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 СОЦІАЛЬНОЇ МЕРЕЖІ З ВИКОРИСТАННЯМ LARAVEL, WEBSOCKET-ПРОТОКОЛУ ТА MYSQL

DEVELOPMENT OF A SOCIAL NETWORK USING LARAVEL, WEBSOCKET PROTOCOL AND MYSQL

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

Виконав: здобувач вищої освіти
групи ІПЗс-21
Мельничук Назар Сергійович

Керівник:
Суринович Олена Миколаївна

Кваліфікаційну роботу
допущено до захисту
«10» 06 2025 р.
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 2024р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Мельничуку Назару Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *«Розробка соціальної мережі з використанням Laravel, Websocket-протоколу та MySQL»*
Керівник роботи: *к.т.н., доцент Суринович О. М.*
затверджені наказом закладу вищої освіти від *«19» грудня 2024 р.* № *474/01- 02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра *«10» червня 2025 р.*
3. Вихідні дані до роботи: *React.js, Next.js, Laravel, Reverb, Reverb Echo, MySQL, Docker, Tanstack Query, Zustand, React Hook Form, Zod, Mantine UI, Motion, Tailwind CSS, i18n, Nginx, WebStorm, PhpStrorm, Postman, PhpMyAdmin, методичні вказівки до виконання кваліфікаційної роботи бакалавра.*
4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *формулювання вимог до нової системи, проєктування архітектури та функціональних можливостей додатку, розробка фронтенд-частини з використанням Next.js і Feature-Sliced Design, реалізація бекенд-частини на Laravel з підтримкою WebSocket, розробка бази даних у MySQL, реалізація обміну повідомленнями в реальному часі, тестування і налагодження системи, SEO-оптимізація (налаштування мета-даних, robots.txt, sitemap.xml), документування процесу розгортання проєкту за допомогою Docker.*
5. Перелік графічного матеріалу: *7 рисунків.*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Суринович О. М.*		
Специфікація вимог до розробленої системи	*Суринович О. М.*		
Розробка об'єкта проектування	*Суринович О. М.*		
Нормоконтроль	*Вознюк А. В.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	3,84 %		
Академічна доброчесність	*Суринович О. 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.	вик.
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 01.03.2025 р.	вик.
3	Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 15.03.2025 р.	вик.
4	Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних	до 29.03.2025 р.	вик.
5	Практична реалізація об'єкта проектування та розробка бази даних	до 26.04.2025 р.	вик.
6	Тестування та налагодження об'єкта проектування	до 03.05.2025 р.	вик.
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру	до 10.06.2025 р.	вик

Здобувач вищої освіти Назар МЕЛЬНИЧУК

Керівник кваліфікаційної роботи Олена СУРИНОВИЧ

АНОТАЦІЯ

Мельничук Н. С. Розробка соціальної мережі з використанням Laravel, Websocket-протоколу та MySQL. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності 121 «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

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

У першому розділі здійснено аналіз існуючих соціальних мереж та сформульовано завдання. У другому розділі визначено вимоги до системи та обґрунтовано вибір технологічного стеку. У третьому розділі розроблено клієнтську та серверну частини, реалізовано обмін повідомленнями в реальному часі, проведено тестування та підготовлено супровідну документацію. У висновках підбито підсумки виконаної роботи та окреслено напрямки подальшого розвитку.

Ключові слова: соціальна мережа, Next.js, Laravel, WebSocket, Reverb, Reverb Echo, MySQL, Docker, Nginx.

ABSTRACT

Melnychuk N. Development of a social network using Laravel, WebSocket protocol, and MySQL. Manuscript. Qualification work of the bachelor's program "Software engineering" in the specialty "Software engineering". Lutsk National Technical University. Lutsk, 2025.

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

The first chapter analyzes existing social networks and formulates the objectives. The second chapter defines the system requirements and justifies the choice of the technology stack. The third chapter describes the development of the client and server parts, the implementation of real-time messaging, testing, and the preparation of supporting documentation. The conclusions summarize theresults of the work and outline directions for further development.

Keywords: social network, Next.js, Laravel, WebSocket, Reverb, Reverb Echo, MySQL, Docker, Nginx.

ЗМІСТ

ВСТУП

Актуальність теми. Соціальні мережі стали невід'ємною частиною сучасного цифрового світу, забезпечуючи комунікацію, обмін інформацією та соціальну взаємодію між користувачами. Вони активно розвиваються, впливаючи на різні сфери життя, зокрема бізнес, освіту та культуру. Основними тенденціями розвитку є покращення персоналізації контенту, удосконалення алгоритмів взаємодії між користувачами та розширення можливостей мультимедійного контенту. Крім того, сучасні соціальні мережі все частіше інтегрують нові засоби модерації та боротьби з дезінформацією.

Сучасні соціальні мережі стали віртуальними просторами, де люди знаходять друзів, діляться емоціями та обмінюються думками. Проте, у морі можливостей часом губиться справжня зручність: складність у керуванні соціальними зв'язками, недостатня інтеграція технологій реального часу та відсутність інтуїтивних механізмів взаємодії. Саме тому виникає потреба у створенні нових платформ, які забезпечать більш комфортну і ефективну комунікацію.

У світі активно розвиваються технології, які спрямовані на вирішення зазначених проблем. Використання WebSocket для чату забезпечує миттєвий обмін повідомленнями, а ефективні алгоритми керування друзями дозволяють покращити соціальну взаємодію користувачів. Також спостерігається тенденція до використання інтуїтивного UI/UX дизайну, що підвищує зручність користування системами. Все більше платформ впроваджують можливість персоналізації профілів та налаштування відображення контенту для покращення користувацького досвіду.

Актуальність даної роботи полягає в розробці сучасної соціальної мережі, яка забезпечить комфортну взаємодію між користувачами, гнучку систему управління запитами у друзі, а також реальний час обміну повідомленнями. Це сприятиме покращенню комунікації та підвищенню загальної зручності користування соціальними платформами. Додатково, інтеграція змінної теми

інтерфейсу (світла/темна) сприяє персоналізації користувацького досвіду та підвищенню зручності використання додатку в різних умовах освітлення.

Отже, основною метою кваліфікаційної роботи є розробка веб-додатку соціальної мережі з базовими можливостями для взаємодії між користувачами.

Об’єктом кваліфікаційної роботи є цифрова платформа соціальної мережі.

Предметом кваліфікаційної роботи є архітектура та функціональні можливості веб-додатку соціальної взаємодії.

Для досягнення поставленої мети були поставлені наступні завдання:

– аналіз існуючих рішень;
– проектування архітектури додатку за допомогою UML діаграм:
– розробка системи додавання та відхилення запитів у друзі;
– реалізація функціоналу відстеження онлайн статусів користувачів;
– створення механізму додавання постів із можливістю лайків;
– інтеграція чату на базі WebSocket для забезпечення реального часу обміну повідомленнями;
– розробка функціоналу перегляду та редагування профілю користувача;
– створення системи авторизації та реєстрації користувачів;
– впровадження можливості зміни теми інтерфейсу (світла/темна);
– автоматизація розгортання за допомогою Docker;
– впровадження SEO-оптимізації.

РОЗДІЛ 1
АНАЛІЗ РОЗРОБКИ ВЕБ-ДОДАТКІВ І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Сучасні соціальні мережі є невід’ємною частиною цифрового простору та виступають головними платформами для комунікації, обміну інформацією, побудови особистих і професійних зв’язків. Найбільш відомими прикладами таких платформ є Facebook, Instagram, X та LinkedIn. Кожна з них орієнтована на свою аудиторію та виконує певні функції: від особистого спілкування до професійного нетворкінгу та обміну контентом різного формату. Згідно з численними дослідженнями, соціальні мережі радикально змінили підходи до комунікації на глобальному рівні. Вони вплинули не лише на особисте життя мільйонів користувачів, але й на бізнес-процеси, маркетинг, журналістику та інші сфери.

Попри широке використання, більшість існуючих платформ мають низку обмежень і викликів, що залишають простір для вдосконалення. Однією з головних проблем залишається недостатній рівень захисту персональних даних та приватності користувачів. Багато великих соціальних мереж, зокрема Facebook, неодноразово ставали об’єктами критики через порушення конфіденційності, витоки інформації або зловживання персональними даними для рекламних цілей. Зокрема, у 2019 році Facebook погодився сплатити рекордний штраф у 5 мільярдів доларів за порушення правил конфіденційності, пов'язане з незаконною передачею персональних даних мільйонів користувачів аналітичній компанії Cambridge Analytica [1]. Крім того, часто користувачі стикаються з обмеженими можливостями персоналізації функцій і взаємодії. Складні алгоритми формування стрічок новин не завжди відповідають реальним інтересам користувачів, що призводить до втрати контролю над тим, який контент вони споживають.

Ще однією актуальною проблемою є технічні обмеження в роботі з різними типами контенту. Наприклад, Instagram орієнтований переважно на візуальні матеріали та має обмежену підтримку текстових публікацій і функціональності для тривалих обговорень. Комунікаційні можливості у вигляді чатів у таких платформах також не завжди відповідають сучасним стандартам зручності та інтерактивності. Зокрема, у LinkedIn, що позиціонується як мережа для професійного спілкування, особисті чати є лише додатковою функцією і не забезпечують повноцінного користувацького досвіду для неформального спілкування або обміну мультимедійними файлами.

Окремо варто зазначити, що патентні пошуки в галузі соціальних мереж свідчать про постійний розвиток технологій, спрямованих на підвищення ефективності взаємодії між користувачами. Серед сучасних технічних напрямів особливе місце займають системи рекомендацій на основі штучного інтелекту, які дозволяють персоналізувати стрічку контенту відповідно до інтересів користувача. Також важливу роль відіграють комунікаційні протоколи реального часу, такі як WebSocket, що дозволяють мінімізувати затримки при обміні повідомленнями й забезпечити миттєву взаємодію між учасниками мережі.

Вивчаючи існуючі соціальні платформи, можна виокремити як їхні сильні сторони, так і недоліки. З одного боку, такі сервіси, як Facebook чи Instagram, мають величезну користувацьку базу та розвинену інфраструктуру, яка дозволяє обробляти мільйони запитів щодня. З іншого боку, ці платформи часто перевантажені зайвими функціями, агресивною рекламою та складною навігацією, що створює бар’єри для нових користувачів і знижує загальний рівень зручності використання.

Проблеми конфіденційності, недостатній рівень персоналізації та перевантаження функціоналом створюють передумови для появи нових соціальних платформ, здатних забезпечити користувачам більш зрозумілий, безпечний і зручний у використанні інтерфейс взаємодії. Особливу увагу при розробці таких систем варто приділяти підтримці реального часу, покращеним інструментам для комунікації, прозорому управлінню персональними даними та

можливості гнучко налаштовувати стрічку контенту відповідно до особистих побажань кожного користувача.

Отже, аналіз сучасного стану галузі підтверджує, що попри велику кількість популярних соціальних мереж, залишається потреба в альтернативних рішеннях, які поєднуватимуть зручність використання, високу швидкодію, реальний контроль над приватністю та підтримку сучасних технологій для миттєвого обміну інформацією. Розробка нової соціальної платформи є актуальним завданням, яке дозволить задовольнити ці потреби та запропонувати користувачам сучасний інструмент для безпечного та ефективного спілкування.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Для реалізації поставленої мети створення веб-додатку соціальної мережі слід виконати низку взаємопов’язаних завдань, що охоплюють весь життєвий цикл створення програмного забезпечення – від дослідження до впровадження.

