МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ВЕБСАЙТУ СОЦІАЛЬНОЇ МЕРЕЖІ ЗАСОБАМИ REACT, LARAVEL, POSTGRESQL»**

Виконав: студент 4 курсу, групи 6.1211-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
В.І. Самолук
(ініціали та прізвище)
Керівник доцент кафедри програмної інженерії, к.ф.-м.н. Кривохата А.Г.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Решевська К.С.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Самолуку Володимиру Ігоровичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка вебсайту соціальної мережі засобами React, Laravel, PostgreSQL

керівник роботи Кривохата Анастасія Григорівна, к.ф.-м.н.
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » грудня 2024 року № 2001-с

2. Строк подання студентом роботи 02.06.2025 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Проєктування та розробка вебзастосунку соціальних мереж.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 26.12.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	09.01.2025	
2.	Збір вихідних даних.	17.01.2025	
3.	Обробка методичних та теоретичних	14.03.2025	
	джерел.		
4.	Розробка першого та другого розділу.	23.04.2025	
5.	Розробка третього розділу.	19.05.2025	
6.	Оформлення та нормоконтроль	26.05.2025	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	19.06.2025	

Студент _____ В.І. Самолук
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.Г. Кривохата
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка вебсайту соціальної мережі засобами React, Laravel, PostgreSQL»: 60 с., 24 рис., 13 джерел.

ВЕБСАЙТ, КОМПОНЕНТ, КОНТРОЛЕР, МОДЕЛЬ, СОЦІАЛЬНА МЕРЕЖА, ФРЕЙМВОРК, LARAVEL, POSTGRESQL, REACT.

Об’єкт дослідження – процес розробки вебсайту соціальної мережі.

Предмет дослідження – архітектура та ключові компоненти вебзастосунку соціальної мережі, що базується на платформах React та Laravel із використанням мов програмування JavaScript і PHP, а також СКБД PostgreSQL для зберігання даних.

Мета роботи – розробка вебсайту соціальної мережі за допомогою платформ React, Laravel та PostgreSQL.

Методи дослідження – методи збору та аналізу вимог до програмного забезпечення, методи моделювання, проєктування, конструювання, REST-архітектура, реляційне проєктування баз даних та тестування програмного забезпечення.

В кваліфікаційній роботі проведено аналіз предметної області, сформульовано перелік функціональних і нефункціональних вимог до системи. На основі отриманих результатів обрано відповідні технології та спроєктовано архітектуру вебсайту з використанням UML-діаграм: варіантів використання, послідовностей, компонентів, розгортання та взаємодії елементів системи.

Реалізовано серверну частину з використанням фреймворку Laravel, клієнтську частину – засобами React та відповідних бібліотек. У результаті створено вебсайт соціальної мережі, що підтримує реєстрацію користувачів, авторизацію, додавання друзів, створення публікацій, коментарів та отримання сповіщень.

SUMMARY

Bachelor’s qualifying paper “Development of a Social Network Website Using React, Laravel, PostgreSQL” 60 pages, 24 figures, 13 references.

WEBSITE, COMPONENT, CONTROLLER, MODEL, SOCIAL NETWORK, FRAMEWORK, LARAVEL, POSTGRESQL, REACT.

The object of the study is the process of developing a social networking website

The subject of the study is the architecture and key components of a social networking web application that is built using the React and Laravel platforms, the JavaScript and PHP programming languages, and the PostgreSQL database management system for data storage.

The purpose of this work is to develop a social networking website using React, Laravel, and PostgreSQL.

The research methods used include collecting and analyzing software requirements, modeling, designing, developing, applying REST architecture, designing relational databases, and software testing.

This qualification project includes an analysis of the subject domain and a formulation of both functional and non-functional system requirements. Based on the results, appropriate technologies were selected, and the system architecture was designed using UML diagrams: use case, sequence, component, deployment, and system interaction diagrams.

The server-side of the application was implemented using the Laravel framework, while the client-side was developed with React and relevant libraries. As a result, a social networking website was created that supports user registration, authentication, friend requests, post and comment creation, and notification delivery.

ЗМІСТ

Завдання на кваліфікаційну роботу……..……..2
Реферат 4
Summary 5
Вступ..... 7
1 Огляд теоретичних властивостей та планування проєкту 9
1.1 Аналіз сучасних засобів розробки вебсайтів соціальних мереж..... 9
1.2 Визначення функціональних та нефункціональних вимог..... 10
1.3 Огляд фреймворка Laravel та його можливостей 12
1.4 Огляд фреймворку React та його можливостей 14
1.5 Порівняння вебзастосунку з іншими аналогами..... 15
1.6 Висновки до розділу 1 17
2 Проєктування вебзастосунку соціальних мереж 18
2.1 Проєктування інтерфейсу користувача 18
2.2 Структура бази даних вебзастосунку..... 21
2.3 Проєктування архітектури вебзастосунку..... 26
2.4 Проєктування серверної частини 31
2.5 Висновки до розділу 2 36
3 Програмна реалізація та тестування 37
3.1 Реалізація серверної частини 37
3.2 Реалізація клієнтської частини 44
3.3 Налаштування бази даних PostgreSQL 52
3.4 Тестування компонентів та функціоналу 53
3.5 Висновки до розділу 3 57
Висновки 59
Перелік посилань..... 60

ВСТУП

Сучасний розвиток засобів створення вебзастосунків призводить до того, що з’являються нові можливості для ефективної розробки та реалізації соціальних мереж у вебсередовищі.

