналіз та вибір серверного веб-фреймворку

Щоб не писати весь серверний код з нуля, розробники використовують спеціальні фреймворки – це готові набори інструментів, які спрощують роботу. Для платформи Node.js існує багато різних рішень, але найпопулярнішим залишається Express.js [16]. Саме його було обрано для розробки нашого сайту.

Express.js відрізняється своєю простотою. Він не змушує розробника використовувати якусь складну або жорстку структуру папок, як це роблять інші фреймворки (наприклад, NestJS). Завдяки цьому ми можемо швидко створити

саме таку архітектуру, яка потрібна для іміджборду: окремо виділити логіку для дощок, окремо – для тредів, і окремо – для публікації постів [26].

Основний принцип роботи Express.js будується на так званих «мідлварах» (проміжних обробниках). Це спеціальні функції, через які проходить запит від користувача, перш ніж сервер дасть відповідь. Наприклад, коли модератор хоче видалити пост, запит спочатку проходить через мідлвар перевірки прав доступу [31]. Якщо прав немає, сервер одразу відмовляє, не навантажуючи систему зайвою роботою.

Порівняльний аналіз популярних фреймворків для Node.js наведено у таблиці 2.2.

Таблиця 2.2 – Порівняння веб-фреймворків для Node.js

Фреймворк	**Переваги**	**Недоліки**	**Доцільність для проєкту**
Express.js	Максимальна гнучкість, велика спільнота, багато готових модулів.	Немає строгих правил структури коду.	Оптимальний вибір завдяки легкості та швидкості розробки.
NestJS	Чітка структура, підходить для великих корпоративних проєктів.	Дуже складний в освоєнні, багато зайвого коду.	Надлишковий для цього проєкту.
Fastify	Рекордна швидкість роботи.	Менше готових бібліотек та рішень у мережі.	Можлива альтернатива, але менш зручна.

Проаналізувавши Таблицю 2.2, можна зробити висновок, що Express.js є найбільш збалансованим інструментом для розв'язання наших задач. Він не перевантажує проєкт зайвим кодом, як це робить NestJS, і при цьому має величезну базу готових рішень, що дозволяє значно пришвидшити процес створення серверного API.

2.3 Обґрунтування вибору системи керування базами даних

Будь-який сайт потребує місця для зберігання текстової інформації. У випадку іміджборду в нас є три основні сутності: дошки, треди та пости. Вони жорстко пов'язані між собою. Одна дошка має багато тредів, а один тред містить багато постів.

Існує два головних типи баз даних: реляційні (де дані зберігаються у вигляді чітких таблиць зі зв'язками) та нереляційні (де дані зберігаються цілими документами без жорсткої структури) [28].

Для нашого проєкту було обрано реляційну базу даних MySQL [8]. Головна причина такого вибору – це надійність збереження зв'язків. Якщо модератор видаляє певну тему (тред), база даних MySQL автоматично і безпомилково видалить усі пости, які належали до цієї теми [29]. У нереляційних базах (наприклад, MongoDB) довелося б писати додатковий складний код, щоб знаходити і видаляти ці пости вручну, інакше вони б залишилися в системі як «сміття».

Також MySQL чудово справляється з автоматичним створенням унікальних номерів (ID) для кожного нового поста [7]. Навіть якщо десятки користувачів одночасно натиснуть кнопку «Відправити», база даних присвоїть кожному повідомленню правильний номер без збоїв.

Порівняльна характеристика обраних технологій представлена у таблиці 2.3.

Таблиця 2.3 – Вибір бази даних для архітектури іміджборду

Критерій оцінки	**MySQL (Реляційна)**	**MongoDB (Нереляційна)**
Структура даних	Жорсткі таблиці, чіткий порядок	Гнучкі документи, відсутність строгих правил
Зв'язки між об'єктами	Дуже сильні (зовнішні ключі)	Слабкі, потребують ручного контролю
Каскадне видалення	Підтримується автоматично на рівні бази	Потребує написання додаткового коду
Пошук інформації	Дуже швидкий завдяки індексам	Швидкий, але складніший в налаштуванні

Спираючись на порівняння у Таблиці 2.3, стає очевидним, що для розробленої ієрархічної структури даних потрібен саме реляційний підхід. Використання MySQL гарантує, що всі зв'язки між категоріями та повідомленнями залишатимуться цілісними, а операції пошуку та сортування будуть виконуватися миттєво [28, 29].

2.3.1 Концептуальна модель даних

Концептуальна модель – це первинне уявлення про те, які саме дані будуть зберігатися в нашій системі та як вони пов'язані між собою на логічному рівні, ще до того, як ми почнемо створювати реальні таблиці в MySQL. Для веб-застосунку нашого іміджборду головне завдання бази даних – забезпечити швидку публікацію та чітку структуризацію анонімних повідомлень.

Під час аналізу предметної області було виділено чотири основні сутності, навколо яких будується вся робота сайту:

- Дошка (Board) – це окремий тематичний розділ сайту (наприклад, дошка про технології, відеоігри чи музику). Кожна дошка має свій унікальний короткий літерний код (наприклад, /b/ чи /tech/), назву та опис.
- Тред (Thread) – це конкретна тема для обговорення, яку створює користувач всередині певної дошки. Тред не може існувати сам по собі – він обов'язково прив'язаний до своєї «материнської» дошки.
- Пост (Post) – це будь-яке текстове повідомлення, яке залишає користувач. Перший пост у треді вважається «головним» (оповіщенням про створення теми), а всі наступні пости є відповідями у цьому треді.
- Медіафайл (Media). Оскільки наш іміджборд орієнтований на обмін картинками, стікерами та GIF-анімаціями, кожен файл має зберігатися в базі даних із прив'язкою до конкретного поста. Пост може містити медіафайл, але може бути і суто текстовим.

Логічний зв'язок між цими сутностями будується за суворим каскадним принципом. Одна дошка містить у собі велику кількість тредів (зв'язок типу «один до багатьох»). Своєю чергою, кожен окремий тред складається з багатьох постів (теж зв'язок «один до багатьох»). Що стосується медіафайлів, то тут діє зв'язок, де один пост може мати один прикріплений файл [29]. Така концепція дозволяє уникнути плутанини: дані чітко розкладені по поличках, що гарантує швидкий пошук інформації сервером.

2.3.2 Діаграма «сутність-зв’язок» (ER-діаграма) бази даних

Для точного відображення структури бази даних, типів полів та архітектурних зв'язків між таблицями було розроблено діаграму «сутність-зв’язок» (Entity-Relationship Diagram). Вона є технічним паспортом нашої бази даних MySQL і демонструє, які саме атрибути (стовпчики) має кожна таблиця та як реалізовані зовнішні ключі (Foreign Keys) для забезпечення цілісності інформації.

На Рисунку 2.1 представлена детальна схема реляційної структури бази даних. Як видно з діаграми, таблиця BOARDS є початковою точкою ієрархії. Зв'язок між таблицями BOARDS та THREADS реалізовано через поле board_id, що є зовнішнім ключем. Аналогічно, таблиця POSTS пов'язана з THREADS за допомогою ключа thread_id.