Ці завдання охоплюють аналітику, проєктування, реалізацію основного функціоналу, налаштування інфраструктури та підготовку до продуктивної експлуатації системи, зокрема:

– Аналіз існуючих рішень. На першому етапі буде проведено аналіз популярних соціальних мереж, таких як Facebook, Instagram, LinkedIn. Планується дослідити їхні функціональні можливості, переваги й обмеження, механізми взаємодії між користувачами, персоналізацію, архітектурні особливості та UX. Результатом стане виявлення недоліків наявних рішень та обґрунтування доцільності розробки нової платформи;

– проєктування архітектури додатку за допомогою UML-діаграм. На цьому етапі буде створено UML-діаграми, що відображають основні компоненти системи, взаємозв’язки між ними, сценарії використання функціоналу та логіку обробки подій. Також буде обґрунтовано вибір стеку технологій, моделювання бази даних і використання ORM для обробки запитів;

– розробка системи додавання та відхилення запитів у друзі. У реалізації буде передбачено повноцінний механізм соціальної взаємодії, який включатиме надсилання, підтвердження та відхилення запитів у друзі, а також збереження відповідних станів взаємин;

– реалізація функціоналу відстеження онлайн статусів користувачів. Цей модуль дозволить відображати присутність користувачів у мережі в реальному часі, що підвищить зручність комунікації між учасниками платформи;

– створення механізму додавання постів із можливістю лайків. Буде реалізовано стрічку новин, де користувачі зможуть публікувати текстові пости та взаємодіяти з ними через функцію вподобання;

– інтеграція чату на базі WebSocket. Передбачено впровадження приватного обміну повідомленнями, який функціонуватиме на основі WebSocket-протоколу, що забезпечить миттєве оновлення інтерфейсу та подій без перезавантаження;

– розробка функціоналу перегляду та редагування профілю користувача. Користувачі зможуть переглядати інформацію про себе, редагувати персональні дані та переглядати власні пости;

– створення системи авторизації та реєстрації користувачів. Буде реалізовано механізми автентифікації та збереження сеансів користувачів, з урахуванням валідації даних і захищеного доступу до функціоналу системи;

– впровадження можливості зміни теми інтерфейсу (світла/темна). Для покращення персоналізації та зручності в різних умовах освітлення буде реалізовано перемикання теми оформлення;

– автоматизація розгортання за допомогою Docker. Проєкт буде розгорнуто у вигляді ізольованих сервісів через Docker-контейнери з попередньо налаштованими конфігураційними файлами, що дозволить легко запускати інфраструктуру локально або на сервері;

– впровадження SEO-оптимізації. Буде реалізовано налаштування мета-тегів, створення файлу robots.txt та sitemap.xml, конфігурацію Open Graph-даних і PWA-параметрів для покращення індексації в пошукових системах.

Висновки до розділу 1

Аналіз сучасних соціальних мереж показав, що попри їхню популярність, вони мають низку недоліків, зокрема проблеми з конфіденційністю та обмежену персоналізацію контенту. Висока конкуренція у сфері соціальних платформ також ускладнює впровадження нових рішень, однак постійний технологічний розвиток відкриває можливості для створення альтернативних підходів.

Завданням даної кваліфікаційної роботи є створення прототипу нової соціальної мережі, що поєднує ефективну взаємодію користувачів, гнучкість у налаштуваннях і високу продуктивність. Для цього використано сучасний технологічний стек: Next.js із серверними компонентами для фронтенду, Laravel для бекенду, WebSocket через Reverb для обміну даними в реальному часі та Laravel Sanctum для безпечної аутентифікації через токени.

Запропонована система спрямована на створення комфортного середовища для спілкування, яке враховує потреби користувачів у швидкій комунікації, персоналізації та безпеці. Завдяки використанню передових технологій вона зможе конкурувати з існуючими рішеннями та забезпечить сучасний підхід до соціальної взаємодії.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

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

Для проєктування соціальної мережі було обрано об’єктно-орієнтований підхід до моделювання з використанням UML (Unified Modeling Language). «UML був створений для визначення, візуалізації, проєктування й документування в основному програмних систем» [2]. Цей підхід є доцільним, оскільки дозволяє:

– чітко визначити структуру системи та зв’язки між її елементами;

– моделювати динаміку поведінки користувачів та взаємодії з підсистемами;

– забезпечити масштабованість та зрозумілу архітектуру, зокрема при командній розробці;

– синхронізувати бачення між front-end та back-end частинами системи, що є критичним для SPA-додатків з API.

Мовою моделювання обрано UML, зокрема:

– діаграми варіантів використання (Use Case) – для опису взаємодії користувачів із системою;

– діаграми класів – для моделювання сутностей (користувач, пост, коментар, чат);

– діаграми діяльності – для демонстрації логіки обробки подій (наприклад, додавання в друзі);

– діаграми послідовності – для показу обміну повідомленнями між фронтом, сервером і базою даних.

Вибір UML зумовлений його широким застосуванням у промисловій розробці та можливістю побудови як високорівневої архітектури, так і

деталізованих моделей компонентів. Це дозволяє систематизувати вимоги, забезпечити точність у реалізації функціоналу та спростити супровід проєкту.

На діаграмі варіантів використання (рис. 2.1) зображено двох акторів – User 1 та User 2, які є типізованими представниками кінцевих користувачів соціальної мережі, причому «діаграма прецедентів показує взаємодію зовнішніх користувачів із системою для досягнення певної мети» [3].

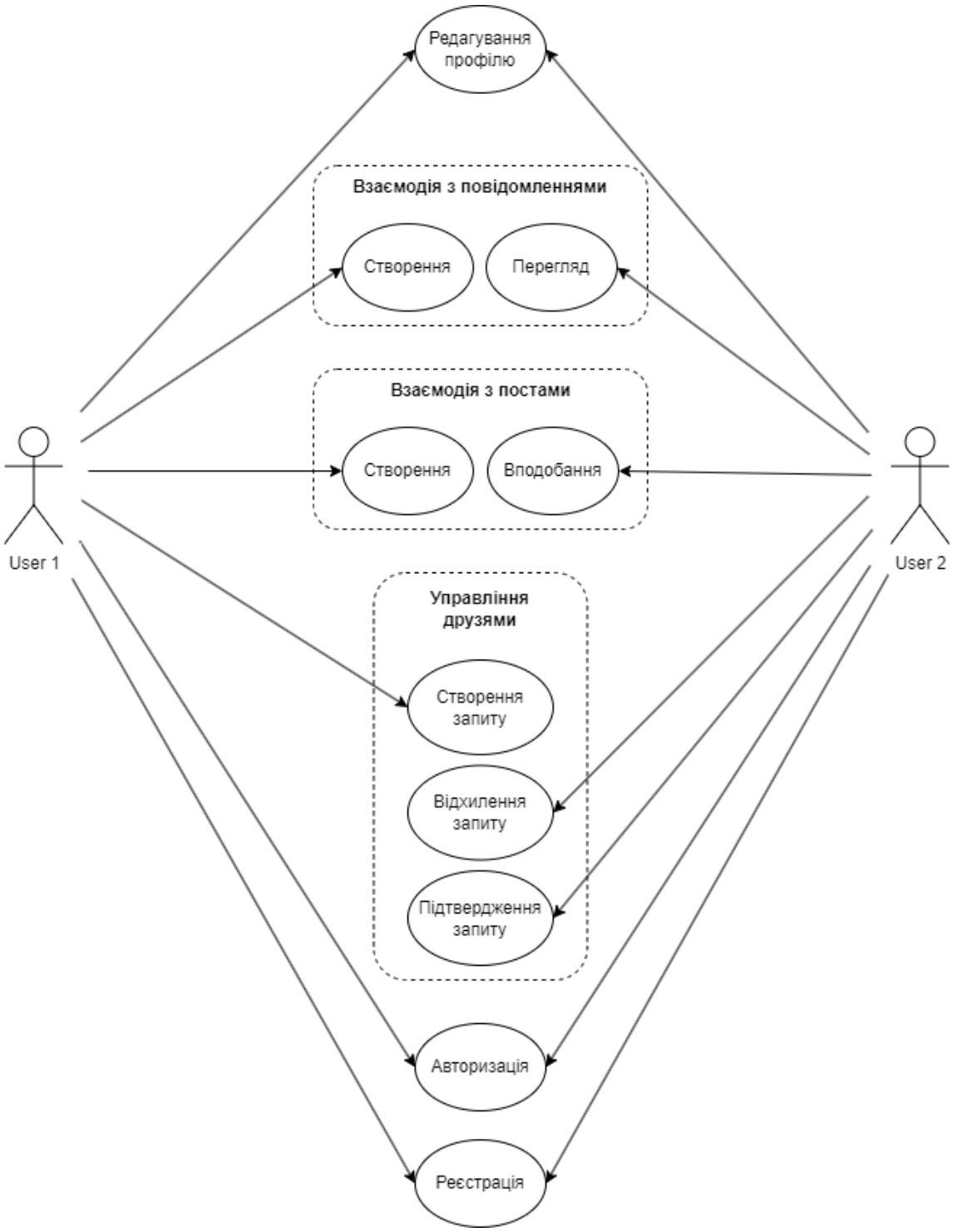

Рисунок 2.1 – UML діаграма варіантів використання

Користувачі можуть здійснювати базові операції автентифікації – реєстрацію та авторизацію. Ці процеси є обов’язковими для доступу до інших можливостей системи та забезпечують базовий рівень захисту.

Після входу до системи користувачі мають змогу редагувати власний профіль, що включає зміну персональної інформації (ім’я, аватар, контактні дані тощо). Це дозволяє підтримувати актуальність облікового запису та персоналізацію взаємодії.

Функціонал взаємодії з повідомленнями охоплює створення та перегляд повідомлень, що реалізовано через WebSocket-з’єднання. Цей функціонал забезпечує миттєвий обмін інформацією між користувачами в режимі реального часу, що є ключовою вимогою сучасних соціальних мереж.

У модулі взаємодії з постами передбачено можливість створення нових публікацій та функцію вподобання. Це формує стрічку новин, де користувачі можуть ділитися контентом і взаємодіяти з публікаціями інших.

Блок управління друзями складається з трьох окремих варіантів використання: створення запиту на дружбу, відхилення запиту, та підтвердження запиту. Така деталізація дозволяє моделювати всі можливі стани взаємин між користувачами та забезпечує логіку побудови списку друзів.

Таким чином, діаграма охоплює всі ключові компоненти системи: автентифікацію, соціальні взаємодії (пости, друзі, повідомлення) та персоналізацію. Вона є узагальненим відображенням функціональної специфікації й допомагає як розробникам, так і стороннім учасникам команди швидко зрозуміти структуру і логіку роботи додатку.

На діаграмі класів (рис. 2.2), у структурі соціальної мережі ключовим елементом є користувач (клас User), причому «діаграма класів – це статичне представлення структури моделі в UML» [4]. Він містить основну інформацію про особу: електронну пошту, ім’я користувача, зображення профілю, ім’я, прізвище, статус, місце проживання, номер телефону та місце роботи. Для взаємодії з системою користувач має можливість зареєструватися, увійти до

облікового запису та оновити свої дані – ці дії реалізовані методами register(), login() та update() відповідно.