Прикладом таких систем є популярні соціальні платформи, такі як Facebook, Instagram, Twitter, де користувачі можуть створювати профілі, публікувати контент, спілкуватися з друзями, створювати спільноти, обмінюватися повідомленнями та взаємодіяти через коментарі та лайки.

Метою кваліфікаційної роботи є проєктування та реалізація вебзастосунку соціальної мережі з використанням фреймворку React для клієнтської частини, Laravel для серверної логіки та PostgreSQL як системи керування базами даних.

Для досягнення поставленої мети сформульовано такі задачі:

– проаналізувати сучасні соціальні платформи та технології веброзробки;
– сформулювати функціональні та нефункціональні вимоги до системи;
– спроєктувати архітектуру застосунку на основі моделі MVC;
– реалізувати основні компоненти інтерфейсу користувача за допомогою React;
– розробити серверну частину на Laravel з побудовою REST API;
– реалізувати систему автентифікації та авторизації з розмежуванням прав доступу;
– виконати тестування компонентів та проведено інтеграційне тестування застосунку.

Структурно робота складається з трьох розділів. У першому розділі проаналізовано сучасні соціальні мережі, їхні функціональні можливості та архітектурні підходи, а також обґрунтовано вибір фреймворку React для

клієнтської частини, Laravel для реалізації серверної логіки та PostgreSQL як системи управління базами даних.

У другому розділі виконано аналіз функціональних і нефункціональних вимог до вебзастосунку, спроєктовано концептуальну модель системи із застосуванням діаграм UML розроблено макети інтерфейсу, логічну архітектуру клієнтської та серверної частин, а також структуру бази даних.

У третьому розділі реалізовано основні компоненти клієнтської частини з використанням React, розроблено серверну логіку на основі Laravel, впроваджено систему автентифікації та авторизації з використанням JWT, а також проведено тестування окремих компонентів і інтеграційне тестування вебзастосунку.

1 ОГЛЯД ТЕОРЕТИЧНИХ ВЛАСТИВОСТЕЙ ТА ПЛАНУВАННЯ ПРОЄКТУ

1.1 Аналіз сучасних засобів розробки вебсайтів соціальних мереж

При розробці соціальних мереж використовуються технологічні рішення, що забезпечують високу продуктивність, масштабованість та зручність користування. Технології розробки повинні підтримувати велику кількість користувачів одночасно, забезпечувати швидку доставку контенту та надавати різноманітні можливості для взаємодії [1].

До основних технологічних тенденцій належить використання архітектури SPA (Single Page Application), API-First підходу, впровадження вебфункціональності в режимі реального часу та оптимізація для мобільних пристроїв.

Для фронтенд розробки використовуються JavaScript фреймворки та бібліотеки, що дозволяють створювати динамічні та інтерактивні користувацькі інтерфейси. React.js часто обирається для розробки соціальних мереж через компонентну архітектуру та ефективне управління станом додатку [1].

React надає віртуальний DOM для швидкого відображення великих списків постів, можливість повторного використання компонентів для різних частин інтерфейсу та широку екосистему додаткових бібліотек. Цей фреймворк використовується у Facebook, Instagram та WhatsApp, що демонструє його придатність для високонавантажених додатків.

Серед альтернатив можна розглянути Vue.js через простоту навчання та Angular через потужну архітектуру для корпоративних рішень. Вибір фреймворку залежить від розміру команди, складності проєкту та специфічних вимог до функціональності [2].

Для серверної частини вебзастосунків соціальних мереж потрібні технології, що можуть ефективно обробляти великі обсяги запитів та

управляти складною бізнес-логікою. Laravel як PHP фреймворк надає інструментарій для створення RESTful API та GraphQL endpoints [2].

Laravel включає ORM (Eloquent) для роботи з базами даних, систему аутентифікації та авторизації, middleware-функції для обробки HTTP запитів та систему черг для асинхронної обробки завдань. Фреймворк також містить інструменти для тестування, кешування та логування, що важливо для підтримки якості коду в великих проєктах.

Інші серверні рішення включають Node.js з Express або Fastify для JavaScript, Python з Django або FastAPI, та Go з високопродуктивними HTTP серверами. Вибір технології залежить від експертизи команди та специфічних вимог до продуктивності [2].

PostgreSQL розглядається як одне з рішень для зберігання даних соціальних мереж через надійність, розширені можливості та підтримку як реляційних. База даних підтримує JSON типи даних, повнотекстовий пошук, геопросторові запити та складні індекси, що підходить для різноманітного інформаційного наповнення соціальних мереж [2].

Альтернативні рішення включають MySQL для простіших проєктів, MongoDB для документо-орієнтованих даних та Redis для кешування та real-time функцій. Часто використовується комбінація кількох типів баз даних залежно від специфічних потреб різних частин додатку.