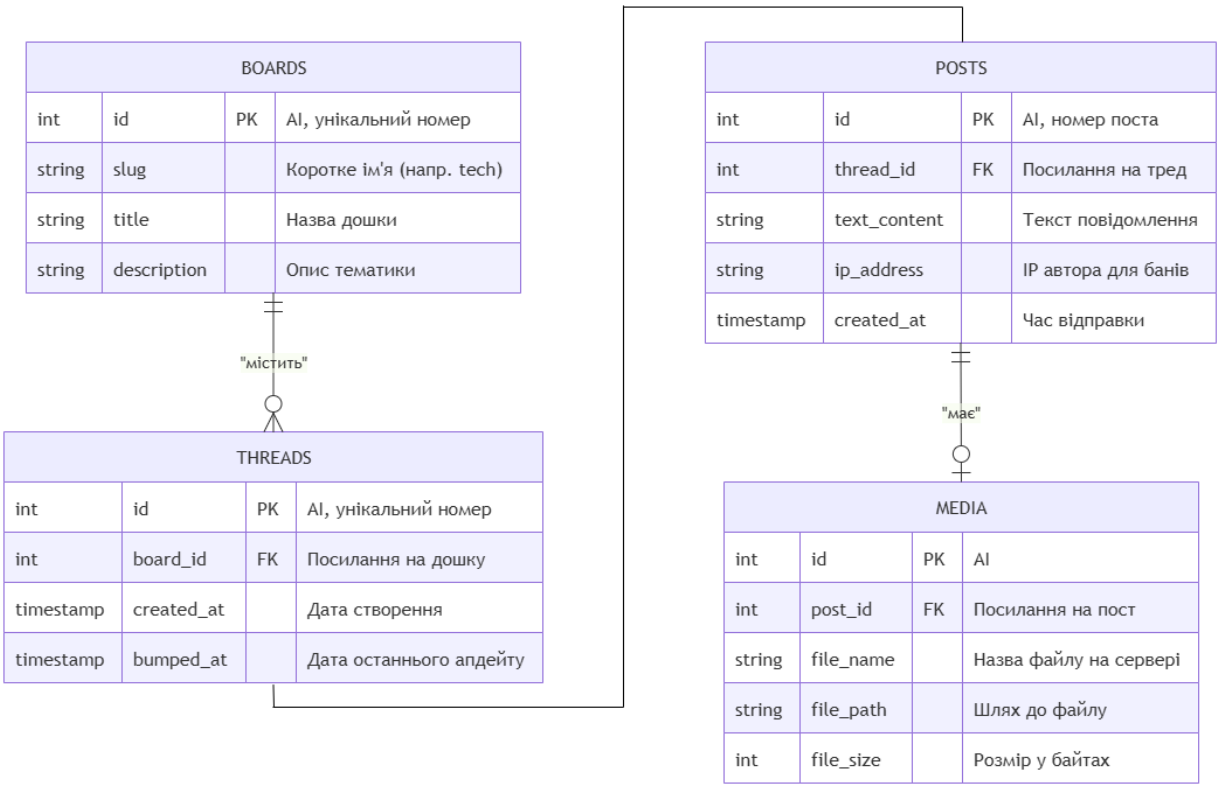

Рисунок 2.1 – Діаграма «сутність-зв’язок» (ER-діаграма) бази даних іміджборду

Особливу увагу приділено таблиці MEDIA: зв'язок із таблицею POSTS організовано через унікальний зовнішній ключ post_id. Налаштування каскадного видалення (ON DELETE CASCADE) на рівні всіх зовнішніх ключів гарантує, що при видаленні модератором будь-якого треду чи поста, всі пов'язані з ними записи та файли автоматично знищаться самою СУБД MySQL. Це унеможливлює появу помилок та накопичення застарілого контенту на сервері [28, 29].

2.4 Огляд допоміжних бібліотек та інструментів розробки

Щоб реалізувати специфічні функції анонімного іміджборду, стандартних можливостей мови JavaScript та базового ядра платформи Node.js недостатньо. Для обробки файлових потоків, валідації медіаконтенту, автоматизації розробки та тестування серверних маршрутів у проєкті було використано набір спеціалізованих бібліотек та професійного програмного забезпечення.

Допоміжні модулі та бібліотеки середовища Node.js:

- Модуль Multer. Ця бібліотека є обов'язковим елементом нашого сервера, оскільки вона відповідає за завантаження файлів. Коли користувач створює пост і прикріплює до нього картинку чи GIF-анімацію, браузер відправляє дані у спеціальному форматі multipart/form-data. Модуль Multer перехоплює цей потік даних, перевіряє розширення файлу, контролює, щоб користувач не завантажив надто великий файл, і безпечно зберігає його у визначену папку на жорсткому диску сервера.
- Бібліотека probe-image-size (модуль probe). Дуже важливий інструмент для оптимізації роботи сайту. Специфіка іміджборду вимагає, щоб сервер знав точну ширину та висоту картинки в пікселях ще до того, як вона буде відображена на сторінці. Модуль probe зчитує лише перші кілька кілобайт файлу (структурний заголовок), миттєво дізнається роздільну здатність зображення і закриває потік, економлячи ресурси сервера [17].

- Модуль nodemon. Спеціальний інструмент для автоматизації процесу розробки. Зазвичай, коли програміст змінює бодай один символ у коді сервера Node.js, йому доводиться вручну зупиняти сервер через консоль і запускати його знову, щоб зміни вступили в дію. Це забирає купу часу. Модуль nodemon постійно стежить за файлами проєкту у фоновому режимі. Як тільки розробник зберігає зміни у коді, nodemon самостійно і миттєво перезапускає сервер за частку секунди.

- Модуль fs (File System). Це вбудований у Node.js інструмент для безпосередньої роботи з файловою системою комп'ютера або сервера. Оскільки на іміджборді користувачі та модератори постійно видаляють пости або цілі треди, разом із текстовим записом у базі даних необхідно фізично стерти й прикріплені картинки з диска. Завдяки модулю fs сервер може перевіряти наявність файлів, створювати нові папки для завантажень та безповоротно видаляти застарілий медіаконтент [27].

- Модуль path. Ще один базовий інструмент Node.js, який допомагає правильно працювати зі шляхами до файлів та папок. Різні операційні системи використовують різні символи для розділення папок у шляху (наприклад, у Windows це зворотний слеш \, а в Linux – прямий /). Щоб наш код працював однаково добре як на домашньому комп'ютері розробника, так і на віддаленому Linux-сервері, модуль path автоматично адаптує та склеює шляхи до картинок і скриптів, запобігаючи критичним збоям у системі.

Для написання коду, організації робочого простору та комплексної перевірки працездатності сервера було використано два ключових програмних комплекси: Visual Studio Code та Postman.

Редактор коду Visual Studio Code (VS Code) – це сучасне, легке та надзвичайно гнучке середовище розробки, яке є стандартом в індустрії створення вебзастосунків. У нашому проєкті VS Code виконував роль головного робочого центру. Завдяки вбудованій системі підсвічування синтаксису мови JavaScript, редактор автоматично підкреслює помилки чи пропущені дужки ще в процесі написання тексту, що мінімізує час на пошук багів.

Також у VS Code інтегровано зручний термінал, який дозволяє керувати сервером, запускати модулі бази даних та відстежувати системні логи в одному вікні, не перемикаючись між різними програмами.

Оскільки наш сайт розробляється за принципом розділення на серверне API та клієнтський інтерфейс, виникає потреба перевіряти роботу сервера окремо від зовнішнього вигляду сайту. Postman – це професійна програма, створена спеціально для тестування веб-API. Вона дозволяє розробнику імітувати будь-які дії користувача в обхід браузера [31].

За допомогою Postman ми можемо вручну відправляти на наш сервер POST-запити зі штучним текстом або картинками, створювати тестові треди, імітувати видалення постів модератором та детально аналізувати відповіді сервера (дізнаватися статус-коди помилок, час обробки запиту та структуру JSON-даних). Це дозволило повністю налагодити і протестувати логіку іміджборду, переконатися в надійності захисту маршрутів від збоїв і гарантувати, що сервер працює стабільно та без помилок ще до того, як була завершена фінальна верстка фронтенду.

Комплекс використаних технологій та інструментів наведено у таблиці 2.4.

Таблиця 2.4 – Призначення інструментарію розробки

Інструмент / Бібліотека	**Основна роль у проєкті**	**Ефект від використання**
Multer	Обробка завантажень multipart/form-data	Безпечний прийом картинок від користувачів
probe-image-size	Швидке зчитування метаданих зображень	Економія ОЗП та оптимізація сітки сайту

Продовження таблиці 2.4

1	**2**	**3**

fs / path	Робота з файловою системою сервера	Надійна робота з файлами незалежно від ОС
Nodemon	Автоматичний перезапуск сервера	Пришвидшення процесу написання коду
VS Code	Написання та редагування вихідного коду	Зручна навігація, автодоповнення тексту
Postman	Емуляція HTTP-запитів та тест маршрутів	Повне налагодження API без участі фронтенду

Аналізуючи дані Таблиці 2.4, можна стверджувати, що підібраний комплекс допоміжних бібліотек та утиліт повністю закриває всі технічні потреби проєкту. Використання спеціалізованих модулів для обробки графіки та файлів дозволило значно розвантажити сервер, а професійні інструменти тестування допомогли створити стабільну та захищену архітектуру вебзастосунку.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ВЕБСАЙТУ

3.1 Розробка серверної частини на Node.js + Express

3.1.1 Архітектура та файлова структура проєкту

Для забезпечення високої масштабованості коду, зручності подальшої підтримти та чіткого розділення відповідальності, серверну частину вебзастосунку було спроєктовано з використанням архітектурного патерну MVC (Model-View-Controller) із виділенням додаткового сервісного шару [3, 23]. Весь вихідний код проєкту логічно розділений на дві основні базові директорії: public та src.