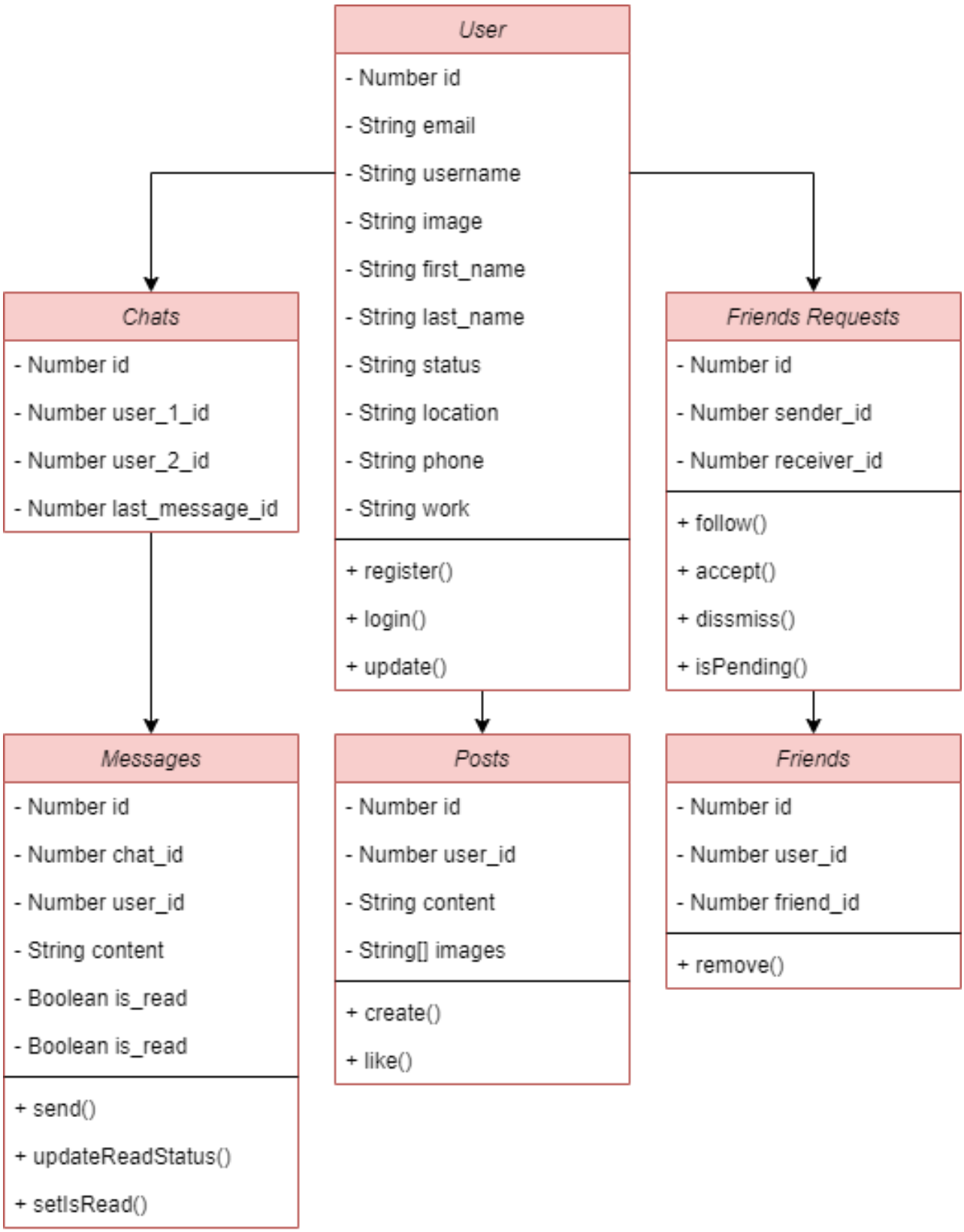

Рисунок 2.2 – UML діаграма класів

Комунікація між користувачами реалізована через систему чатів. Кожен чат (клас Chats) пов’язує двох користувачів і зберігає ідентифікатор останнього повідомлення. Повідомлення (клас Messages) належать до певного чату 1 на 1, мають автора повідомлення, текстовий вміст та відмітку про прочитання. Повідомлення можна надсилати за допомогою методу send(), а також оновлювати їхній статус прочитання через setIsRead() та змінювати позначку для відправника за допомгою updateReadStatus(), щоб при наступному відркитті чату було відображено з останього прочитаного ним повідомлення.

Окрім спілкування, користувачі можуть створювати пости (клас Posts). Кожен пост містить текстовий контент і, за потреби, прикріплені зображення. Основні дії з постами – це створення (create()) і вподобання (like().

Для реалізації дружніх зв’язків між користувачами використовуються дві сутності: FriendRequests та Friends. Перший клас відповідає за логіку запитів у друзі – хто кому надіслав запит, хто його прийняв або відхилив. Відповідні дії реалізовані методами add(), accept() і dissmiss(). Після підтвердження дружби запис про неї потрапляє до таблиці Friends, де зберігається зв’язок між двома користувачами. Дружбу можна скасувати через метод remove().

На першій діаграмі діяльності відображено процес створення запиту на дружбу (рис. 2.3), причому «діаграма діяльності (англ. activity diagram) – в UML та SysML, візуальне представлення графу діяльностей» [5]. Дія починається з ініціації запиту користувачем. Система перевіряє, чи існує вже зворотній запит від особи, якій він адресований. У випадку виявлення такого запису, система автоматично фіксує факт дружби між двома користувачами без додаткового підтвердження – що забезпечує швидку і взаємну соціальну взаємодію. Якщо зворотного запиту не існує, система створює новий запис у таблиці заявок і надсилає повідомлення отримувачу про новий запит. Така логіка дозволяє уникнути дублювання та відповідає типовій поведінці користувачів соціальних мереж.

Рисунок 2.3 – UML діаграма діяльності створення запиту на дружбу

Наступна діаграма діяльності моделює сценарій прийняття запиту на дружбу (рис. 2.4). У цьому процесі система перевіряє, чи дійсно існує відповідний запит у базі даних заявок. Якщо запит відсутній – користувач отримує повідомлення про це, і обробка завершується. Якщо ж запис знайдено, він видаляється з бази заявок, а у таблиці друзів створюється новий запис, який формалізує зв’язок між двома користувачами. Після цього відправник запиту отримує повідомлення про успішне встановлення дружби. Така послідовність дій забезпечує логічну цілісність і правильне оновлення стану відносин між користувачами. Важливо, що перевірка наявності запиту виконується до внесення змін у базу, що запобігає помилковому оновленню та гарантує узгодженість даних. Даний сценарій є прикладом транзакційної логіки, де кожен крок залежить від успішного виконання попереднього.

Рисунок 2.4 – UML діаграма діяльності прийняття заявки дружби

Остання діаграма діяльності описує відхилення запиту в друзі (рис. 2.5). Як і в попередніх сценаріях, першим кроком є перевірка наявності заявки в базі. Якщо запит відсутній – процес завершено без змін. У разі його існування відбувається видалення відповідного запису з таблиці заявок. Додаткових дій або сповіщень не передбачається, що дозволяє користувачам делікатно відмовляти у встановленні контакту без створення соціального дискомфорту. Такий механізм сприяє збереженню нейтрального досвіду взаємодії на платформі. Водночас це знижує ризик конфліктів між користувачами та спрощує логіку обробки запитів на рівні системи.

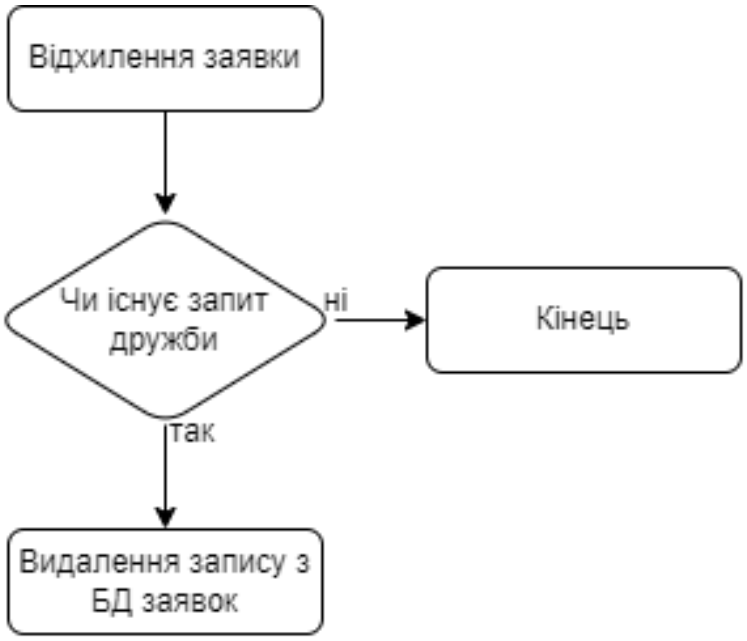

Рисунок 2.5 – UML діаграма діяльності відхилення заявки дружби

На діаграмі послідовності (рис. 2.6) зображено процес надсилання повідомлення користувачем у чаті, де «діаграма послідовності відображає взаємодії об’єктів, впорядковані за часом» [6]. У взаємодії беруть участь чотири основні компоненти: користувач, Frontend, API та база даних (БД).

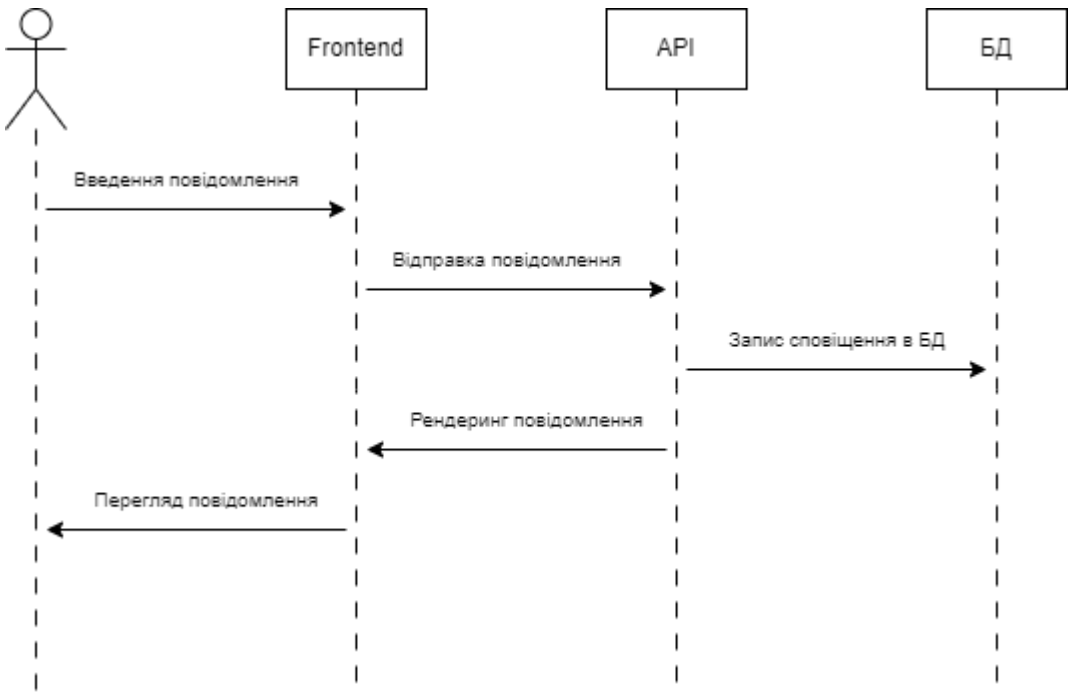

Рисунок 2.6 – UML діаграма послідовності чату

Процес розпочинається з того, що користувач вводить повідомлення в інтерфейсі клієнтської частини (Frontend) та ініціює його відправку. Frontend приймає це повідомлення і передає його на сервер через API-запит. Серверна частина (API) приймає запит, проводить обробку повідомлення, включно з перевіркою прав доступу, валідацією вмісту та визначенням чату, до якого належить повідомлення.

Далі API надсилає запит до бази даних із метою збереження повідомлення. Після успішного запису система повертає підтвердження API, який, у свою чергу, сповіщає клієнтську частину про те, що повідомлення було успішно надіслане.