Для забезпечення взаємодії в режимі реального часу в соціальних мережах використовуються технології WebSocket та Server-Sent Events, що дозволяють миттєво оновлювати чати, коментарі та стрічку новин без постійних запитів до сервера [3].

1.2 Визначення функціональних та нефункціональних вимог

Усі вимоги до програмного продукту можна розділити на дві основні категорії: функціональні та нефункціональні. Функціональні вимоги описують

«що» саме повинна робити система, тобто вони визначають основну поведінку та функціональність. Натомість нефункціональні вимоги вказують, «як» система має працювати, встановлюючи якісні характеристики, такі як продуктивність, надійність, безпека тощо. Незважаючи на те, що нефункціональні вимоги не описують безпосередньо функціональність, вони мають істотний вплив на якість користувацького досвіду [3].

Функціональні вимоги визначають основні функції, які має виконувати вебсайт соціальної мережі. Вони охоплюють взаємодію між користувачем і системою, визначають, як система поводиться у відповідь на різні дії, і що саме вона повинна робити.

Наведемо основі функціональні вимоги до вебзастосунку соціальних мереж:

- реєстрація та авторизація користувачів із валідацією введених даних;
- створення, редагування та видалення публікацій (текст, зображення, відео);
- можливість ставити вподобайки, коментувати та поширювати пости;
- функціонал пошуку користувачів, постів та хештегів;
- сторінка профілю користувача з можливістю змінювати особисті дані;
- додавання друзів або підписок;
- вбудований чат для обміну повідомленнями в реальному часі;
- панель адміністратора з можливістю редагування контенту та управління користувачами.

Функціональні вимоги формують основу проєкту програмного забезпечення та керують командою розробників у створенні бажаних функцій.

Нефункціональні вимоги описують якісні характеристики системи, що визначають її ефективність, стабільність та зручність використання. Вони не відповідають на питання «що система повинна робити», але чітко формулюють обмеження та критерії, які впливають на те, як саме функціональність буде реалізована [4].

Наведемо основні нефункціональні вимоги до системи:

– продуктивність: вебзастосунок має забезпечувати швидке та стабільне завантаження сторінок при типовому користувацькому навантаженні, без помітних затримок у відображенні основного вмісту;
– безпека: обов’язкове використання шифрування паролів (bcrypt), захист від XSS та CSRF атак, а також використання JWT для аутентифікації;
– масштабованість: система повинна підтримувати збільшення навантаження на 100% без необхідності перезапуску сервісів;
– надійність: час безвідмовної роботи має становити не менше 99.9% у середньому за місяць;
– зручність технічного обслуговування: код має бути структурованим та документованим, зі стандартами стилю (ESLint, Prettier);
– стандартизація API: RESTful API повинні відповідати стандартам OpenAPI для полегшення інтеграцій.

Врахування обох типів вимог дозволяє створити повноцінну та збалансовану систему.

1.3 Огляд фреймворка Laravel та його можливостей

Laravel у цьому проєкті використовується як фреймворк для реалізації серверної частини вебзастосунку соціальної мережі. Його оптимізована архітектура дозволяє ефективно обробляти велику кількість одночасних запитів, що особливо важливо для функціоналу обміну повідомленнями, стрічки публікацій та взаємодії між користувачами. Завдяки вбудованим механізмам кешування, маршрутизації, авторизації та захисту даних, Laravel забезпечує стабільну роботу застосунку навіть при високому навантаженні та сприяє безперервній і швидкій взаємодії з користувачем [5].

У межах розробки вебзастосунку соціальної мережі Laravel надає зручну вбудовану підтримку тестування за допомогою бібліотеки PHPUnit. Це дозволяє створювати модульні та інтеграційні тести для ключових компонентів, таких як авторизація, обробка публікацій, система коментарів та обмін повідомленнями. Написання тестів дає змогу перевірити коректність роботи функціоналу ще на етапі розробки та вчасно виявити можливі помилки. Крім того, це забезпечує стабільність проєкту при внесенні нових змін, оскільки тести допомагають контролювати, щоб новий код не порушував уже реалізовану логіку.

Laravel також постачається з вбудованою системою міграції, через яку легко оновлюються та змінюються схеми баз даних без складних SQL-запитів, що економить час [6].

Laravel має гнучку модульну архітектуру «модель-вид-контролер» (MVC), яка дозволяє чітко розділити логіку роботи з даними, інтерфейс користувача та керування взаємодією між ними. У цій архітектурній схемі кожен компонент виконує свою роль: модель (Model) відповідає за взаємодію з базою даних, контролер (Controller) – за обробку запитів і логіку застосунку, а вид (View) – за відображення інформації користувачу. У розробленому вебзастосунку соціальної мережі модель обробляє такі сутності, як користувачі, публікації, коментарі та повідомлення; контролери координують обробку запитів REST API; а роль інтерфейсу виконує клієнтська частина, реалізована засобами React. Такий підхід до структурування коду сприяє зрозумілості архітектури, полегшує масштабування, підтримку та тестування системи.