Директорія public призначена для зберігання статичних ресурсів, до яких клієнтський браузер має прямий доступ без додаткової обробки контролерами. Вона містить:

- Графічні елементи інтерфейсу (логотипи сайту, іконки, фавікони).
- Клієнтські скрипти (JavaScript-файли фронтенду), які відповідають за асинхронну взаємодію із сервером.
- Каталог із медіафайлами користувачів, куди сервер автоматично зберігає всі завантажені картинки та прев'ю до постів.

Директорія src є головним контейнером серверної логіки платформи. Її внутрішня структура відповідає сучасним стандартам розробки на Node.js та містить наступні компоненти:

- app.js – головний вхідний файл застосунку. Саме тут відбувається ініціалізація вебсервера Express, підключення базових проміжних обробників (наприклад, для парсингу JSON та налаштування статичних папок) і запуск прослуховування мережевого порту [15].
- config/ – директорія конфігураційних файлів. Тут зберігаються параметри підключення до бази даних MySQL, налаштування середовища та глобальні константи проєкту [8].

- models/ – шар доступу до даних (Model). Файли у цій папці відповідають за безпосередню взаємодію з таблицями бази даних (дошки, треди, пости) за допомогою SQL-запитів [28].
- controllers/ – шар управління логікою (Controller). Функції-контролери приймають вхідні дані від клієнта, здійснюють їх валідацію, викликають необхідні моделі та формують остаточну HTTP-відповідь.
- routes/ – шар маршрутизації. Файли цієї директорії відповідають за реєстрацію API-ендпоінтів та зв'язування конкретних URL-адрес (наприклад, створення нового поста) із відповідними методами у контролерах [26].
- services/ – шар ізольованої бізнес-логіки. Сюди винесені допоміжні модулі, які виконують ресурсомісткі або специфічні операції, наприклад, стиснення зображень за допомогою бібліотеки Sharp або робота з файловою системою сервера (fs) [27].
- views/ – шар представлення (View). Містить файли шаблонів, які використовуються для генерації HTML-сторінок на боці сервера перед їх відправкою користувачеві [21].

Логічна організація цієї структури та взаємозв’язки між описаними компонентами системи наочно представлені на рисунку 3.1.

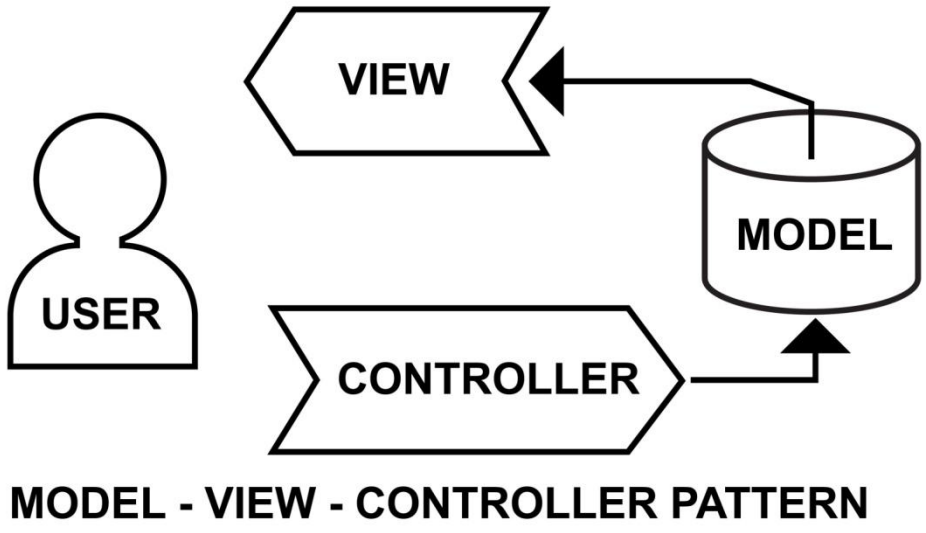

Рисунок 3.1 – Архітектура взаємодії компонентів вебзастосунку

Така багаторівнева файлова структура дозволяє легко масштабувати іміджборд, додавати нові функції та ізолювати помилки в окремих модулях без ризику пошкодити роботу всього застосунку [23].

3.1.2 Ініціалізація сервера та налаштування проміжного програмного забезпечення

Основним ядром застосунку є файл app.js, у якому відбувається ініціалізація вебсервера на базі фреймворку Express. Важливим етапом налаштування сервера є підключення проміжного програмного забезпечення (middleware) – спеціальних функцій, які перехоплюють, аналізують та модифікують усі вхідні HTTP-запити до того, як вони потраплять до основних контролерів маршрутизації [26].

Для забезпечення коректної роботи іміджборду було налаштовано наступні ключові модулі:

- Модуль CORS (Cross-Origin Resource Sharing). З міркувань безпеки сучасні браузери блокують запити до сервера, якщо клієнтська та серверна частини знаходяться на різних доменах або портах (політика єдиного походження – Same-Origin Policy) [31]. Підключення middleware cors дозволяє гнучко налаштувати заголовки HTTP-відповідей, надаючи дозвіл фронтенду безперешкодно обмінюватися даними з нашим API.
- Парсери вхідних даних (express.json та express.urlencoded). Оскільки взаємодія між клієнтом та сервером відбувається у форматі JSON (JavaScript Object Notation), сервер повинен вміти десеріалізувати вхідні потоки даних [25]. Метод express.json() автоматично перетворює тіло запиту (наприклад, текст нового поста) у зручний об'єкт JavaScript. Додатково використовується express.urlencoded() для розпізнавання даних, відправлених через стандартні HTML-форми.
- Маршрутизація статичних файлів (express.static). Іміджборд передбачає постійне завантаження та відображення медіаконтенту. Замість того, щоб писати

окремі обробники для кожної картинки чи скрипта, було використано вбудований метод express.static(). Він вказує серверу директорію public як публічно доступну. Завдяки цьому браузер користувача може напряму отримувати доступ до завантажених зображень, логотипів та CSS-стилів за прямими посиланнями.

В лістингу 3.1 наведено базовий код ініціалізації сервера у файлі app.js.

Лістинг 3.1 – Базовий код ініціалізації сервера

```
const express = require('express');
     const cors = require('cors');
     const path = require('path');

     // Ініціалізація застосунку
     const app = express();

     // Підключення проміжного програмного забезпечення
     app.use(cors());
     app.use(express.json());
     app.use(express.urlencoded({ extended: true }));

     // Налаштування доступу до статичних файлів у папці public
     app.use(express.static(path.join(__dirname, '../public')));

     // Запуск прослуховування порту
     const PORT = process.env.PORT || 3000;
     app.listen(PORT, () => {
         console.log(`Сервер іміджборду успішно запущено на порту
     ${PORT}`);
     });
```

Така конфігурація є оптимальною для RESTful API [11]. Вона створює захищене, швидке та гнучке середовище, готове до обробки асинхронних запитів та роботи з файловою системою застосунку.

3.1.3 Логіка обробки запитів та маршрутизація (Routing)

Уся навігація та обробка дій користувачів на іміджборді керується системою маршрутизації (Router) фреймворку Express [16]. Замість того, щоб описувати всі можливі шляхи в одному громіздкому файлі, логіку було розділено

на окремі модулі за їхнім функціональним призначенням. Це робить код читабельним та полегшує його підтримку.

Головний файл app.js виступає диспетчером, який перенаправляє запити до відповідних роутерів. Крім того, на рівні диспетчера підключено шаблонізатор EJS (Embedded JavaScript). Це дозволяє серверу динамічно генерувати HTML-сторінки, вставляючи дані з бази (наприклад, список постів) безпосередньо в шаблон перед відправкою користувачеві.

Внутрішня архітектура маршрутів поділена на два основні файли:

- Маршрути головної сторінки (homeRoutes.js). Цей модуль відповідає за єдину точку входу на сайт. Він обробляє звичайний GET-запит за кореневою адресою. Коли користувач заходить на сайт, роутер викликає метод renderHomePage з відповідного контролера, який компілює EJS-шаблон вітальної сторінки з переліком доступних дощок.
- Маршрути дощок та тредів (boardRoutes.js). Це основний модуль, який містить усю бізнес-логіку. Тут застосовано механізм динамічних параметрів URL. Замість створення окремих маршрутів для кожної дошки, використовується універсальний шлях /:boardCode. Сервер автоматично зчитує цю частину адреси як змінну, що дозволяє одному контролеру renderBoardPage обслуговувати нескінченну кількість різних дощок, спираючись на дані з бази [26].

Ці два модулі виконують основну роботу сервера. Один відповідає за головну сторінку, інший за дошку та треди.

У цьому ж модулі налаштовано чітке розділення за HTTP-методами. GET-запити використовуються виключно для отримання даних та відображення сторінок (відкриття списку тредів на дошці або читання конкретного треду через параметр /:threadId).