Frontend після отримання підтвердження оновлює інтерфейс користувача та виводить повідомлення у вікні чату. У результаті користувач бачить, що його повідомлення з’явилося в чаті, що завершує цикл взаємодії.

Інтерфейс користувача розробленої соціальної мережі побудований на принципах сучасного UX/UI-дизайну, орієнтованого на інтуїтивність, логіку та комфортність взаємодії. Основна навігація розташована в лівій частині екрану у вигляді вертикального меню, яке дозволяє швидко перемикатися між основними розділами: пости, друзі, користувачі та повідомлення. Така структура забезпечує постійну доступність основних функцій незалежно від контексту, в якому перебуває користувач.

Вся система виконана в темній кольоровій схемі з акцентами синього кольору, що не лише покращує візуальне сприйняття, але й знижує навантаження на очі під час тривалої роботи. Існує додаткова можливість переключити на світлу тему. Елементи керування мають чітку ієрархію та достатній простір між собою, що дозволяє комфортно взаємодіяти з ними навіть на сенсорних екранах. Усі інтерактивні компоненти – кнопки, поля вводу, іконки – мають виразний візуальний стан: активний, наведений, неактивний, що підвищує передбачуваність і керованість інтерфейсу.

Користувацький досвід також враховує адаптивність – інтерфейс коректно масштабується під різні розміри екранів, що забезпечує повноцінну роботу на мобільних пристроях та десктопах. Переходи між станами додатку реалізовані без перезавантаження сторінки, що створює враження безперервної взаємодії. Всі дані оновлюються в реальному часі завдяки використанню WebSocket, що особливо важливо для модуля повідомлень і загальної динамічності середовища.

Інтерфейс профілю користувача надає можливість перегляду основної інформації, зображення, особистих даних і публікацій. Персоналізований простір дозволяє підтримувати актуальність відображуваної інформації та демонструвати активність у мережі. Дизайн спрямований на те, щоби мінімізувати кількість кліків до цільової дії, забезпечуючи при цьому логічну послідовність і передбачуваність кожного кроку.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

У процесі створення соціальної мережі було проаналізовано низку сучасних технологічних стеків та інструментів, які можуть бути застосовані для побудови вебзастосунків із високим рівнем взаємодії та масштабованості. З огляду на специфіку системи – наявність реального часу, взаємодії між користувачами, публікаційного контенту та персоніфікації – було прийнято рішення використовувати архітектурно розділену модель із незалежними фронтенд- і бекенд-частинами.

Фронтенд реалізовано за допомогою Next.js 15, який забезпечує підтримку Server Components, що дозволяє рендерити частини інтерфейсу на сервері, мінімізуючи обсяг даних, які передаються клієнту, адже «Next.js – це фреймворк для серверного рендерингу веб-додатків на React» [7]. У структурі фронтенду застосовано архітектуру Feature-Sliced Design, що дозволяє ізольовано розробляти окремі функціональні блоки системи, адже «методологія дозволяє

зручно розділяти застосунок на незалежні блоки, які легко масштабувати та супроводжувати» [8].

Для валідації форм обрано Zod, який інтегрується з React Hook Form, забезпечуючи декларативне управління станом форм. Клієнтське кешування та синхронізація даних реалізовані за допомогою React Query, що дозволяє ефективно керувати запитами до API, обробляти статуси, помилки та повторні запити. Глобальний стан програми управляється через Zustand – легку бібліотеку для зберігання даних, яка ідеально підходить для сучасних SPA-додатків.

Візуальний інтерфейс побудовано з використанням UI-бібліотеки Mantine, яка забезпечує компоненти з високим рівнем кастомізації та адаптивності. Завдяки цьому вдалося реалізувати єдину візуальну мову для всієї системи.

Бекенд-сервер реалізований з використанням Laravel 11, який забезпечує швидку розробку REST API з підтримкою авторизації, валідації запитів, управління ресурсами та взаємодії з базою даних. Аутентифікація реалізована через Sanctum, що дозволяє безпечно працювати з токенами доступу для SPA.

Для забезпечення обміну повідомленнями в реальному часі використано Laravel Reverb з підтримкою WebSocket-з’єднання. Це дозволяє реалізувати динамічний чат, миттєве оновлення статусів, сповіщень та інші інтерактивні елементи. Передача даних між клієнтом і сервером відбувається асинхронно, що забезпечує плавну взаємодію.

База даних реалізована на основі MySQL, розгорнутої у Docker-контейнері. Такий підхід дозволяє швидко розгортати та ізолювати середовище, спрощує тестування та перенесення проєкту між машинами. Всі сервіси також оркестровані за допомогою docker-compose, що дозволяє керувати середовищем у складі одного конфігураційного файлу.

Алгоритмічна частина розробки включає базові операції фільтрації, сортування, пагінації, перевірки статусів (наприклад, перевірка, чи є користувач другом, чи вже відправлено запит) та обробки потоків даних у реальному часі. Для кожного модуля (пости, друзі, чат) було побудовано окремі сервіси з чітким розмежуванням відповідальностей. У випадку чату – реалізовано логіку обробки

подій message:sent, message:read, а також підтримку асинхронної доставки через буфер обміну.

У розробці системи також застосовано моделі, побудовані у нотації UML: діаграма класів описує основні сутності, діаграма послідовності демонструє процес надсилання повідомлення в чат, а діаграми варіантів використання описують поведінку користувача в системі. Це дозволяє краще структурувати логіку проєкту й забезпечити візуальну підтримку архітектурних рішень.

Загалом обраний набір засобів і методів дозволяє створити продуктивну, стабільну та масштабовану соціальну мережу з широкими можливостями для подальшого розвитку та інтеграції нових функцій.

Висновки до розділу 2

У межах другого розділу було здійснено детальний аналіз вимог до програмного забезпечення та обґрунтовано вибір інструментів і технологій для реалізації соціальної мережі. На основі поставлених функціональних задач – таких як створення публікацій, обмін повідомленнями, управління контактами – побудовано модельну основу системи у вигляді UML-діаграм, що відображають структуру даних, сценарії взаємодії та послідовність дій користувача.

Для реалізації функціоналу було обрано сучасний стек технологій, що забезпечує гнучкість і масштабованість: Next.js з підтримкою Server Components як фронтенд-платформу, Laravel як надійний бекенд-фреймворк, WebSocket (Reverb) для обміну даними в реальному часі, а також Laravel Sanctum для токен-орієнтованої аутентифікації. Особливу увагу приділено архітектурі застосунку: використання Feature-Sliced Design дало змогу чітко структуризувати логіку клієнтської частини та забезпечити ізольованість функціональних модулів.

РОЗДІЛ 3
РОЗРОБКА СОЦІАЛЬНОЇ МЕРЕЖІ З ВИКОРИСТАННЯМ LARAVEL, WEBSOCKET-ПРОТОКОЛУ ТА MYSQL

3.1 Практична реалізація об’єкта проєктування

Розробка соціальної мережі вимагала поєднання сучасних підходів у створенні веб-застосунків, що орієнтовані на масштабованість, реальний час роботи системи та зручність користування. У межах проєкту реалізація програмного забезпечення охоплювала побудову клієнтської та серверної частин, налаштування безпечної аутентифікації, обробку взаємодій між користувачами та забезпечення миттєвого обміну даними.

Процес практичної реалізації розпочався із розробки архітектури фронтенд-частини на основі Next.js 15 із використанням Server та Client Components. Це дозволило частину завдань обробляти як на сервері та на клієнті. Використання Server Components суттєво покращило швидкість завантаження сторінок і оптимізувало SEO-індексацію, а a Client Components дозволила реалізувати інтерактивний функціонал, який вимагає взаємодії з користувачем у режимі реального часу, обробку подій, керування локальним станом та роботу з асинхронними діями без необхідності повторного завантаження сторінки.

Взаємодія користувача із системою починається із автентифікації. Для реалізації реєстрації та авторизації було використано бібліотеку React Hook Form, яка забезпечує «ефективне керування станом форм, зменшуючи кількість повторних рендерів і спрощуючи обробку введених даних» [9]. У комбінації з Zod для типізованої валідації введених даних, адже «Zod дозволяє створювати схеми перевірки даних з чіткою типізацією, що полегшує інтеграцію з бібліотеками для обробки форм» [10]. Такий підхід дозволяє гнучко адаптувати логіку перевірки залежно від сценаріїв використання та зменшує ризики помилок при взаємодії з бекендом. Схему валідації форми авторизації наведено у лістингу 3.1.

Лістинг 3.1 – Демонстрація валідації форми реєстрації на Zod

```
export const Schema = z.object({
    email: z.string().min(1, "Enter email")
       .email("Invalid email address"),
    username: z.string().min(1, "Enter username"),
    password: z.string().min(1, "Enter password"),
    c_password: z.string().min(1, "Enter confirm password"),
}).refine((data) => data.password === data.c_password, {
    message: "Passwords do not match",
    path: [ "c_password" ],
});

export type SignUpFormType = z.TypeOf<typeof Schema>
```

кінець лістингу 3.1

В лістингу 3.2 наведено обробку даних після успішного входу та збереження доступів та інформації про користувача за допомогою Zustand.

Лістинг 3.2 – Демонстрація збереження даних користувача в store

```
export const useUserStore = create<UserStore>()((set) => ({
    user: {
       id: null,
       username: null,
       first_name: null,
       last_name: null,
       image: null,
       info: {
          location: null,
          phone: null,
          email: null,
          birthday: null,
       },
    },
    token: null,
    setUser: (user) => set((state) => ({
       user: {
          ...state.user,
          ...user,
       },
    })),
    setToken: (token) => set(() => ({ token: token })),
}));
```

кінець лістингу 3.2

Крім базових функцій авторизації, клієнтська частина підтримує взаємодію із записами користувачів, їх друзями та постами (ліст. 3.3). Для отримання даних використовувалася React Query, що забезпечила ефективне кешування запитів і повторне завантаження даних у разі необхідності.

Лістинг 3.3 – Хук для отримання постів з API

```
export const useGetPostsApi = (page: number, userId?: number,
isMyProfile?: boolean) => {
    return useQuery<GetPostsResponse>({
       queryKey: [ "posts", page ],
       queryFn: async () => {
          return $axios.get("posts", {
             params: {
                "user_id": userId,
                "my": isMyProfile,
                "page": page
             }
          })
       }
    })
};

const { data, isLoading } = useGetPostsApi(page, userId, isMyProfile)
```

кінець лістингу 3.3

Серверна частина була розгорнута на базі Laravel 11 із побудовою стандартного REST API. Для захищеної авторизації використовувалася технологія Laravel Sanctum, яка забезпечувала роботу із токенами без необхідності використання OAuth-авторизації. Sanctum дозволяє реалізувати простий та ефективний механізм автентифікації для SPA-додатків, мобільних клієнтів або звичайних API. Після успішної авторизації клієнт отримує персональний токен, який передається у заголовках HTTP-запитів для доступу до захищених маршрутів. Такий підхід значно спрощує роботу з аутентифікацією, не потребує складної конфігурації та забезпечує достатній рівень безпеки. Приклад коду реєстрації наведено у лістингу 3.4.