У межах проєкту Laravel обрано як основу для реалізації серверної частини вебсайту соціальної мережі, оскільки він забезпечує ефективну роботу з базою даних PostgreSQL, має потужну систему маршрутизації, підтримку REST-архітектури та дозволяє легко масштабувати проєкт. Завдяки розвиненій екосистемі, Laravel підходить для створення соціального функціоналу – управління користувачами, обробки запитів, взаємодії з контентом і забезпечення безпеки даних [5].

Окрім регулярного оновлення коду, він використовує захищені кешовані паролі, які ніколи не будуть збережені у базі даних у вигляді тексту, вбудовані функції безпеки, такі як захист від підробки міжсайтових запитів, запобігання SQL-ін’єкціям та захист від міжсайтових скриптів.

1.4 Огляд фреймворку React та його можливостей

Для розробки клієнтської частини вебсайту соціальної мережі було обрано бібліотеку React, яка зарекомендувала себе як одна з найпотужніших та найгнучкіших технологій для побудови сучасних інтерфейсів. Основною перевагою React є його компонентно-орієнтований підхід, що дозволяє розбивати інтерфейс на незалежні, повторно використовувані блоки. Це забезпечує кращу структуризацію коду, пришвидшує розробку та спрощує його підтримку й масштабування у майбутньому [6].

Окрім того, React використовує віртуальний DOM, який значно підвищує продуктивність застосунку завдяки мінімізації операцій з реальним DOM – оновлюються лише ті частини інтерфейсу, які справді зазнали змін. Такий підхід, у поєднанні з декларативним стилем програмування, дозволяє розробникам зосередитися на логіці відображення даних, а не на низькорівневій роботі з елементами сторінки.

У процесі реалізації застосовуються як функціональні, так і станні компоненти (stateful components). Функціональні компоненти використовуються переважно для відображення статичних або простої логіки частин інтерфейсу – таких як заголовки, кнопки, картки публікацій, списки коментарів. Натомість станні компоненти використовуються для реалізації складнішої взаємодії – авторизація користувачів, динамічне завантаження постів зі сторінки на сторінку, обробка форм введення, перевірка введених даних та інтеграція з сервером через API-запити [6].

Такий розподіл дозволяє не лише забезпечити високу якість та модульність коду, а й полегшує тестування окремих частин інтерфейсу та дає

змогу швидко впроваджувати новий функціонал без ризику порушити існуючу логіку.

React активно використовується у багатьох провідних світових соціальних платформах, зокрема Facebook, Instagram, WhatsApp, Reddit, що підтверджує його ефективність для реалізації високонавантажених, масштабованих застосунків із великою кількістю користувачів та частими оновленнями контенту в реальному часі. Саме тому його використання у межах проєкту є цілком обґрунтованим технічним вибором [7].

1.5 Порівняння вебзастосунку з іншими аналогами

У процесі розробки вебзастосунку для соціальної мережі з використанням фреймворків React, Laravel та PostgreSQL важливо оцінити його конкурентоспроможність на тлі сучасних альтернативних платформ, таких як Threads, Session тощо [7].

Threads, створений компанією Meta як альтернатива Twitter, орієнтований на короткі текстові пости, інтеграцію з Instagram та спрощену систему комунікації. Його перевагами є тісна інтеграція з існуючими акаунтами Instagram, зручність створення публікацій і простий, лаконічний інтерфейс. Проте Threads досі обмежений у функціональності: відсутня повноцінна підтримка десктоп-версії, слабка модерація контенту, обмеження щодо доступу для користувачів з певних країн, а також загальна новизна платформи, що не дозволяє їй ще конкурувати з усталеними сервісами на рівних.

Session – це децентралізована та орієнтована на безпеку альтернатива традиційним месенджерам і соціальним мережам. Основною її перевагою є високий рівень конфіденційності: використання технологій анонімного обміну інформацією (onion routing), відсутність необхідності у введенні персональних даних для реєстрації, наскрізне шифрування. Однак, така орієнтація на приватність також призводить до певних обмежень – відсутність

розширених соціальних функцій, обмеження у взаємодії між користувачами, і, як наслідок, обмежене коло цільової аудиторії [8].

У порівнянні з цими платформами, вебзастосунок соціальної мережі, реалізований на основі React, Laravel та PostgreSQL, пропонує більш гнучкий, інтуїтивно зрозумілий і сучасний підхід до взаємодії користувачів. Інтерфейс розроблено з використанням React, що забезпечує високу швидкість роботи, динамічне оновлення даних без повного перезавантаження сторінки, а також просту адаптацію користувачів до нових функцій. Завдяки модульній структурі, інтерфейс легко доповнюється новими компонентами, що забезпечує масштабованість системи. На відміну від Threads чи Session, у застосунку відсутня реклама, що значно покращує загальний користувацький досвід [8].

Щодо функціональності, платформа охоплює всі базові можливості соціальної мережі, такі як створення постів, коментарі, вподобайки, обмін повідомленнями, додавання друзів, створення груп і отримання сповіщень. Laravel як серверна технологія дозволяє забезпечити ефективну обробку запитів, стабільність і можливість роботи в реальному часі, що вигідно відрізняє наш вебзастосунок від конкурентів.