POST-запити використовуються для створення нового контенту (публікація нового треду або відправка повідомлення-відповіді).

Особливістю маршрутів публікації є вбудована інтеграція з модулем multer. Для маршрутів створення треду та відповіді (POST /:boardCode/thread та

POST /:boardCode/thread/:threadId/reply) перед запуском основного контролера викликається проміжний обробник upload.array('media', 4). Ця функція автоматично перехоплює до чотирьох медіафайлів із форми відправки, зберігає їх у директорію public/uploads/ і лише після цього передає керування контролеру. Це гарантує, що бізнес-логіка працюватиме з уже збереженими файлами, уникаючи помилок під час запису на диск.

3.1.4 Алгоритм обробки та публікації повідомлень

Процес публікації контенту в системі побудований за принципом каскадної обробки даних [23], що забезпечує стабільність роботи навіть при високих навантаженнях. Коли користувач ініціює запит на створення треду або відповіді через інтерфейс, сервер спочатку виконує етап первинної валідації. На цьому рівні контролер createThread або createReply перевіряє наявність обов'язкових даних: якщо це створення нової теми, система суворо вимагає наявності прикріпленого медіафайлу, тоді як для відповіді допустимим є або текстовий контент, або файл. Така фільтрація на ранньому етапі дозволяє відсікати «сміттєві» запити без звернення до бази даних, що суттєво економить обчислювальні ресурси.

Після проходження валідації запит передається до сервісного шару threadService, де виконується основна бізнес-логіка. Формування даних починається з операції вставки в таблицю threads або posts. Важливою особливістю реалізації є отримання ідентифікатора (insertId) щойно створеного запису, який стає ключовим елементом для подальшої прив’язки медіафайлів. Саме цей механізм дозволяє уникнути розсинхронізації даних та забезпечити цілісність зв’язків між сутностями. Якщо користувач додав файли, система входить у цикл обробки, де кожен елемент масиву files проходить через процес реєстрації в БД.

Приклад реалізації даного алгоритму наведено в лістингу 3.2.

Лістинг 3.2 – Реалізація методу створення треду в threadService.js

```
const express = require('express');
     const cors = require('cors');
     const path = require('path');
async function createThreadWithPost(boardCode, body, files) {
     const { text, author_name, subject } = body;
     const author = author_name || 'Анон';

     // Створення запису в БД та отримання ідентифікатора
     const [threadResult] = await db.query('INSERT INTO threads
(board_id, subject) VALUES (?, ?)', [boardCode, subject || '']);
     const threadId = threadResult.insertId;

     const [postResult] = await db.query('INSERT INTO posts
(thread_id, author_name, text) VALUES (?, ?, ?)', [threadId, author,
text]);
     const postId = postResult.insertId;

     // Циклічна обробка файлів та їх перейменування
     if (files && files.length > 0) {
         for (const file of files) {
             const ext = path.extname(file.originalname);
             const [fileResult] = await db.query('INSERT INTO files
(post_id, extension) VALUES (?, ?)', [postId, ext]);
             const newPath = path.join(file.destination,
`${fileResult.insertId}${ext}`);
             fs.renameSync(file.path, newPath);
         }
     }
     return threadId;
     }
```

Особливе місце в алгоритмі посідає обробка та структурування медіафайлів. Для кожного файлу, завантаженого користувачем, система автоматично генерує унікальне ім'я на основі його ідентифікатора в базі, після чого модуль fs здійснює фізичне перейменування та переміщення файлу у постійну директорію зберігання. Для забезпечення коректного відображення контенту на сторінці, сервер додатково використовує бібліотеку probe-image-size. Замість повного завантаження файлу в оперативну пам'ять, цей інструмент через потокову передачу зчитує лише заголовки зображення, видобуваючи метадані щодо його роздільної здатності [17]..

На фінальному етапі публікації система формує об'єкт відповіді у форматі JSON, який містить посилання для перенаправлення користувача на створений

тред. Така реалізація, що базується на асинхронному виконанні запитів Promise.all, дозволяє серверу паралельно збирати дані про пости, їхні медіа-атрибути, технічні параметри (вагу та розміри) і збирати все в єдиний цілісний об'єкт перед відправкою клієнту. Завдяки такому підходу інтерфейс сайту отримує готовий до відображення контент миттєво, що відповідає вимогам швидкої та інтерактивної роботи сучасних вебзастосунків.

3.1.5 Управління медіаконтентом та робота з файловою системою

Окрім створення контенту, серверна частина забезпечує повний життєвий цикл медіафайлів, включаючи їх безпечне видалення. Використання модуля fs (File System) дозволяє системі підтримувати чистоту дискового простору, що є критично важливим для хостингу іміджборду, де обсяг завантажених зображень може швидко зростати [27]..

Архітектурно робота з файлами реалізована через синхронізацію стану бази даних та файлового сховища. Коли виникає потреба у видаленні поста (наприклад, за ініціативою автора або модератора), сервер не просто видаляє запис із таблиці posts, а спочатку виконує операцію пошуку всіх асоційованих файлів у таблиці files. За допомогою модуля path формується точний шлях до медіаконтенту на диску, що гарантує кросплатформеність та коректну роботу незалежно від операційної системи сервера.

Фізичне видалення виконується лише після успішної верифікації існування файлу на диску через метод fs.existsSync. Це запобігає виникненню помилок у роботі застосунку, якщо файл був видалений раніше або був пошкоджений. Такий підхід забезпечує ідеальну узгодженість між структурованими даними в MySQL та неструктурованим контентом у папці public/uploads.

Приклад реалізації функції видалення поста з одночасним очищенням файлової системи наведено нижче (див. лістинг 3.3):

Лістинг 3.3 – Реалізація видалення поста та пов'язаних файлів

```
async function deletePost(postId) {
    // 1. Отримуємо список файлів, що належать посту
    const [files] = await db.query('SELECT id, extension FROM files
WHERE post_id = ?', [postId]);

    // 2. Фізично видаляємо кожен файл з папки uploads
    for (const file of files) {
        const filePath = path.join(__dirname,
'../../public/uploads', `${file.id}${file.extension}`);
        if (fs.existsSync(filePath)) {
            fs.unlinkSync(filePath);
        }
    }

    // 3. Видаляємо записи про файли та сам пост з бази даних
    await db.query('DELETE FROM files WHERE post_id = ?', [postId]);
    await db.query('DELETE FROM posts WHERE id = ?', [postId]);
}
```

Як видно з Лістингу 3.2, алгоритм передбачає попереднє зчитування метаданих про файли. Це критично важливий крок: якщо ми спочатку видалимо запис із бази, ми втратимо інформацію про імена файлів і не зможемо їх знайти на диску, що призведе до накопичення «сирітських» файлів, які нікому не належать, але займають дисковий простір [29].

3.2 Розробка функціоналу клієнтської частини (фронтенду) за допомогою HTML, CSS та JavaScript

3.2.1 Архітектура клієнтської частини та шаблонізація (EJS)

Клієнтська архітектура іміджборду базується на концепції серверного рендерингу (Server-Side Rendering, SSR), що передбачає виконання більшої частини логіки формування сторінок на стороні сервера [14]. Завдяки цьому браузер користувача отримує вже готовий до відображення HTML-код. Такий підхід суттєво пришвидшує початкове завантаження сторінки та забезпечує кращу індексацію контенту пошуковими системами, оскільки клієнтському

пристрою не потрібно виконувати додаткові складні обчислення чи завантажувати важкі JavaScript-фреймворки, такі як React або Angular [18].

Основним інструментом для реалізації цієї стратегії став шаблонізатор EJS (Embedded JavaScript templates) [20]. Його інтеграція дозволила ефективно розділити структуру HTML-документа та дані, що динамічно надходять із бази даних MySQL [28]. Вибір EJS зумовлений його простотою та високою швидкістю генерації: шаблони компілюються безпосередньо під час обробки запиту, завдяки чому сервер миттєво віддає готовий HTML-код після отримання даних з БД.