Лістинг 3.4 – Код методу реєстрації

```
public function register(Request $request): JsonResponse
{
    $validator = Validator::make($request->all(), [
        ‘username’ => ‘required’,
        ‘email’ => ‘required|email’,
        ‘password’ => ‘required’,
        ‘c_password’ => ‘required|same:password’,
    ]);

    if($validator->fails()){
        return response()->json([
            "errors" => $validator->errors()
        ], 404);
    }

    $errors = [];

    if (User::query()->where(‘email’, $request->email)->exists()) {
        $errors[‘email’] = "This email is already registered.";
    }

    if (User::where(‘username’, $request->username)->exists()) {
        $errors[‘username’] = "This username is already taken.";
    }

    if (!empty($errors)) {
        return response()->json([
            "errors" => $errors
        ], 409);
    }

    $input = $request->all();
    $input[‘password’] = bcrypt($input[‘password’]);

    $user = User::create($input);
    $token = $user->createToken(‘MyApp’)->plainTextToken;

    $success[‘username’] = $user->username;
    $success[‘image’] = $user->image;

    return response()->json([
        "user" => $success,
        "token" => $token,
        "massage" => ‘User login successfully.’
    ], 200);
}
```

кінець лістингу 3.4

Приклад коду авторизації у систему наведений у лістингу 3.5.

Лістинг 3.5 – Код методу авторизації

```
public function login(Request $request): JsonResponse
{
    $userExists = User::query()
        ->where('email', $request->email)
        ->findOrFail();

    if(Auth::attempt([
        'email' => $request->email,
        'password' => $request->password
    ])){
        $user = Auth::user();
        $token = $user->createToken('MyApp')->plainTextToken;

        return response()->json([
            "user" => ProfileUserResource::make($user),
            "token" => $token,
            "message" => 'User login successfully.'
        ], 200);
    }

    else{
        return response()->json([
            "message" => 'Incorrect email or password'
        ], 401);
    }
}
```

кінець лістингу 3.5

Однією з важливих вимог до функціональності соціальної мережі було забезпечення обміну повідомленнями в реальному часі між користувачами, адже «WebSocket – це двонаправлений повнодуплексний протокол зв'язку між клієнтом та сервером» [11]. Для реалізації цієї задачі було використано технологію WebSocket через інструмент Laravel Reverb, що дозволяє підтримувати постійне з'єднання між клієнтом і сервером для миттєвої передачі даних без необхідності оновлення сторінки.

Коли користувач відправляє нове повідомлення в чаті, на бекенді генерується подія SentMessageEvent (ліст. 3.6). Створення та відправлення події

реалізується за допомогою стандартного механізму подій у Laravel. Дана подія передає два параметри: ідентифікатор чату та об’єкт повідомлення, обгорнутий у ресурс MessageResource, що дозволяє передавати лише потрібні дані в структурованому вигляді. Такий підхід забезпечує гнучкість і контроль над тим, яка саме інформація надсилається клієнту, що є важливим з точки зору безпеки та продуктивності.

Лістинг 3.6 – Код виклику події для відправки повідомлення

```
event(new SentMessageEvent($chat_id, new MessageResource($message)));
```

кінець лістингу 3.6

Клас SentMessageEvent (ліст. 3.7) реалізує інтерфейс ShouldBroadcast, що означає автоматичну трансляцію події через Reverb. Метод broadcastOn() визначає канал, на який транслюється повідомлення. У цьому випадку використовується приватний канал chat.{chatId}, що дозволяє обмежити доступ до подій лише авторизованим учасникам конкретного чату.

Лістинг 3.7 – Код виклику події для відправки повідомлення

```
class SentMessageEvent implements ShouldBroadcast
{
    use Dispatchable, InteractsWithSockets, SerializesModels;

    public function __construct(
        public int $chat_id,
        public MessageResource $message,
    )
    {}

    public function broadcastOn(): array
    {
        return [
            new PrivateChannel(‘chat.’ . $this->chat_id),
        ];
    }
}
```

кінець лістингу 3.7

Щоб гарантувати безпеку доступу до приватного каналу, у системі реалізовано клас авторизації ChatChannel (ліст. 3.8). Метод join перевіряє:

– чи автентифікований користувач;
– чи існує чат із заданим ідентифікатором;
– чи є користувач учасником цього чату.

Тільки за виконання всіх цих умов користувач отримує доступ до подій у цьому приватному каналі.

Лістинг 3.8 – Код доступу до приватного каналу

```
class ChatChannel
{
    public function __construct()
    {
        //
    }

    public function join(User $user, int $chatId): bool
    {

        if (!auth()->check()) {
            return false;
        }

        $chat = Chat::query()->find($chatId);

        if (!$chat) {
            return false;
        }

        return
            $user->id === $chat->user_1_id ||
            $user->id === $chat->user_2_id;
    }
}
```

кінець лістингу 3.8

Підписка на приватний канал здійснюється на клієнтській стороні за допомогою бібліотеки Laravel Echo, яка працює поверх протоколу WebSocket. Налаштування Laravel Echo (ліст. 3.9) супроводжувалася передачею

налаштувань, таких як ключ додатку, хост, порт, шлях до WebSocket-сервера, а також параметрів безпеки. Окремо було реалізовано механізм авторизації: при спробі підключення до приватного каналу клієнтська частина надсилає POST-запит на сервер за маршрутом /broadcasting/auth, передаючи ідентифікатор з'єднання (socket_id) та ім'я каналу (channel_name). Сервер, у свою чергу, перевіряє права доступу користувача та у разі успішної аутентифікації дозволяє приєднання до каналу.

Лістинг 3.9 – Основна частина налаштування Laravel Echo

```
useEffect(() => {
    window.Pusher = window.Pusher || Pusher
    window.Echo = new Echo({
       broadcaster: "reverb",
       key: process.env.NEXT_PUBLIC_REVERB_APP_KEY,
       wsHost: process.env.NEXT_PUBLIC_REVERB_HOST,
       wsPath: process.env.NEXT_PUBLIC_REVERB_PATH,
       wsPort: process.env.NEXT_PUBLIC_REVERB_PORT,
       wssPort: process.env.NEXT_PUBLIC_REVERB_PORT,
       forceTLS: false,
       enabledTransports: [ "ws" ],
       withCredentials: true,
       authorizer: (channel: any, options: any) => {
          return {
             authorize: (socketId: any, callback: any) => {
                $axios
                   .post("broadcasting/auth", {
                      socket_id: socketId,
                      channel_name: channel.name,
                   })
                   .then((response) => {
                      callback(false, response)
                   })
                   .catch((error) => {
                      callback(true, error)
                   })
             },
          }
       },
    })

    setEcho(window.Echo)
}, [])
```

кінець лістингу 3.9

При ініціалізації підписки, що продемонстровано у лістингу 3.10, вказується ім’я каналу та тип прослуховуваної події. У разі отримання події SentMessageEvent нове повідомлення додається у локальний стан застосунку без перезавантаження сторінки, що забезпечує миттєве оновлення інтерфейсу користувача. Такий підхід дозволяє реалізувати повноцінну взаємодію в реальному часі, зберігаючи плавність і динаміку спілкування.

Лістинг 3.10 – Підписка до приватного каналу ws

```
echo.private(`chat.${chatId}`)
    .listen("SentMessageEvent", (data: MessageResponse) => {
       setMessage(data)
    })
```

кінець лістингу 3.10

На рисунку 3.1 наведено приклад зовнішнього вигляду сторінки з повідомленнями – в інтерфейс чату, що оновлюється автоматично при надходженні нових даних через WebSocket. Він ілюструє реалізацію механізму прослуховування подій на практиці та демонструє, як інтерфейс реагує на зміни без додаткової взаємодії користувача.

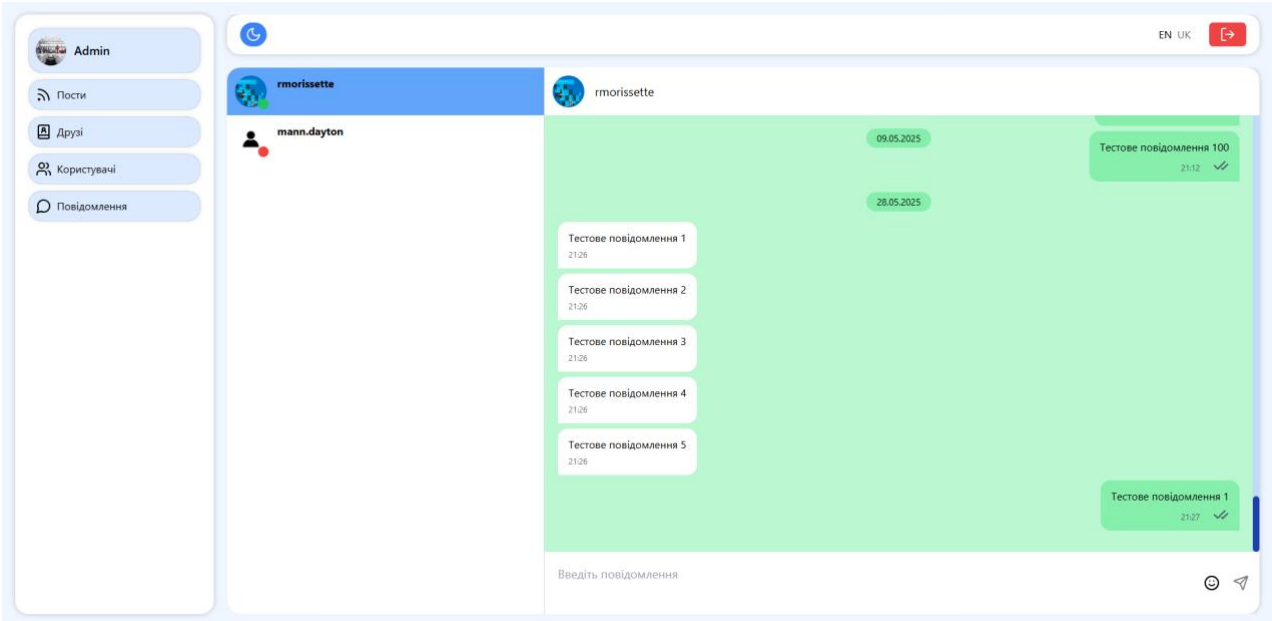

Рисунок 3.1 – Відображення сторінки з повідомленнями

3.2 Тестування та налагодження інформаційно-комп’ютерної системи

У процесі розробки інформаційної системи соціальної мережі було здійснено поетапне тестування основного функціоналу, перевірку поведінки системи в типових та прикордонних випадках, а також налагодження взаємодії між модулями бекенду та фронтенду. Особливу увагу було приділено функціоналу взаємодії між користувачами, обміну повідомленнями в реальному часі та забезпеченню коректної роботи WebSocket-підключень

Для перевірки сценаріїв надсилання, приймання та відхилення запитів у друзі було створено два тестові облікові записи. Один користувач авторизувався в одному браузері, другий – в іншому. Такий підхід дозволив імітувати реальну взаємодію між клієнтами системи без конфлікту сесій. Під час тестування було вручну перевірено правильність оновлення статусу відносин між користувачами, зміну кнопок в інтерфейсі залежно від поточного стану (запит надіслано, запит отримано, підтверджено, відхилено), а також реакцію на асинхронні події.

Цей спосіб дозволив швидко виявляти помилки, пов’язані з некоректною перевіркою статусу дружби та проблемами синхронізації. Було також виявлено декілька дрібних помилок логіки, зокрема: неправильне повернення статусу при взаємних запитах, що були усунені після доопрацювання контролерів API.

Для спрощення початкового тестування та уникнення ручного створення великої кількості акаунтів було застосовано можливості Laravel Seeder. Було створено фабрику користувача та сідер, який автоматично генерує велику кількість випадкових облікових записів із унікальними іменами та електронними адресами. Це дозволило миттєво створити тестову базу користувачів, яку надалі використовували для тестування системи пошуку, додавання в друзі, перегляду профілів та побудови постів у стрічках.

Тестування системи обміну повідомленнями здійснювалося у два етапи: генерація великої кількості повідомлень через консольну команду та перевірка роботи WebSocket-з’єднань вручну.