У плані продуктивності даний вебзастосунок демонструє високу ефективність завдяки поєднанню React на фронтенді, Laravel на бекенді та PostgreSQL як системи керування базами даних. Це дозволяє досягти швидкого завантаження сторінок, оперативного доступу до інформації та безперебійної роботи з великими обсягами даних.

Інтеграція з іншими сервісами є ще однією сильною стороною даного проєкту. Завдяки RESTful API та можливостям Laravel, застосунок можна легко поєднувати з платіжними системами, чат-ботами або іншими соціальними мережами для кросплатформного поширення контенту. На відміну від цього, Threads та Session, хоча і підтримують інтеграцію, часто обмежують можливості зовнішніх розробників, створюючи бар’єри у вигляді складних правил доступу до API [8].

Таким чином, вебзастосунок має низку очевидних переваг. Насамперед це повна відсутність реклами, що дозволяє сконцентруватися лише на спілкуванні та контенті. По-друге, більша свобода у персоналізації профілів, створенні груп і спільнот розширює можливості взаємодії. По-третє, технологічна основа проєкту забезпечує стабільну, швидку й масштабовану роботу. Нарешті, відкритість до інтеграцій робить систему гнучкою до розширення функціоналу, що вигідно вирізняє застосунок серед конкурентів і робить його перспективною платформою для розвитку соціальних комунікацій.

1.6 Висновки до розділу 1

У першому розділі проведено аналіз сучасних вебсайтів соціальних мереж та технологій, що використовуються для їх розробки. Обґрунтовано вибір React як основної технології для реалізації клієнтської частини, враховуючи його компонентно-орієнтовану архітектуру, високу продуктивність та гнучкість у створенні динамічних інтерфейсів. Для серверної частини обрано Laravel – сучасний PHP-фреймворк, що забезпечує зручну роботу з маршрутизацією, автентифікацією, базами даних та REST API. Як систему управління базами даних обрано PostgreSQL – надійне та масштабоване рішення, яке ефективно працює з реляційними даними.

Таким чином, у результаті проведеного аналізу було сформовано технічне підґрунтя для реалізації вебсайту соціальної мережі з урахуванням сучасних вимог до функціональності, продуктивності, безпеки та масштабованості.

2 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ СОЦІАЛЬНИХ МЕРЕЖ

2.1 Проєктування інтерфейсу користувача

Проєктування інтерфейсу користувача для вебсайту соціальної мережі є важливим етапом, що впливає на зручність, привабливість та ефективність взаємодії користувачів із системою. Нижче наведено кілька кроків і рекомендацій для створення зрозумілого та сучасного інтерфейсу.

Основними елементами інтерфейсу нашої соціальної мережі є:

– головна стрічка новин;
– сторінка профілю користувача;
– сторінка повідомлень (чат);
– сторінка списку друзів;
– налаштування облікового запису.

На головній сторінці (стрічці новин) користувач бачить актуальні публікації від друзів, можливість взаємодії з ними, а також основні функції платформи. Приклад структури головної сторінки:

– блок створення нового допису (текст, зображення);
– стрічка новин з публікаціями друзів;
– кнопки взаємодії (вподобати, прокоментувати, поділитися, тощо);
– меню навігації (профіль, друзі, повідомлення, сповіщення, тощо);
– панель пошуку користувачів або публікацій.

Такий підхід до проєктування інтерфейсу дозволяє забезпечити інтуїтивне використання, швидкий доступ до основних функцій та приємний візуальний досвід для користувачів.

При проєктуванні програмного забезпечення важливу роль відіграє створення прототипів (макетів) ключових компонентів майбутнього вебзастосунку. Прототипування дозволяє на ранньому етапі візуалізувати функціональність системи, перевірити логіку розташування елементів, а також забезпечити зручність і зрозумілість інтерфейсу для користувачів.

На рисунку 2.1 наведено макет головної сторінки соціальної мережі після входу користувача. На цій сторінці відображається персоналізована стрічка новин з останніми оновленнями друзів і спільнот.