Гнучкість вбудовування логіки є ще однією вагомою перевагою EJS, оскільки він дозволяє використовувати звичайний синтаксис JavaScript безпосередньо всередині HTML-розмітки. Це дало можливість легко реалізувати цикли для виведення списку всіх постів у треді та умовні конструкції, як-от приховування форми, якщо тред закритий, без винесення складної логіки на клієнтську сторону. Приклад такої інтеграції логіки в шаблон наведено нижче (див. лістинг 3.4):

Лістинг 3.4 – Цикл створення об’єктів HTML за допомогою JavaScript

```
<% posts.forEach(post => { %>
     <div class="post">
          <p class="content"><%= post.text %></p>
          <% if (post.file) { %>
          <img src="/uploads/<%= post.file %>" alt="media">
          <% } %>
     </div>
<% }); %>
```

Крім того, такий підхід забезпечує мінімалістичність інтерфейсу, адже дозволив обмежитися мінімальною кількістю стилів, інтегрованих безпосередньо в шаблони [22]. Це виключає зайві запити до сервера за зовнішні файлами та додатково пришвидшує роботу. Механізм роботи з даними при цьому виглядає наступним чином: під час запиту клієнта контролер отримує масив об'єктів з бази даних і передає його до шаблону EJS. Шаблонізатор виконує ітерацію по цьому

масиву, підставляючи значення – текст поста, ім'я автора чи посилання на медіафайли – у підготовлені HTML-теги [21].

У підсумку, обрана архітектура дозволила зосередитися на функціональній надійності бекенду, зробивши фронтенд легким, надійним та зручним для подальшого масштабування. Вона створює оптимальний баланс між швидкістю відгуку та простотою підтримки коду [13].

3.2.2 Асинхронна взаємодія з API (використання Fetch)

Для реалізації інтерактивної взаємодії між клієнтом та сервером у проєкті було використано сучасний стандарт API – fetch. Ця технологія дозволяє виконувати мережеві запити асинхронно, тобто у фоновому режимі, без необхідності повного перезавантаження сторінки під час кожної дії користувача [17]. Це суттєво покращує загальний користувацький досвід (UX), оскільки інтерфейс залишається доступним для взаємодії навіть у момент обробки даних на сервері, а користувач не втрачає поточну позицію на сторінці після відправки повідомлення.

Механізм роботи fetch у проєкті базується на асинхронних функціях async/await, що дозволяє писати чистий та зрозумілий код для обробки мережевих операцій [24]. Коли користувач ініціює відправку форми, клієнтський скрипт перехоплює подію submit, зупиняє стандартну поведінку браузера і збирає дані у форматі FormData. Далі ці дані передаються на сервер через POST-запит, де вони обробляються бекендом, а у відповідь сервер повертає JSON-об'єкт із результатом операції [25]. Технічна реалізація цього процесу, що включає перехоплення стандартної події відправки форми та виконання асинхронного запиту до API, детально представлена у Лістингу 3.5.

Лістинг 3.5 – Асинхронна відправка даних через Fetch API

```
<% posts.forEach(post => { %>
     <div class="post">
          <p class="content"><%= post.text %></p>
          <% if (post.file) { %>
          <img src="/uploads/<%= post.file %>" alt="media">
          <% } %>
     </div>
<% }); %>
```

Використання такого підходу дозволяє забезпечити стабільний зв'язок між частинами системи. Оскільки сервер завжди повертає структуровані дані у форматі JSON, клієнтська частина отримує повний контроль над тим, як саме відобразити результат [25]. Це робить застосунок набагато гнучкішим порівняно зі стандартними методами відправки HTML-форм, де кожен крок користувача супроводжувався б «миготінням» екрана при перезавантаженні сторінки.

3.2.3 Реалізація динамічних оновлень (AJAX-логіка)

Для підвищення інтерактивності інтерфейсу іміджборду було впроваджено механізм динамічної обробки подій через AJAX. Це рішення дозволяє уникнути класичного «миготіння» сторінки при кожній дії користувача та забезпечує миттєвий зворотний зв’язок [18]. Замість стандартної синхронної відправки HTML-форми, клієнтський скрипт перехоплює подію надсилання, валідує дані та передає їх на сервер у фоновому режимі.

Технічна реалізація цього процесу передбачає використання універсального обробника, який автоматично знаходить форми з відповідним класом (.ajax-form). Такий підхід забезпечує гнучкість: достатньо додати цей клас до нової форми, і вона автоматично стає «інтерактивною». У Лістингу 3.6 наведено повний код клієнтського обробника, який включає валідацію, управління станом інтерфейсу та обробку мережевих помилок.

Лістинг 3.6 – Універсальна обробка AJAX-форм через Fetch API

```
document.addEventListener('DOMContentLoaded', () => {
```

```
    const forms = document.querySelectorAll('.ajax-form');

    forms.forEach(form => {
        form.addEventListener('submit', async (e) => {
            e.preventDefault();

            const                    submitBtn                    =
form.querySelector('button[type="submit"]');
            const originalBtnText = submitBtn.innerText;

            // Блокуємо кнопку для запобігання дублюванню запитів
            submitBtn.disabled = true;
            submitBtn.innerText = 'Завантаження...';

            const formData = new FormData(form);

            try {
                const response = await fetch(form.action, {
                    method: 'POST',
                    body: formData
                });

                const result = await response.json();

                if (result.success) {
                    window.location.href = result.redirectUrl;
                } else {
                    alert('Сталася помилка: ' + (result.message ||
'Невідома помилка'));
                    submitBtn.disabled = false;
                    submitBtn.innerText = originalBtnText;
                }
            } catch (err) {
                console.error(err);
                alert('Помилка фронтенду: ' + err.message);
                submitBtn.disabled = false;
                submitBtn.innerText = originalBtnText;
            }
        });
    });
});
```

Як видно з коду, використання FormData дозволяє автоматично збирати дані всіх полів форми, включаючи медіафайли, що суттєво спрощує передачу контенту на сервер. Важливим елементом є також обробка винятків (try...catch), яка захищає застосунок від непередбачуваних помилок мережі, гарантуючи, що користувач отримає зрозуміле повідомлення замість простого зависання інтерфейсу.

Ця реалізація дозволяє серверу оперувати лише структурованими даними (JSON), що значно зменшує час відгуку системи. В результаті інтерфейс стає «живим» і чуйним, а навантаження на мережу мінімізується, що є критично важливим для проєктів, де користувачі активно завантажують контент.

3.3 Взаємодія сервера з MySQL: написання запитів для створення таблиць та управління даними

3.3.1 Проєктування структури даних

Архітектура бази даних базується на принципах нормалізації, що мінімізує надмірність інформації та запобігає аномаліям оновлення [29]. Схема взаємозв'язків між основними сутностями (дошки, треди, пости та файли) представлена на рисунку 3.2.

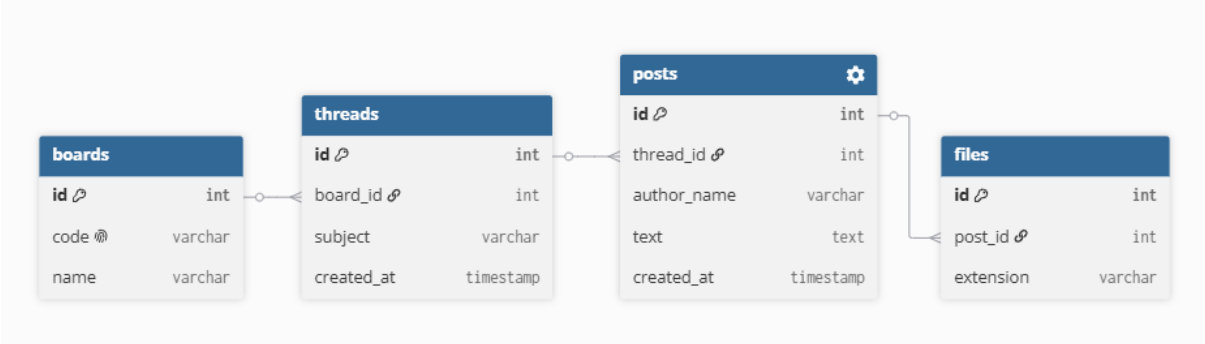

Рисунок 3.2 – Схема зв'язків бази даних (ER-діаграма)

Така ієрархічна структура дозволяє чітко розмежувати відповідальність: кожна таблиця зберігає унікальний тип даних, а зв'язки Foreign Key гарантують, що при видаленні треду система коректно обробляє пов'язані з ним пости та медіафайли [28]. Це робить проєкт гнучким для подальшого розширення – наприклад, при додаванні функцій користувачів або системи рейтингу достатньо буде створити нову таблицю та пов'язати її з існуючою структурою через user_id.

3.3.2 Оптимізація запитів до бази даних

Одним із ключових завдань при розробці було забезпечення високої швидкості вибірки даних для відображення тредів. Ми уникаємо надлишкових запитів, застосовуючи принцип "все в одному", що реалізовано через оператор LEFT JOIN [10].

Окрім використання JOIN-ів, для забезпечення швидкодії системи було налаштовано індексацію ключових полів (таких як thread_id та board_id) [28]. Завдяки індексації, СУБД знаходить потрібні пости миттєво, навіть якщо кількість записів у таблиці обчислюється тисячами. Це суттєво знижує навантаження на диск і процесор сервера.