З метою зручного наповнення чату тестовими повідомленнями було створено Laravel-команду (php artisan chat:generate-messages {fromUserId} {toUserId} {count}), яка приймає три параметри: ID відправника, ID отримувача та кількість повідомлень. Після запуску команди відбувається генерація повідомлень зі змістом типу «Нове повідомлення N», які зберігаються безпосередньо в базу даних.

Це дозволяло протестувати інтерфейс перегляду чату, пагінацію старих та нових повідомлень, автоскрол до останнього повідомлення та поведінку при великому обсязі виводу. Однак, через те, що повідомлення створювалися напряму в БД, вони не викликали логіку контролера й відповідно – не ініціювали подію WebSocket. Через це перевірка реального часу та обробки подій (наприклад, з’явлення повідомлення в іншому вікні браузера) здійснювалася вручну шляхом входу в систему під двома різними користувачами та надсилання повідомлень через інтерфейс.

Такий підхід дозволив перевірити не лише відображення повідомлень, а й коректність підписки на приватні канали, надійність WebSocket-з’єднання, оновлення UI в реальному часі та правильну реакцію системи на події типу SentMessageEvent.

Окрім базової перевірки надсилання та прийому повідомлень, проводилось тестування логіки індикації непрочитаних повідомлень, що відіграє важливу роль в UX системи. Була реалізована логіка відображення кількості чатів із новими повідомленнями, а також підрахунок кількості непрочитаних повідомлень у конкретному чаті. В ході тестування перевірялося, чи правильно оновлюються ці значення під час надсилання нових повідомлень та переходу користувача до відповідного діалогу.

Особлива увага приділялася тестуванню поведінки інтерфейсу у випадках, коли в чаті накопичено велику кількість нових повідомлень (наприклад, понад 500). Було реалізовано автоматичне позиціонування на перше непрочитане повідомлення, що дозволяє користувачу швидко зорієнтуватися в новому контенті, не прокручуючи вручну всю історію.

Також проводилась перевірка коректності пагінації як для нових, так і для старих повідомлень. При досягненні межі viewport або при скролі до верхньої межі чату автоматично підвантажувалася попередня частина історії. Для цього реалізовано двосторонню пагінацію з урахуванням ID останнього повідомлення на екрані.

Окремо тестувалася логіка віднімання вже прочитаних повідомлень із загального лічильника, що дозволяло актуалізувати індикацію нових повідомлень у реальному часі. Таким чином забезпечено динамічне оновлення інтерфейсу при відкритті діалогу або перемиканні між чатами.

Ці перевірки дозволили гарантувати, що функціонал повідомлень працює стабільно навіть за високого навантаження, а також що UX лишається передбачуваним і комфортним для користувача незалежно від обсягу вхідної інформації.

На етапі, коли клієнтська частина ще не була реалізована, для тестування бекенду використовувався Postman. Це дало змогу перевірити роботу REST API – коректність відповідей, статус-кодів та бізнес-логіки – без потреби у UI. Через Postman тестувалися основні ендпоінти: автентифікація, заявки в друзі, повідомлення. Такий підхід допоміг надійно налагодити серверну логіку до інтеграції з фронтендом.

Розроблена інформаційна система передбачає розгортання як серверної, так і клієнтської частин. Для забезпечення стабільної роботи проєкту використовувалася технологія Docker. Це дозволило створити ізольоване середовище, в якому всі залежності були вже налаштовані, а запуск системи зводився до виконання кількох команд з офіційної документації проєкту.

Мінімальні вимоги до запуску серверної частини:

- наявність встановленого Docker і Docker Compose;
- підтримка Linux, Windows (WSL) або MacOS;
- вільні порти: 8000 (Laravel), 8080 (Reverb), 3306 (MySQL).

Для зручності запуску серверної частини було створено скрипт up.sh, який автоматизує запуск контейнерів Docker та відкриває інтерактивну сесію у бекенд-

контейнері. Під капотом цей скрипт виконує команду: docker compose up -d && docker exec -it backend bash. Після входу в контейнер розробник виконує скрипт start.sh, який запускає одночасно Laravel-сервер і Reverb-сервер: php artisan serve --host=0.0.0.0 & php artisan reverb:start.

Це дозволяє за один крок підняти повністю працездатне середовище з підтримкою WebSocket-з'єднань, без потреби в окремому налаштуванні процесів вручну. Такий підхід забезпечує високу швидкість розгортання системи для розробки, тестування та демонстрації функціональності, автоматизує стандартні дії при кожному запуску, мінімізує ризик людських помилок, гарантує коректну послідовність запуску сервісів і водночас робить процес доступним навіть для нових учасників команди.

3.3 Автоматизоване розгортання середовища розробки за допомогою Docker

У сучасній розробці програмного забезпечення однією з ключових потреб є забезпечення стабільного, повторюваного та ізольованого середовища, в якому програма буде розгортатися та виконуватися незалежно від особливостей операційної системи, апаратного забезпечення чи конфігурацій машини розробника. Для задоволення цих потреб було розроблено технологію Docker, яка стала стандартом де-факто в області контейнеризації застосунків.

Розроблена інформаційна система передбачає розгортання як серверної, так і клієнтської частин. Для забезпечення стабільної роботи проєкту використовувалася технологія «Docker – програмне забезпечення, яке дозволяє відокремити певну кількість ресурсів під контейнер та керувати ними» [12]. Контейнер у цьому випадку виступає як ізольована одиниця виконання, яка працює на базі ядра операційної системи хоста, але не містить повноцінного образу ОС, як у віртуальних машинах. Завдяки цьому контейнери запускаються дуже швидко, споживають мінімум ресурсів і забезпечують високу

портативність: один і той самий контейнер буде ідентично працювати на будь-якій системі, де встановлено Docker.

Контейнери створюються на основі образів, які збираються за допомогою спеціального конфігураційного файлу – Dockerfile. У ньому послідовно описуються інструкції, необхідні для створення середовища: яка версія інтерпретатора буде використана, які пакети потрібно встановити, яку директорію використовувати як робочу, які файли скопіювати тощо. Кожна інструкція у Dockerfile створює новий шар (layer), а всі шари в сукупності формують завершений образ, з якого потім створюється контейнер. Таким чином, Docker дозволяє стандартизувати процес збирання середовища, виключити залежність від локальних налаштувань та уникнути конфліктів версій.

Проте в реальних умовах сучасні веб-застосунки рідко складаються з одного сервісу. Як правило, існує потреба у взаємодії кількох компонентів – веб-сервера, бази даних, інтерфейсів адміністрування, систем черг тощо. Для спрощення керування багатьма контейнерами одночасно було створено інструмент Docker Compose, який «дозволяє визначати та керувати кількома взаємопов’язаними сервісами за допомогою єдиного конфігураційного файлу у форматі YAML» [13]. Він дозволяє описати структуру всієї системи – всі сервіси, їхні образи, змінні середовища, мережі, порти, томи – у вигляді одного YAML-файлу. Після цього, лише однією командою можна запустити всю інфраструктуру, де кожен контейнер буде створений, налаштований та підключений до інших згідно з описом.

Принцип роботи Docker Compose полягає в тому, що всі описані сервіси запускаються у рамках однієї віртуальної мережі, що забезпечує їхню взаємодію на рівні імен хостів. Наприклад, сервіс веб-сервера може звертатися до бази даних не за IP-адресою, а за ім’ям контейнера. Також можна визначити залежності між сервісами, які гарантують правильний порядок запуску – наприклад, база даних буде запущена раніше, ніж веб-додаток, який потребує підключення до неї.

Використання Docker і Docker Compose дозволяє стандартизувати процес розгортання, зменшити кількість помилок, пов’язаних із ручною конфігурацією, та забезпечити консистентність між середовищами розробки, тестування та продакшну. У проєктній роботі ця технологія виступає не просто як допоміжний інструмент, а як критично важливий компонент, що гарантує стабільність, масштабованість і ефективність життєвого циклу програмного забезпечення.

3.4 Застосування фреймворку Next.js в розробці веб-застосунку

У сучасному веб-розробництві ключовими викликами залишаються продуктивність, SEO-індексація, швидкість початкового завантаження сторінки, масштабованість, ефективне управління станом і можливість обробки логіки як на клієнті, так і на сервері. Відповіддю на ці потреби став «Next.js – це фреймворк React, який розширює його можливості додатковими функціями, включаючи рендеринг на стороні сервера та генерацію статичних веб-сайтів» [14]. Він надає зручну маршрутизацію, повну підтримку типізації, можливість обробки запитів на сервері, оптимізацію зображень, lazy loading, розділення коду та ще багато іншого «з коробки». Зокрема, «Next.js забезпечує міцну основу для SEO завдяки підтримці серверного рендерингу (SSR) та статичної генерації сайтів (SSG), які покращують сканування та індексацію контенту пошуковими системами» [15] .

Основна проблематика, яку вирішує Next.js:

– повільне початкове завантаження – вирішується за допомогою серверного рендерингу або статичної генерації, що дозволяє отримати готовий HTML ще до завантаження JS;

– слабка SEO-оптимізація в SPA – стандартний React не дозволяє ефективно індексувати сторінки пошуковими ботами. Next.js надає інструменти для SSR/SSG, що дозволяє ботам бачити повноцінний HTML;

– неефективна маршрутизація – замість ручного налаштування, як у React Router, Next.js використовує файлову структуру для створення маршрутів;

– складність виводу динамічних даних – розв’язується використанням Server Components, які дозволяють отримувати дані ще до рендеру.

Next.js підтримує кілька підходів до рендерингу сторінок, кожен з яких розрахований на певні сценарії використання та дозволяє досягти оптимального балансу між продуктивністю, динамічністю контенту та пошуковою оптимізацією. Статична генерація (Static Site Generation, або SSG) застосовується до сторінок із рідко змінюваним вмістом. У такому випадку HTML-файли створюються на етапі білду і можуть одразу віддаватися користувачу, що забезпечує максимальну швидкість завантаження. Цей підхід є ідеальним для лендингів, статей або документації. «SSG – це стратегія рендерингу, яка генерує HTML-розмітку під час збирання, а не на сервері чи клієнті. Завдяки SSG HTML-розмітка для кожної сторінки попередньо генерується та подається клієнту як набір статичних файлів» [16].

Якщо ж сторінка вимагає динамічного контенту, який залежить від параметрів запиту або стану користувача, доцільним є використання серверного рендерингу (Server-Side Rendering, або SSR). У цьому режимі HTML генерується на сервері під час кожного звернення, що дозволяє відобразити актуальні дані, не втрачаючи переваг SEO-індексації. «SSR допомагає першій сторінці вашої програми стати більш оптимізованою для SEO і збільшує швидкість завантаження» [17]. Для випадків, коли необхідно створити багатий інтерфейс із клієнтською взаємодією, а пошукова оптимізація не є критичною, використовується клієнтський рендеринг (Client-Side Rendering, або CSR). У такому випадку логіка та дані обробляються виключно у браузері користувача.

Особливе місце посідає інкрементальна статична генерація (Incremental Static Regeneration, або ISR) – гібридний варіант SSG, який дозволяє оновлювати кешовані сторінки у фоні без повного перевипуску застосунку. Це рішення є надзвичайно ефективним для проєктів із великим обсягом контенту, що змінюється частково або з різною частотою. «Даний метод поєднує переваги двох попередніх. Не завжди потрібно генерувати сторінку при кожному запиті, та й

заздалегідь нагенеровані сторінки не є рішенням, якщо контент час від часу оновлюється» [18].