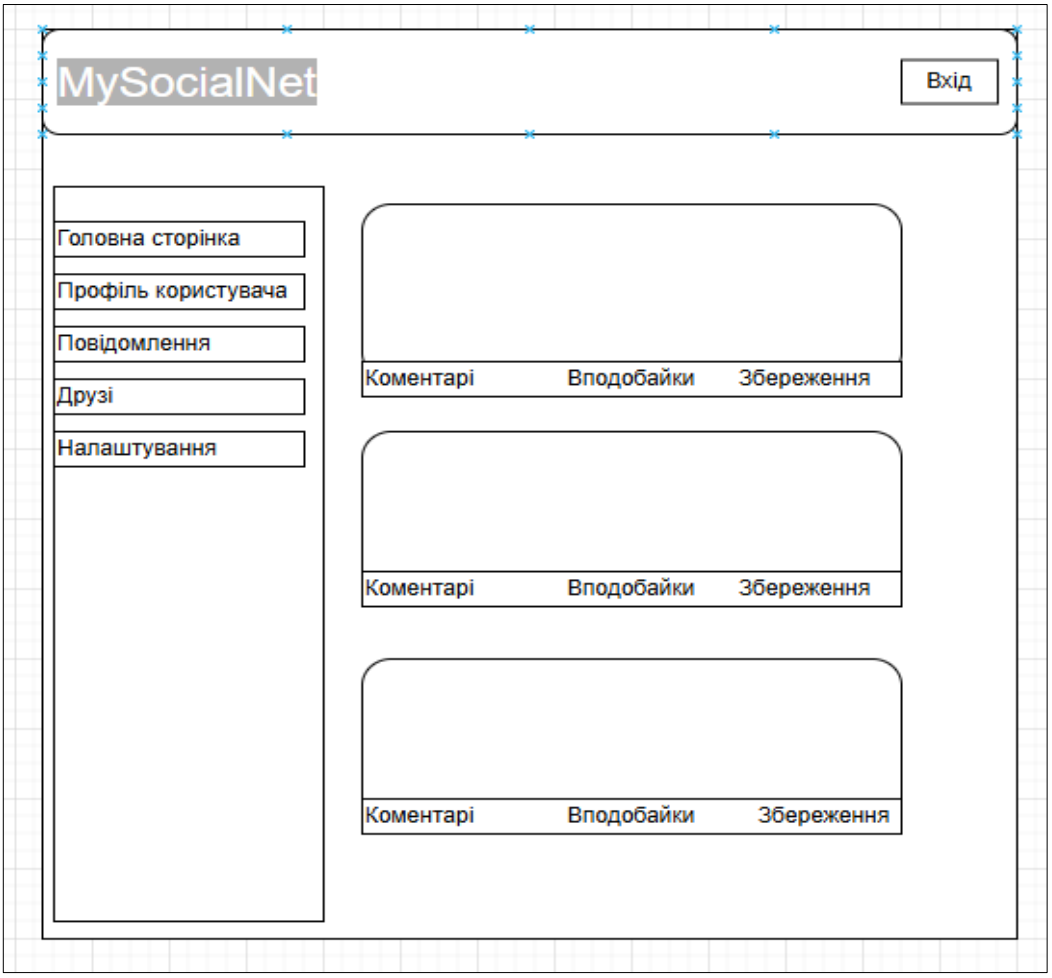


Рисунок 2.1 – Макет головної сторінки

Відповідно до наведеного макету соціальної мережі, інтерфейс користувача організований у вигляді трьох основних візуальних зон, які забезпечують інтуїтивну навігацію та зручність взаємодії з системою.

Верхній блок (хедер) – це заголовок сторінки з назвою соціальної мережі MySocialNet, оформлений у вигляді горизонтальної панелі і білим текстом. Цей елемент постійно присутній у верхній частині інтерфейсу й виконує роль фірмового логотипу системи та головного навігаційного маркера. Така реалізація асоціюється з класичним інтерфейсом популярних соціальних платформ і додає впізнаваності.

Ліва панель (sidebar) – це вертикальна навігаційна панель, розташована зліва, яка містить набір з шести елементів меню:

– головна сторінка;
– профіль користувача;
– повідомлення;
– друзі;
– налаштування.

Цей блок розміщено у білому прямокутнику з тінню, що виділяє його на фоні сторінки. Меню представлено вертикальним списком з відступами та жирним шрифтом для зручної навігації. Такий підхід спрощує орієнтацію користувача між модулями.

Центральна частина (feed) займає основну ширину екрана і містить стрічку у вигляді карток постів. Кожна картка має стандартну структуру.

На рисунку 2.2 наведено діаграму варіантів використання (діаграму прецедентів) для вебсайту соціальної мережі.