Для вибірки даних із бази даних було реалізовано оптимізований SQL-запит. Його реалізація, що дозволяє отримати повний контекст поста (включаючи прикріплені файли) за одну операцію, наведена у лістингу 3.7.

Лістинг 3.7 – SQL-запит для отримання даних треду

```
SELECT posts.*, files.id AS file_id, files.extension
     FROM posts
     LEFT JOIN files ON posts.id = files.post_id
     WHERE posts.thread_id = ?
     ORDER BY posts.created_at ASC;
```

Використання цього запиту дозволяє серверу отримати повний контекст поста – включаючи наявність файлів – за одну транзакцію [9]. Такий підхід мінімізує Latency (час затримки) і гарантує, що сторінка треду завантажиться стабільно швидко незалежно від активності користувачів на інших дошках [7].

3.4 Особливості взаємодії користувача з інтерфейсом

У цьому розділі наведено огляд реалізованого вебзастосунку. Дизайн системи базується на принципах мінімалізму, що дозволяє користувачу зосередитися на контенті, не витрачаючи час на вивчення складних навігаційних елементів.

3.4.1 Головна сторінка

Головна сторінка є точкою входу, де реалізована чітка навігація між тематичними розділами (дошками), що дозволяє користувачу миттєво знайти цікаву йому категорію контенту. Візуальне представлення інтерфейсу головної сторінки з навігаційною панеллю наведено на рисунку 3.3.

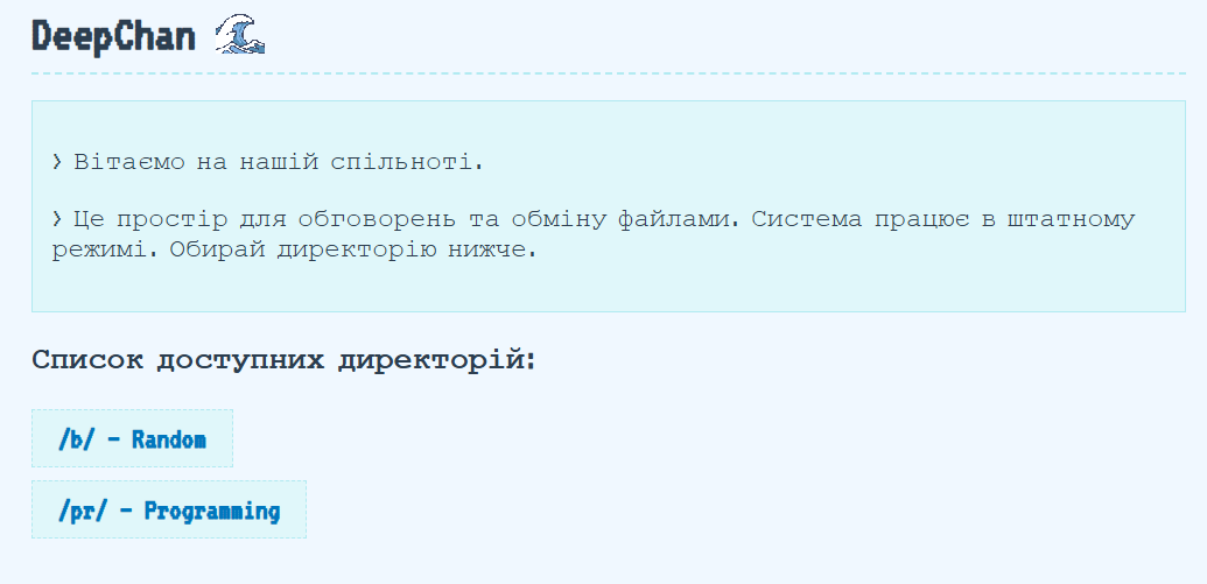

Рисунок 3.3 – Головна сторінка з навігацією за категоріями

Як показано на рисунку 3.3, інтерфейс забезпечує швидкий доступ до двох основних дощок: Random, призначеної для спілкування на будь-які теми, та Programming, яка є вузькоспеціалізованою і зосереджена на обговоренні технологічних питань. Така структура категорій дозволяє ефективно розділяти

потоки інформації, забезпечуючи зручність навігації та впорядкованість дискусій на ресурсі, що сприяє утриманню аудиторії [1].

3.4.2 Сторінка дошки

При переході на дошку користувач бачить перелік актуальних тредів. Було зосереджено увагу на тому, щоб заголовки тредів та їх короткий опис були читабельними, а список — структурованим [21]. Це забезпечує швидкий візуальний пошук потрібної дискусії. Візуальне представлення інтерфейсу дошки зі списком активних тредів наведено на рисунку 3.4.

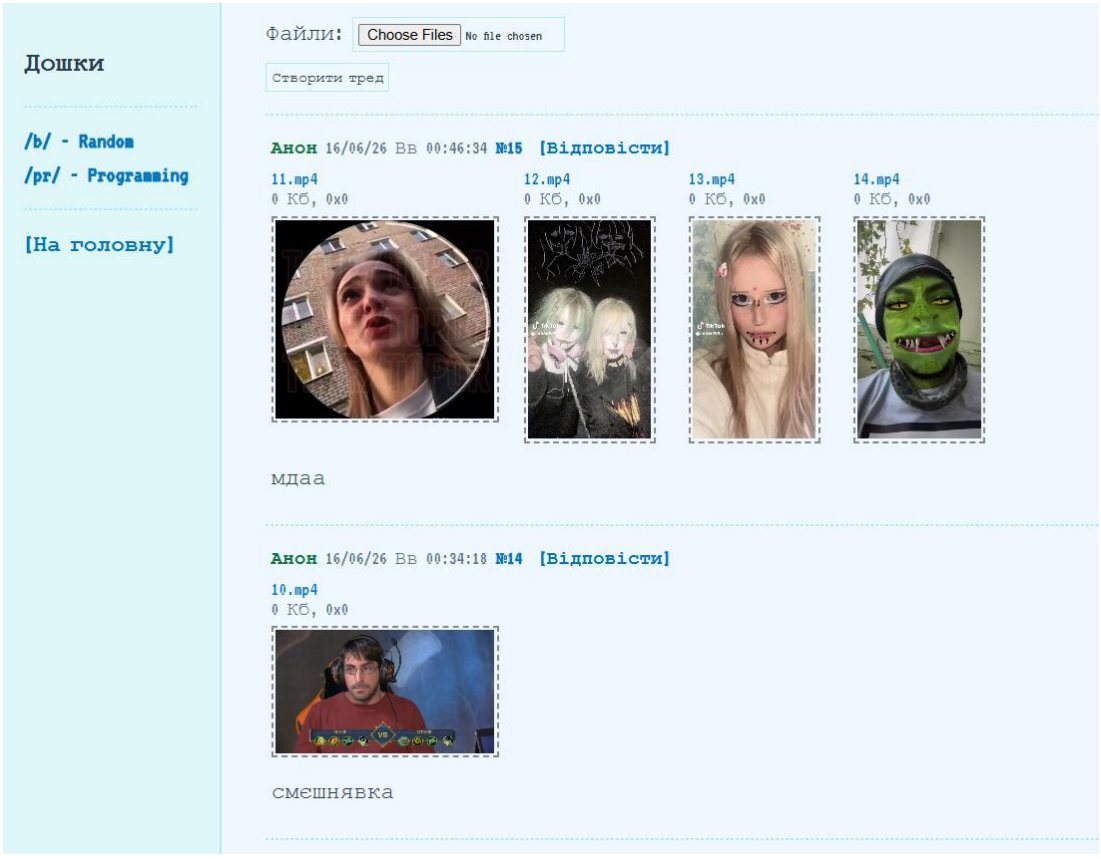

Рисунок 3.4 – Інтерфейс дошки з тредами

Як можна побачити на рисунку 3.4, структура сторінки дозволяє користувачу швидко оцінити вміст дискусій. При натисканні на кнопку «Відповісти» або заголовок конкретного треду система перенаправляє

користувача до детального перегляду всієї гілки обговорення, де відображаються всі наявні пости. Така навігація значно спрощує взаємодію з контентом та дозволяє користувачу миттєво долучитися до цікавої йому дискусії, що є критично важливим елементом ергономіки та зручності інтерфейсу [18].

3.4.3 Сторінка треду

Центральним елементом взаємодії є форма публікації. Вона спроєктована таким чином, щоб користувач міг додати як текстове повідомлення, так і прикріпити медіафайл за лічені секунди. Завдяки AJAX-логіці, описаній раніше, публікація відбувається без перезавантаження сторінки, що забезпечує миттєвий зворотний зв'язок після натискання кнопки відправки.