З нових версій Next.js була запроваджена концепція серверних компонентів (Server Components), яка кардинально змінила підхід до побудови інтерфейсів. Ці компоненти виконуються виключно на сервері, не потрапляючи у JavaScript-бандл користувача. Такий підхід дозволяє безпечно обробляти дані, уникати витоку конфіденційної інформації та зменшувати обсяг коду, що завантажується у браузер. Крім того, серверні компоненти можуть безпосередньо виконувати запити до бази даних або API без необхідності створювати окремі контролери.

У той же час клієнтські компоненти залишаються необхідними для частин інтерфейсу, що взаємодіють із користувачем: обробка кліків, стану, анімацій тощо. Їх використання активується директивою «use client», яка дозволяє Next.js визначити, що компонент повинен бути оброблений у браузері.

Ще однією важливою особливістю Next.js є його потужна система маршрутизації, яка базується на файловій структурі та не потребує додаткових налаштувань. Фреймворк підтримує як статичні маршрути (наприклад, /about), так і динамічні (наприклад, /user/[id]), що дозволяє створювати гнучкі інтерфейси з параметризованими сторінками. Динаміка досягається завдяки синтаксису у вигляді квадратних дужок, який автоматично перетворюється у відповідні шляхи при генерації маршруту. Також підтримується вкладеність (гніздові маршрути), що дає змогу будувати багаторівневу ієрархію сторінок із чіткою логічною структурою.

Нові концепції, такі як layout-и, дають змогу визначати спільні області (напр. хедер, футер, навігацію), які автоматично підключаються до певних груп маршрутів. Паралельні маршрути (parallel routes) дозволяють рендерити незалежні частини інтерфейсу одночасно, що є корисним для комплексних розділів. А перехоплюючі маршрути (intercepting routes) застосовуються у випадках, коли потрібно показати модальне вікно поверх поточної сторінки, зберігаючи маршрут і стан інтерфейсу.

3.5 Розробка бази даних

Процес проєктування бази даних відіграє ключову роль у побудові надійної інформаційної системи, оскільки від структури схеми та ефективності запитів залежить стабільність, масштабованість і швидкодія всього застосунку. У межах розробки даної соціальної мережі для збереження й обробки даних було використано систему керування базами даних MySQL, яка є потужним рішенням, що поєднує в собі швидкість, гнучкість і простоту керування, роблячи її ідеальним вибором для різноманітних додатків і сайтів [19]. Для візуального керування таблицями було використано інструмент phpMyAdmin, що значно спростив початкову перевірку структури та взаємозв'язків між сутностями.

Безпосереднє створення, зміна та підтримка схеми бази даних реалізовувалась за допомогою вбудованої в Laravel системи міграцій. Такий підхід дозволяє описувати структуру таблиць на рівні коду, що дає змогу автоматизувати процес розгортання схеми на будь-якому середовищі. Міграції створювались щоразу, коли виникала потреба у збереженні нових типів даних – наприклад, після розробки модуля повідомлень або системи дружби.

На початковому етапі було створено базові таблиці: users, posts, friends, messages, chats та пов'язані допоміжні структури. У випадках, коли виявлялася потреба у додаткових полях (наприклад, додавання статусу запиту чи часової мітки останньої активності), до проєкту додавались нові міграції з відповідними змінами. Завдяки системі накочування та відкочування міграцій (php artisan migrate, rollback), процес еволюції структури БД відбувався керовано, без втрати даних та з можливістю точкового контролю.

Взаємодія з базою даних здійснювалась за допомогою Eloquent ORM, яка надає зручний та декларативний API для виконання типових SQL-запитів без необхідності ручного написання SQL-інструкцій. Наприклад, для отримання всіх постів користувача не потрібно писати запит типу SELECT * FROM posts WHERE user_id = ?, достатньо викликати метод $user->posts. Така абстракція підвищує читабельність коду, зменшує ймовірність помилок у синтаксисі та

дозволяє легко працювати з пов’язаними моделями. «Eloquent – вбудований об’єктно-реляційний маппер (ORM) у Laravel. Він дає змогу працювати з базами даних із використанням об’єктів і методів» [20].

Для підвищення ефективності та зменшення дублювання логіки, багато SQL-операцій були винесені в окремі методи моделей. Наприклад, у моделі User було реалізовано метод friends(), який повертає список друзів поточного користувача. Це дозволило централізувати логіку роботи з даними, спростити тестування й пришвидшити написання нового функціоналу.

3.6 SEO інформаційно-комп’ютерної системи

Одним із важливих аспектів сучасної веб-розробки є забезпечення SEO-оптимізації (Search Engine Optimization) – процесу підготовки ресурсу до ефективної індексації та ранжування пошуковими системами. Хоча класичні соціальні мережі здебільшого орієнтовані на зареєстрованих користувачів і взаємодію всередині системи, наявність публічних сторінок, таких як головна, інформаційні блоки чи профілі, вимагає належної оптимізації для доступу до них через пошукові сервіси.

Важливою частиною SEO є логічна структура URL-адрес та їх доступність без зайвих параметрів або хешів. У розробленій системі передбачено чисті та читабельні адреси, зокрема /profile/username замість складних комбінацій з ідентифікаторами. Такий підхід не лише покращує досвід користувачів, а й підвищує довіру пошукових систем до ресурсу. Чітка структура маршрутів є додатковим сигналом для індексації, а також дозволяє краще контролювати сторінки, доступні до перегляду ззовні. Завдяки цьому SEO-оптимізація стала не лише технічним елементом, а й частиною загальної стратегії покращення юзабіліті платформи.

У межах реалізації SEO-оптимізації було сформовано набір основних мета-даних (ліст. 3.11), які описують загальні характеристики проєкту та забезпечують

правильне відображення ресурсу в пошукових системах і на сторонніх платформах.

Лістинг 3.11 – Головна частина налаштування Meta тегів

```
export const metadata: Metadata = {
    title: "Friendify – Social Media",
    description: "Social media app for diploma project developed by Nazar
Melnychuk",
    icons: [
       { rel: "icon", type: "image/png", sizes: "96x96", url:
"/favicon/favicon-96x96.png" },
       { rel: "icon", type: "image/svg+xml", url: "/favicon/favicon.svg"
},
       { rel: "shortcut icon", url: "/favicon/favicon.ico" },
       { rel: "apple-touch-icon", sizes: "180x180", url: "/favicon/apple-
touch-icon.png" },
    ],
    manifest: "/favicon/site.webmanifest",
    openGraph: {
       title: "Friendify – Social Media",
       description: "Social media app for diploma project developed by
Nazar Melnychuk",
       images: [ OgImage.src ],
    },
}
```

кінець лістингу 3.11

Для кожної сторінки застосунку було налаштовано мета-теги, що описують заголовок, опис і пов’язані ресурси. На основі статичної конфігурації у Next.js було сформовано об’єкт metadata, що містить ключові елементи: назву застосунку, короткий опис і набір іконок для різних платформ і пристроїв. Зокрема, було додано стандартні іконки для браузерів, Apple-пристроїв та спрощений manifest-файл для підтримки PWA-функціоналу. «PWA, або Progressive Web Apps – це вебдодатки, які поєднали в собі все найкраще з вебсайтів і мобільних додатків. Схожі на нативні за своїм функціоналом, вони працюють на будь-якому пристрої, де є веббраузери» [21].

Важливим доповненням стали Open Graph-теги, які забезпечують правильне відображення сторінок при поширенні посилань у соціальних мережах та месенджерах. У них було вказано заголовок, опис і зображення, що збагачують картку посилання при його попередньому перегляді в сторонніх сервісах. «Протокол Open Graph був розроблений фахівцями Facebook для покращення візуальної привабливості прев'ю вебсторінок у соціальних мережах. Завдяки йому посилання на сайт виглядають привабливіше, і це збільшує кількість кліків на них» [22].

Окрім мета-тегів, у межах SEO-налаштувань було створено файл robots.txt, який дозволяє пошуковим системам індексувати весь сайт, що зафіксовано через директиву. «Файл robots.txt – це текстовий документ, який використовують власники сайту для взаємодії з пошуковим роботом під час індексації сайту. Robots.txt розміщується в кореневому каталозі сайту та містить інструкції для пошукових роботів» [23]. Файл було розміщено в кореневому каталозі застосунку, що відповідає вимогам пошукових систем до його розташування. У ньому вказано дозвіл на сканування всіх сторінок ресурсу, що сприяє повній індексації публічного контенту.

Для полегшення обходу сторінок пошуковими ботами було також згенеровано сайтмап (sitemap.xml), що включає всі основні публічні сторінки проєкту. Особливістю реалізації стало те, що у sitemap автоматично додаються посилання на всі існуючі профілі користувачів, що дозволяє пошуковим системам виявляти та індексувати профілі без додаткових дій з боку адміністратора. Це сприяє підвищенню видимості контенту користувачів у пошукових системах і покращує загальне просування платформи. «Sitemap – це файл, у якому вказано всі сторінки сайту, доступні для індексації пошуковими системами. Він допомагає пошуковим роботам швидше знаходити та індексувати нові або оновлені сторінки» [24].

Таким чином, завдяки налаштованим мета-даним, Open Graph-тегам, конфігурації robots.txt та динамічно згенерованому sitemap.xml, система повністю готова до коректної індексації пошуковими системами.

3.7 Розробка документації для програмного продукту

Для забезпечення коректної роботи програмного продукту необхідно підготувати середовище, яке відповідає мінімальним технічним вимогам.

Для серверної частини (Laravel, Reverb, MySQL у Docker-контейнерах):

– операційна система: Windows (WSL2), Linux або MacOS;
– docker версії 24.x або вище;
– docker Compose;
– вільні порти: 8000 (Laravel), 6001 (Reverb), 3306 (MySQL);
– рекомендована апаратна конфігурація: CPU – 4 ядра, RAM – 8 ГБ.

Для клієнтської частини (Next.js):

– node.js версії 22.14.0;
– NPM або Yarn;
– вільний порт: 3000.

Для зручності розгортання інформаційної системи до кожного з репозиторіїв – як фронтенду, так і бекенду – було додано README-файли, які містять детальні інструкції щодо встановлення, налаштування та запуску проєкту. У цих інструкціях покроково описані всі необхідні дії: від встановлення залежностей до запуску серверів і підключення до бази даних. Додатково в обох репозиторіях передбачено файли .env.example, які слугують шаблонами для налаштування змінних середовища. На основі цих файлів адміністратор може швидко підготувати середовище, вказавши необхідні параметри: шляхи до API, ключі доступу, конфігурацію портів та інші важливі налаштування. Таке структурування дозволяє мінімізувати ймовірність помилок під час ініціалізації проєкту новими розробниками.

Фронтенд-частина побудована на базі Next.js 15, для коректної роботи якого необхідно встановити Node.js версії 22.14.0. Після клонування репозиторію користувачу потрібно виконати встановлення залежностей через команду npm install, а потім запустити локальний сервер розробки командою npm run dev. Усі

налаштування для підключення до API і WebSocket-конфігурації задаються у файлі .env.local. Фронтенд за замовчуванням доступний на порту 3000, або через Nginx, якщо використовується проксирування доменів. Крім того, структура фронтенду організована згідно з принципами Feature-Sliced Design, що спрощує масштабування і підтримку коду.

Бекенд-частина реалізована на базі Laravel та Reverb для роботи з WebSocket. Розгортання серверної частини автоматизовано за допомогою спеціального скрипта up.sh, який запускає побудову та запуск контейнерів Docker і одразу відкриває сесію з PHP-контейнером. Після запуску ./up.sh потрібно виконати ще один скрипт start.sh безпосередньо всередині контейнера. Цей скрипт запускає Laravel-сервер та Reverb WebSocket-сервер, роблячи бекенд доступним на порту 8000, а WebSocket – на порту 6001. База даних MySQL також розгортається в Docker-контейнері на порту 3306 і додатково phpMyAdmin на порті 8081. Таким чином, повноцінне середовище для тестування і розробки створюється автоматично без потреби в ручному налаштуванні інфраструктури.