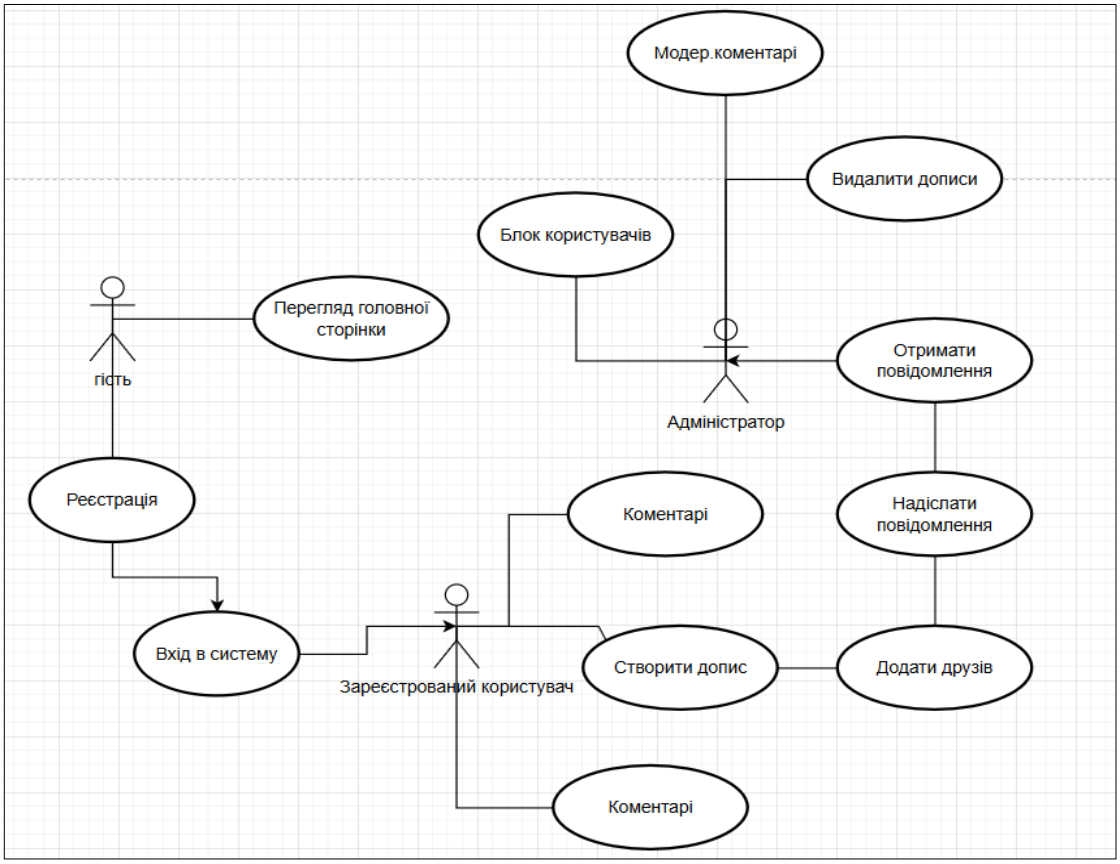


Рисунок 2.2 – Діаграма прецедентів соціальних мереж

Діаграма демонструє основні функціональні можливості платформи та відображає взаємодію різних типів користувачів із системою. За допомогою цієї діаграми можна побачити, які дії доступні кожному типу користувача.

Діаграма прецедентів показує взаємодію трьох типів користувачів із системою соціальної мережі: неавторизованого користувача, авторизованого користувача та адміністратора.

Неавторизований користувач має доступ лише до головної сторінки з описом можливостей сервісу та формами реєстрації або авторизації. Реєстрація передбачає введення пошти й пароля з підтвердженням, авторизація – вхід за логіном або поштою.

Після входу користувач отримує доступ до стрічки новин, створення дописів, коментарів, приватних повідомлень, керування профілем і друзями. Адміністратор, у свою чергу, може модерувати контент: блокувати користувачів, видаляти дописи та контролювати дотримання правил.

2.2 Структура бази даних вебзастосунку

У розробці вебсайту соціальної мережі використовується система керування базами даних PostgreSQL, яка забезпечує стабільну роботу з великим обсягом взаємозалежних даних. База даних побудована відповідно до принципів реляційної моделі, що дозволяє гнучко оперувати сутностями, встановлювати зв’язки між ними та ефективно виконувати запити.

Основна мета реалізації бази даних – зберігати дані про користувачів, їхню активність (пости, коментарі, повідомлення, вподобання), соціальні зв’язки (дружба), а також сповіщення та медіафайли. Така модель дозволяє реалізувати ключові функції соціальної мережі: створення дописів, вподобайки, коментарі, обмін повідомленнями, додавання в друзі, перегляд профілів та отримання сповіщень [9].

Для побудови ефективної структури бази даних необхідно визначити ключові сутності та ролі користувачів.

Користувачі – основні учасники системи. Мають поля: ім’я, email, пароль, фото, статус тощо.

Ролі:

– звичайний користувач – має доступ до публікацій, створення коментарів, додавання друзів;

– адміністратор – може модерувати контент, блокувати користувачів, переглядати статистику.

Основні сутності бази даних:

– users – зберігає облікові записи користувачів;

– posts – публікації, створені користувачами;

– comments – коментарі до публікацій;

– likes – реакції на публікації або коментарі;

– friendships – таблиця зв’язків між користувачами;

– roles – ролі користувачів для системи авторизації;

– notifications – повідомлення користувачам;

– media_files – медіа ресурси(відео,фото, тощо).

На рисунку 2.3 представлено ER-діаграму бази даних вебзастосунку, яка демонструє взаємозв’язки між основними сутностями та їх атрибутами.