Інтерфейс цієї форми та приклад її функціонування в контексті перегляду треду наведено на рисунку 3.5.

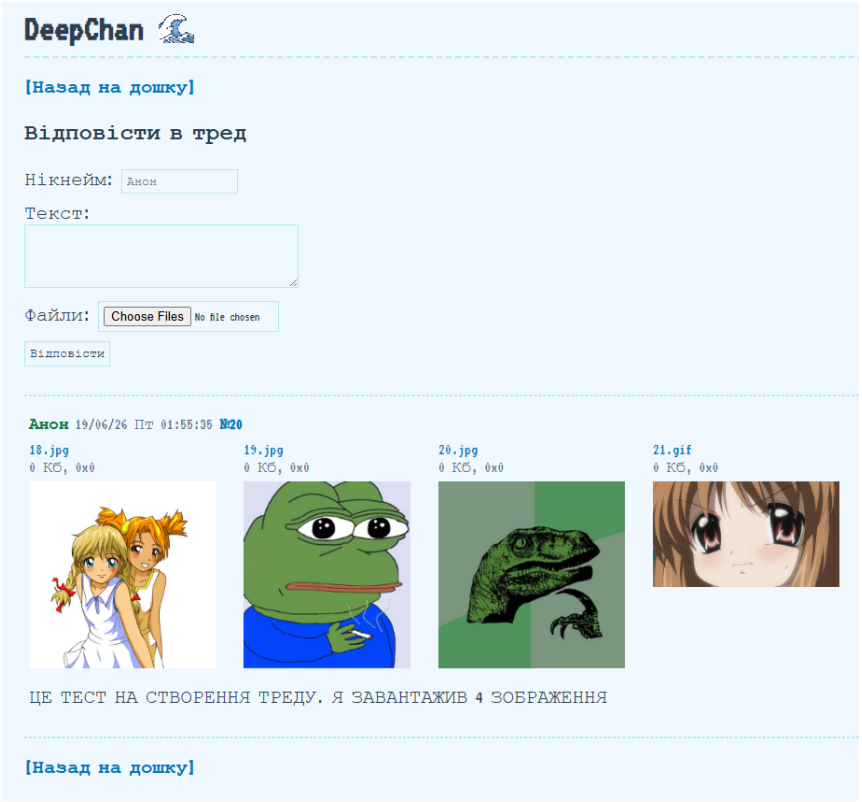

Рисунок 3.5 – Інтерфейс треду

На рисунку 3.5 зображено тред, в якому лише один пост з 4 зображеннями. Зверху від нього знаходиться поле для додавання постів. Якщо ввести текст у відповідне поле, вибрати файл (опціонально) і натиснути кнопку створення поста, то наша відповідь автоматично появиться в даному треді, і ми побачимо її без оновлення сторінки, завдяки динамічному оновленню через JavaScript [24].

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Аналіз психосоціальних ризиків та інформаційної безпеки користувачів анонімного веб-ресурсу

Специфіка анонімних іміджбордів полягає у відсутності реєстрації та публічних профілів [1]. Така архітектура діє як двосічний меч. З одного боку, відсутність рейтингів і лайків нівелює соціальний статус: спільнота оцінює виключно контент, а не його автора [1, 2]. Це гарантує приватність і дозволяє безпечно обговорювати табуйовані теми чи шукати психологічну підтримку без страху осуду в реальному житті [1]. З іншого боку, абсолютна анонімність неминуче створює специфічне цифрове середовище з підвищеними психосоціальними ризиками та серйозними загрозами інформаційній безпеці як для користувачів, так і для спільноти загалом.

4.1.1 Психосоціальні виклики

Відсутність ідентифікації послаблює моральні обмеження користувачів, перетворюючи іміджборди на сприятливе середовище для кібербулінгу, дезінформації та агресії [2, 32]. Через брак цензури там швидко формуються «ехо-камери», де радикальні ідеї безперешкодно посилюються, а мова ворожнечі стає субкультурною нормою [2]. Найбільша загроза полягає у «гейміфікації» та романтизації насильства, що здатне спровокувати учасників на реальні злочини заради визнання на форумі. Водночас відсутність премодерації наражає відвідувачів на раптовий вплив шок-контенту [31]. Це спричиняє не лише гострі психологічні травми, а й небезпечну довгострокову десенсибілізацію: психіка поступово звикає до жахів, що призводить до втрати емпатії та повного стирання моральних кордонів [32].

Попри декларовану приватність, користувачі іміджбордів є вразливими до деанонімізації. Викриття реальної особистості в такому середовищі часто перетворюється на «цифрове полювання», де головними інструментами стають аналіз відкритих джерел (OSINT) та соціальна інженерія [31]. Користувачі самі, навіть не усвідомлюючи цього, залишають у тредах дрібні підказки: специфічний сленг, часові проміжки активності, випадкові деталі на фотографіях або згадки локальних подій. Зловмисники збирають ці крупиці інформації докупи (цей процес називається доксингом), що дозволяє їм крок за кроком вирахувати реальну людину. Найбільша небезпека полягає в тому, що стати жертвою цькування можна абсолютно безпідставно – просто через незгоду з думкою більшості або заради абсурдного жарту спільноти (так званих «лулзів»). Якщо користувач викликає роздратування, сотні інших анонімів можуть скоординуватися, щоб знайти його реальні профілі в соцмережах, телефон чи адресу. Це переростає у масштабний терор у реальному житті: погрози, неправдиві виклики екстрених служб (сваттінг) або фейкові скарги за місцем навчання чи роботи з метою повністю зруйнувати життя людини.

Водночас цей самий механізм колективного пошуку має і зворотну, позитивну сторону, коли іміджборд перетворюється на інструмент стихійного інтернет-правосуддя. Відомо чимало прикладів, коли саме анонімні спільноти об'єднували зусилля для виявлення небезпечних злочинців – педофілів, шкуродерів чи шахраїв, які вважали себе невловимими. Тисячі користувачів паралельно аналізують кожну секунду опублікованих зловмисниками кадрів: вираховують геолокацію за формою гір чи типом дерев на фоні, розпізнають відображення в зіницях чи дзеркалах, визначають модель розетки чи побутової техніки в кімнаті. Така гігантська аналітична робота часто дозволяє встановити точні координати злочинця, після чого зібрані докази передаються правоохоронним органам для здійснення реальних арештів.

4.1.2 Технічні загрози та архітектурні методи захисту

Технічна деанонімізація користувачів найчастіше відбувається через витік прихованих EXIF-даних із цифрових зображень (GPS-координати, модель пристрою, час) або через фішингові посилання та шкідливі скрипти, що дозволяють перехопити IP-адресу. Оскільки повністю усунути людський фактор неможливо, головне завдання розробника полягає у створенні надійної серверної архітектури, яка автоматично перекриває шляхи до викриття особи [32]. Для цього впроваджується обов'язкове програмне очищення EXIF-даних під час завантаження файлів, а також безпечне криптографічне хешування IP-адрес із додаванням «солі» та швидким видаленням логів. Своєю чергою, для мінімізації поширення деструктивного шок-контенту застосовуються обмеження частоти публікацій (Rate Limiting) та інтегрується система користувацьких скарг, що дає змогу модераторам оперативно реагувати на загрози.

4.2 Аналіз небезпек та заходи електробезпеки при обслуговуванні серверного обладнання

4.2.1 Основні електричні небезпеки

Під час обслуговування серверного обладнання персонал постійно наражається на комплекс серйозних небезпек [31]. Найбільшою загрозою є ризик ураження струмом від промислової мережі або від високовольтних конденсаторів, які зберігають смертельний заряд навіть після повного відключення серверів від розетки [36]. Через високу щільність монтажу випадкове падіння інструментів чи зношення ізоляції здатне миттєво спричинити коротке замикання та утворення руйнівної електричної дуги з температурою в кілька тисяч градусів. Крім того, перевантаження ліній або слабкі контакти викликають перегрівання, яке завдяки безперервній роботі потужних серверних вентиляторів за лічені секунди роздмухується у масштабну

пожежу. Окрему небезпеку становить статична електрика, чиї мікророзряди (ESD) безповоротно пропалюють мікросхеми, а несподіваний удар струмом викликає у працівників рефлекторні рухи, що призводять до механічних травм об гострі краї стійок [35]. Нарешті, виконання «гарячої» заміни компонентів (Hot-Plug) без захисних рукавичок загрожує важкими термічними опіками від радіаторів

4.2.2 Комплексні заходи електробезпеки

Під час обслуговування серверного обладнання персонал постійно наражається на комплекс серйозних небезпек [31]. Найбільшою загрозою є ризик ураження струмом від промислової мережі або від високовольтних конденсаторів, які зберігають смертельний заряд навіть після повного відключення серверів від розетки [37]. Через високу щільність монтажу випадкове падіння інструментів чи зношення ізоляції здатне миттєво спричинити коротке замикання та утворення руйнівної електричної дуги з температурою в кілька тисяч градусів. Крім того, перевантаження ліній або слабкі контакти викликають перегрівання, яке завдяки безперервній роботі потужних серверних вентиляторів за лічені секунди роздмухується у масштабну пожежу. Окрему небезпеку становить статична електрика, чиї мікророзряди (ESD) безповоротно пропалюють мікросхеми, а несподіваний удар струмом викликає у працівників рефлекторні рухи, що призводять до механічних травм об гострі краї стійок [35]. Нарешті, виконання «гарячої» заміни компонентів (Hot-Plug) без захисних рукавичок загрожує важкими термічними опіками від радіаторів

ВИСНОВКИ

У кваліфікаційній роботі спроєктовано та реалізовано вебзастосунок анонімного іміджборду, який забезпечує високу швидкодію та стабільну обробку медіаконтенту.