Окрім цього, у проєкті передбачено попередньо налаштований конфігураційний файл Nginx, який дозволяє проксирувати запити до фронтенду, API та WebSocket. Для коректної роботи цієї конфігурації достатньо додати локальні доменні імена до файлу hosts операційної системи. «Nginx – це вебсервер і проксі-сервер з відкритим вихідним кодом, який здатен обробляти великі навантаження та забезпечувати високу швидкодію» [25]. Після цього система буде доступною за звичними адресами замість використання локальних портів, що спрощує тестування і моделює умови продуктивного розгортання.

Після цього система буде доступною за звичними адресами замість використання локальних портів, що спрощує тестування і моделює умови продуктивного розгортання.

Таким чином, розроблена система супроводжується документацією, шаблонами налаштувань і готовими скриптами для розгортання. Це дозволяє значно спростити процес встановлення та забезпечити стабільну роботу як у локальному, так і в серверному середовищі.

Висновки до розділу 3

У третьому розділі було проведено комплексну практичну реалізацію інформаційно-комп'ютерної системи відповідно до раніше визначених вимог та проєктних рішень. У процесі реалізації застосовано сучасні веб-технології та інструменти, що забезпечили високу продуктивність, гнучкість та масштабованість програмного продукту.

Практичне впровадження охоплювало розробку як клієнтської частини на основі Next.js, так і серверної частини на базі Laravel із підтримкою обміну повідомленнями в реальному часі через WebSocket. Значну увагу приділено архітектурному підходу Feature-Sliced Design, що дозволив структуровано організувати код і забезпечити ізольовану відповідальність окремих модулів. Це створило надійне підґрунтя для подальшого масштабування та підтримки проєкту.

У ході розробки було реалізовано повноцінну систему управління даними з використанням MySQL та ORM Eloquent, що забезпечило зручну роботу з базою даних на всіх етапах – від створення схеми через міграції до обробки запитів через моделі. Тестування системи здійснювалося як автоматизованими інструментами, так і ручними перевірками функціональних сценаріїв, що дозволило виявити й усунути помилки ще на етапі розробки.

Було впроваджено базові механізми захисту, включно з хешуванням паролів, авторизацією через Laravel Sanctum, контролем доступу до приватних WebSocket-каналів, захистом від CSRF-атак і налаштуванням конфігураційного середовища через .env-файли. Реалізовано SEO-оптимізацію через формування мета-тегів, налаштування robots.txt та автоматичну генерацію sitemap.xml, що дозволяє пошуковим системам індексувати публічні сторінки і профілі користувачів.

Окремо було розроблено супровідну документацію у вигляді README-файлів і шаблонів конфігурацій, що спрощують розгортання системи на будь-якому серверному або локальному середовищі. Автоматизовані скрипти up.sh та

start.sh дозволяють запускати серверну частину без зайвої ручної конфігурації. Додатково підготовлено конфігурацію для Nginx, яка забезпечує доступ до системи через локальні доменні імена, що моделює умови реального серверного розгортання.

У межах розробки, було реалізовано функціонал обміну повідомленнями в реальному часі. Результати роботи свідчать про повну технічну готовність системи до впровадження та подальшого розвитку, забезпечуючи надійну основу для масштабування, просування та використання в реальному середовищі.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи досягнуто поставлену мету – розроблено інформаційну систему соціальної мережі, що забезпечує ефективну взаємодію користувачів, обмін повідомленнями в реальному часі, управління зв’язками та публікаціями.

На першому етапі було здійснено аналіз існуючих соціальних мереж, таких як Facebook, Instagram та LinkedIn, з метою виявлення сильних і слабких сторін реалізації функціоналу взаємодії між користувачами. Особливу увагу приділено таким аспектам, як наявність стрічки новин, чати, приватність даних, гнучкість у налаштуваннях і комфортність інтерфейсу. Отримані висновки лягли в основу функціонального плану нової системи.

У процесі проєктування архітектури додатку було створено UML-діаграми, які відобразили основні компоненти системи та взаємозв’язки між ними. Це дозволило формалізувати логіку роботи застосунку, спростити комунікацію між розробниками та забезпечити прозорість у реалізації функціоналу.

У межах практичної реалізації створено функціонал додавання, підтвердження та відхилення запитів у друзі. Логіка побудована таким чином, щоб охопити всі можливі стани відносин між користувачами, забезпечуючи зрозумілий і надійний механізм управління контактами в межах платформи.

Розроблено механізм відстеження онлайн-статусів користувачів, який дозволяє в реальному часі дізнаватися про наявність співрозмовника в мережі. Це сприяє ефективнішому плануванню взаємодії та покращує загальний користувацький досвід.

Система підтримує можливість створення постів із текстовим вмістом та вподобанням, що формує динамічну стрічку новин. Такий функціонал реалізовано з урахуванням зручності читання, гнучкого відображення елементів і швидкої взаємодії з публікаціями.

Важливою частиною проєкту стала інтеграція чату, який працює через WebSocket-з’єднання. Це забезпечило передачу повідомлень без затримок, а

також реагування системи на події в реальному часі, що критично для сучасних соціальних платформ.

Реалізовано функціонал перегляду та редагування профілю, де користувач може змінювати своє ім’я, зображення, переглядати власні пости та дані. Це забезпечує базовий рівень персоналізації і дозволяє користувачеві ефективно керувати своєю присутністю в системі.

Розроблено систему авторизації та реєстрації з використанням Laravel Sanctum, що забезпечує захищене підключення до системи за допомогою токенів. Реалізація передбачає перевірку прав доступу через middleware, що дозволяє працювати з API без використання класичних сесій.

Інтерфейс додатку доповнено можливістю перемикання теми (світла/темна), що забезпечує комфорт користування як удень, так і в темний час доби. Це рішення відповідає сучасним очікуванням користувачів і підвищує інклюзивність застосунку.

Окрему увагу приділено автоматизації розгортання через Docker і підготовлені скрипти запуску. Це дозволяє підготувати повне середовище роботи всього за кілька команд без складних ручних налаштувань. Для зручності доступу до системи налаштовано проксіювання через Nginx і використання локальних доменів.

Проведено SEO-оптимізацію через налаштування мета-тегів, створення robots.txt та автоматичне формування sitemap.xml, що забезпечує індексацію сторінок користувачів пошуковими системами. Це дозволяє платформі бути видимою в пошуковому середовищі та розширювати охоплення.

Таким чином, усі поставлені завдання виконано в повному обсязі. Створений веб-додаток є прикладом практичного застосування сучасних веб-технологій і може бути використаний як основа для запуску реального проєкту або розширений відповідно до нових вимог.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Право на приватність. Що може замінити Google і Facebook? BBC News Україна. BBC News Україна. URL: https://www.bbc.com/ukrainian/features-51236114 (дата звернення: 28.05.2025).

2. Учасники проектів Вікімедіа. Unified Modeling Language – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Unified_Modeling_Language (дата звернення: 03.04.2025).

3. Учасники проектів Вікімедіа. Діаграма прецедентів – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Діаграма_прецедентів (дата звернення: 06.04.2025).

4. Учасники проектів Вікімедіа. Діаграма класів – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Діаграма_класів (дата звернення: 07.04.2025).

5. Учасники проектів Вікімедіа. Діаграма діяльності – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Діаграма_діяльності (дата звернення: 10.04.2025).

6. Учасники проектів Вікімедіа. Діаграма послідовності – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Діаграма_послідовності (дата звернення: 14.04.2025).

7. Сайти та додатки на NEXT.JS: розробка швидких рішень – Brander. Створення і розробка інтернет-магазинів від Brander. URL: https://brander.ua/technologies/nextjs (дата звернення: 14.04.2025).

8. Feature Sliced Design overview. Feature Sliced Design. URL: https://feature-sliced.design/docs/get-started/overview (date of access: 16.04.2025).

9. Home – React Hook Form – Simple React forms validation. URL: https://www.react-hook-form.com/. (date of access: 17.04.2025).

10. Girard G. Інтеграція React Hook Form з EmailJs і Zod. https://www.tempmail.us.com. URL: https://www.tempmail.us.com/uk/react-hook-form/Інтеграція-react-hook-form-з-emailjs-i-zod-validation (date of access: 18.04.2025).

11. Єленська В. WebSockets: навіщо потрібні та як з ними працювати ProIT. ProIT: медіа для профі в IT. URL: https://proit.ua/websockets-navishcho-potribni-ta-iak-z-nimi-pratsiuvati-2 (дата звернення: 20.04.2025).

12. Простими словами про контейнеризацію та Docker. IT Education Center Blog. URL: https://itedu.center/ua/blog/devops/docker-prostimi-slovami-pro-kontejnerizaciyu/?srsltid=AfmBOoqfstIzBrUIV00xKTYlUBWeJQu9eEr45Zj0ViNhMejIptNqOnmL (дата звернення: 21.04.2025).

13. Docker Compose для DevOps – спрощення роботи з контейнерами. IT Education Center Blog. URL: https://itedu.center/ua/blog/review/what-is-docker-compose (дата звернення: 22.04.2025).

14. Учасники проектів Вікімедіа. Next.js – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Next.js (дата звернення: 24.04.2025).

15. Next.js SEO. Ranktracker The all-in-one platform for effective SEO. URL: https://www.ranktracker.com/uk/blog/next-js-seo/ (date of access: 27.04.2025).

16. Gallego R. Next.js Rendering Strategies: SSR, SSG, and ISR Compared. Hybrid Heroes Blog. URL: https://hybridheroes.de/blog/2023-05-31-next-js-rendering-strategies/ (date of access: 27.04.2025).

17. PNN Soft. Компанія по розробці програмного забезпечення: WEB розробка, веб-сайти – PNN. URL: https://pnn.com.ua/ua/blog/detail/server-side-rendering-in-react-benefits-challenges-peculiarities (дата звернення: 29.04.2025).

18. Порівнюємо способи генерації сторінок: CSR, SSR, SSG, ISR. Гайд на основі стеку React. URL: https://dou.ua/forums/topic/41585/ (дата звернення: 03.05.2025).

19. Система управління базами даних MySQL. Lemon school. URL: https://lemon.school/blog/systema-upravlinnya-bazamy-danyh-mysql (дата звернення: 03.05.2025).

20. Переваги фреймворку Laravel. Створення і розробка інтернет-магазинів від Brander. URL: https://brander.ua/blog/perevahy-freymvorku-laravel (дата звернення: 08.05.2025).

21. Що таке PWA: детальна інструкція зі зразками коду. Asabix. URL: https://asabix.com.ua/what-is-pwa-comprehensive-guide-with-code-samples/ (дата звернення: 08.05.2025).

22. Сапінська О. Як працювати з Open Graph. Netpeak Journal. Медіа про інтернет-маркетинг та онлайн-бізнес у деталях. URL: https://netpeak.net/uk/blog/yak-pratsyuvati-z-open-graph/ (дата звернення: 10.05.2025).

23. Robots.txt. Хорошоп. URL: https://horoshop.ua/ua/glossary/robots.txt/ (дата звернення: 15.05.2025).

24. Build and Submit a Sitemap – Google Search Central – Documentation Google for Developers. Google for Developers. URL: https://support.google.com/webmasters/answer/183668?hl=uk (date of access: 13.05.2025).

25. Nginx – ommarketing. URL: https://ommarketing.com.ua/technology-uk/nginxs-uk/ (дата звернення: 15.05.2025).