У першому розділі кваліфікаційної роботи освітнього рівня «Бакалавр»:

- Подано аналіз технологій створення дискусійних платформ.
- Розглянуто та обґрунтовано вибір стека Node.js, Express.js та MySQL.
- Проаналізовано методи оптимізації роботи серверної частини.

У другому розділі кваліфікаційної роботи:

- Досліджено принципи побудови трирівневої клієнт-серверної архітектури.
- Обґрунтовано застосування архітектурного патерну MVC.
- Сформовано оптимальну структуру та схему зв’язків бази даних.

У третьому розділі кваліфікаційної роботи:

- Розроблено функціонал для створення тредів та публікації контенту.
- Запропоновано використання AJAX для динамічного оновлення без перезавантаження.
- Протестовано роботу API, що підтвердило стабільність системи.

У розділі «Безпека життєдіяльності, основи охорони праці» висвітлено ключові вимоги щодо створення безпечних та ергономічних умов праці розробника при роботі з комп’ютерною технікою, а також проаналізовано заходи екологічної безпеки та захисту від ураження електричним струмом.

ПЕРЕЛІК ДЖЕРЕЛ

1 Mozdeika L. Playing with misinformation, lying with truth: satirical conspiracy theories and sacred seriousness of play in online imageboard cultures // Continuum. 2024. Vol. 38, No. 2. P. 270–282. DOI: https://doi.org/10.1080/10304312.2024.2354241.

2 Sardá Thais et al. Understanding Online Anonymity // Media, Culture & Society. 2019. Vol. 41, No. 4. P. 557–564. DOI: https://doi.org/10.1177/0163443719842074.

3 Ковалевський С. В., Олійник О. М. Проєктування високонавантажених веб-систем на основі асинхронної архітектури // Вісник технічного університету. 2023. Т. 14, № 2. С. 112–119.

4 Семчишин П. М. Архітектурні рішення для розробки веб-застосунків // Матеріали XIV Міжнародної науково-технічної конференції молодих учених та студентів «Актуальні задачі сучасних технологій». 2025. С. 340–342.

5 Третяк М. В. Розробка архітектури веб-сервера із застосуванням Node.js та NestJS Framework : кваліфікаційна робота другого (магістерського) рівня вищої освіти за спеціальністю 126 «Інформаційні системи та технології» / ЛНУП. Львів, 2026.

6 Sharma S., Singh P., Sain J., Shrivastava V., Pandey A. Modern Backend Development Technologies: A Comparative Review and Case Study // Emerging Trends in Expert Applications and Security. ICE-TEAS 2024. Lecture Notes in Networks and Systems. Vol. 1030. Springer, Singapore, 2024. DOI: https://doi.org/10.1007/978-981-97-3745-1_12.

7 Boicea A., Radulescu F., Agapin L. I. MongoDB vs Oracle — Database Comparison // 2012 Third International Conference on Emerging Intelligent Data and Web Technologies. Bucharest, 2012. P. 330–335. DOI: 10.1109/EIDWT.2012.32.

8 MySQL 8.0 Reference Manual : official site / Oracle Corporation // MySQL. URL: https://dev.mysql.com/doc/refman/8.0/en/ (дата звернення: 21.06.2026).

9 Ткаченко А. П. Оптимізація взаємодії веб-серверів Node.js з реляційними базами даних MySQL // Сучасні комп'ютерні технології. 2024. № 1. С. 88–94.

10 Forta B. SQL in 10 Minutes, Sams Teach Yourself // Sams Publishing. Indianapolis, 2014. 288 p.

11 Shakirat S. Client-Server Model Architecture and Core Principles // ResearchGate. 2015. 12 p. URL: https://www.researchgate.net/publication/271295146 (дата звернення: 21.06.2026).

12 Rajdev Sapan. A Simple Guide to MVVM Architecture // Medium. 2025. URL: https://medium.com/@sapanrajdev/a-simple-guide-to-mvvm-architecture (дата звернення: 23.06.2026).

13 Желібо Є. П. Безпека життєдіяльності : підручник / Є. П. Же Зацарний. – Київ : Каравела, 2009. – 344 с.

14 Verma Dhruv. A comparison of web framework efficiency: performance and network analysis of modern web frameworks // [Електронний ресурс] : магістерська робота. 2022.

15 Node.js Documentation // Node.js Foundation. URL: https://nodejs.org/docs/ (дата звернення: 23.06.2026).

16 Express.js Official Documentation // Express.js Foundation. URL: https://expressjs.com/ (дата звернення: 23.06.2026).

17 Myers Daniel S. et al. MapJAX: Data Structure Abstractions for Asynchronous Web Applications // USENIX Annual Technical Conference. 2007. P. 1–14.

18 Хлиста І. Вирішення проблеми часткового оновлення вмісту веб-сторінки шляхом оптимізації параметрів SPA // Комп’ютерне моделювання та інформаційні технології. 2023. С. 286–291.

19 Fryz M., Mlynko B. Determination of the characteristic function of discrete-time conditional linear random process and its application // Scientific Journal of TNTU. 2023. Vol. 109, № 1. P. 16–23.

20 Helenius Jouni. Web Application Upgrade to New Modern Technology: Case Tarmo Volunteer Enrolment Service // Helsinki Metropolia University of Applied Sciences. 2022. 54 p.

21 Duckett J. HTML and CSS: Design and Build Websites // John Wiley & Sons. Indianapolis, 2011. 490 p.

22 Meyer E. A., Weyl E. A. CSS: The Definitive Guide // O'Reilly Media. Sebastopol, 2020. 512 p.

23 Fowler M. Patterns of Enterprise Application Architecture // Addison-Wesley Professional. Boston, 2002. 560 p.

24 Flanagan D. JavaScript: The Definitive Guide // O'Reilly Media. Sebastopol, 2020. 706 p.

25 Bassett Lindsay. Introduction to JavaScript Object Notation: A To-the-Point Guide to JSON // O'Reilly Media, Inc. 2015. 136 p.

26 Mardan Azat. Pro Express.js: Master Express.js: The Node.js Framework for Your Web Development // Apress. 2014. 350 p.

27 Casciaro Mario, Mammino Luciano. Node.js Design Patterns: Design and implement production-grade Node.js applications using proven patterns and techniques // Packt Publishing Ltd. 2020. 582 p.

28 Schwartz Baron, Zaitsev Peter, Tkachenko Vadim. High Performance MySQL: Optimization, Backups, and Replication // O'Reilly Media, Inc. 2012. 828 p.

29 Newman Sam. Building Microservices: Designing Fine-Grained Systems // O'Reilly Media, Inc. 2021. 600 p..

30 Бедрій І. Я. Безпека життєдіяльності : навч. посіб. / І. Я. Бедрій, В. Я. Нечай. – Львів : Магнолія 2006, 2007. – 499 с.

31 Stuttard Dafydd, Pinto Marcus. The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws // John Wiley & Sons. 2011. 912 p.

32 Глущенко С. Д. Соціально-психологічні особливості Інтернетаддиктивної поведінки особистості // Молодь: освіта, наука, духовність. 2008.

33 ДСТУ 7239:2011. Система стандартів безпеки праці // Держспоживстандарт України. 2011.

34 Гігієнічна класифікація праці // МОЗ України. Київ, 2019. 55 с.

35 Жидецький В. Ц. Охорона праці користувачів комп'ютерів : підручник // Афіша. Львів, 2020. 176 с.