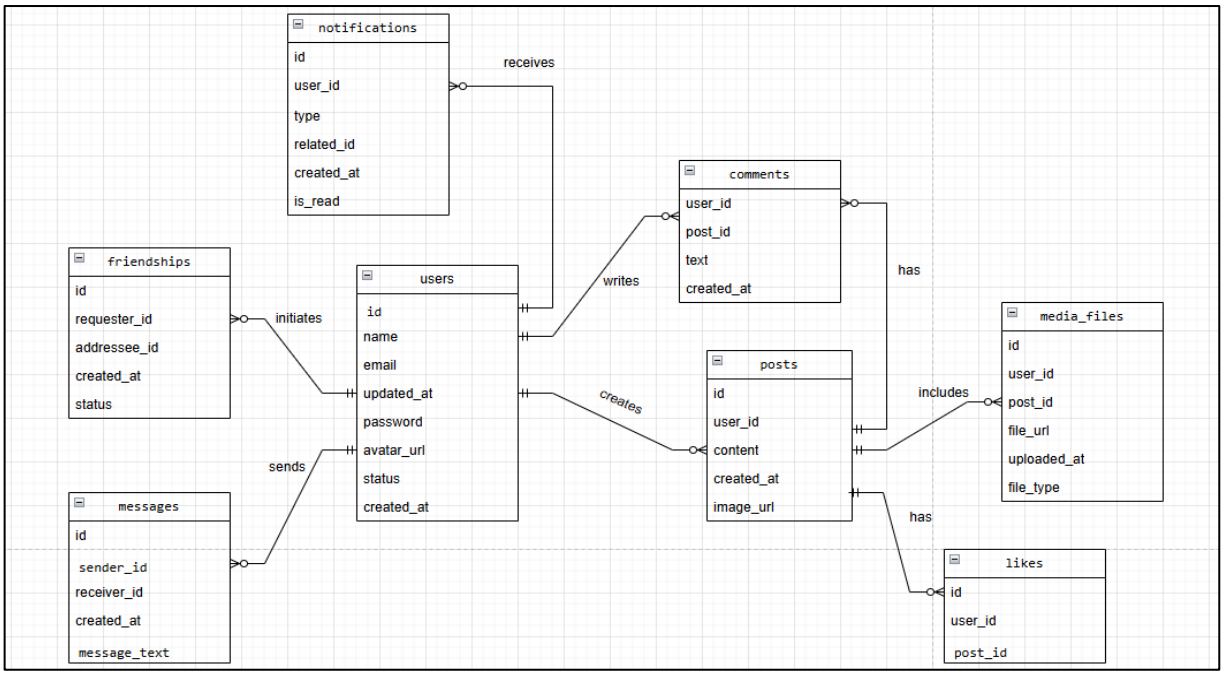


Рисунок 2.3 – ER-діаграм бази даних

Діаграма дає змогу візуально оцінити структуру даних і показує, як реалізовано залежності між таблицями, включаючи зовнішні ключі та обмеження цілісності.

Опишемо таблиці сутностей і зв’язків (ERD) на основі наданої діаграми бази даних.

Таблиця users (користувачі):

– id – унікальний ідентифікатор користувача, автоматично генерується;
– name – ім’я або нікнейм користувача;
– email – електронна пошта користувача (унікальна);
– password – зашифрований пароль користувача;
– avatar_url – посилання на аватар користувача (необов’язково);
– status – статус користувача, наприклад «онлайн», «офлайн», тощо;
– created_at – дата та час створення облікового запису;
– updated_at – дата та час останнього оновлення інформації користувача.

Таблиця posts (публікації):

– id – унікальний ідентифікатор публікації;
– user_id – зовнішній ключ, що посилається на користувача, який створив пост;
– content – текст публікації;
– image_url – посилання на зображення, прикріплене до посту (необов’язково);
– created_at – дата та час створення публікації.

Таблиця comments (коментарі):

– id – унікальний ідентифікатор коментаря;
– user_id – зовнішній ключ, що посилається на користувача, який написав коментар;
– post_id – зовнішній ключ, що посилається на пост, до якого належить коментар;

– text – текст коментаря;
– created_at – дата та час створення коментаря.

Таблиця likes (вподобання):

– id – унікальний ідентифікатор вподобайки;
– user_id – зовнішній ключ, що посилається на користувача, який поставив вподобайку;
– post_id – зовнішній ключ, що посилається на пост, до якого поставлено вподобайку.

Таблиця friendships (дружба):

– id – унікальний ідентифікатор зв’язку;
– requester_id – зовнішній ключ, що посилається на користувача, який відправив запит на дружбу;
– addressee_id – зовнішній ключ, що посилається на користувача, якому було відправлено запит;
– status – статус дружби (наприклад, “pending”, “accepted”, “rejected”);
– created_at – дата та час створення запису про дружбу.

Таблиця messages (повідомлення):

– id – унікальний ідентифікатор повідомлення;
– sender_id – зовнішній ключ, що посилається на користувача, який надіслав повідомлення;
– receiver_id – зовнішній ключ, що посилається на користувача, який отримав повідомлення;
– message_text – текст повідомлення;
– is_read – позначка, чи було прочитано повідомлення;
– created_at – дата та час відправлення повідомлення.

Таблиця notifications (сповіщення):

– id – унікальний ідентифікатор сповіщення;
– user_id – зовнішній ключ, що посилається на користувача, якому адресовано сповіщення;
– type – тип сповіщення (наприклад, “like”, “comment”, “friend_request”, “message”);

– related_id – ID об’єкта, пов’язаного зі сповіщенням (наприклад, ID поста, повідомлення тощо);
– is_read – позначка, чи було прочитано сповіщення;
– created_at – дата та час створення сповіщення.

Таблиця media_files (медіафайли):

– id – унікальний ідентифікатор медіафайлу;
– user_id – зовнішній ключ, що посилається на користувача, який завантажив файл;
– post_id – зовнішній ключ, що посилається на пост, до якого прикріплено файл (може бути порожнім);
– file_url – посилання на медіафайл;
– file_type – тип файлу (наприклад, “image”, “video”);
– uploaded_at – дата та час завантаження файлу.

У базі даних проєкту реалізовано зв’язки відповідно до логіки взаємодії між сутностями. Зокрема, між таблицями “users” і “posts”, “comments”, “likes”, “notifications”, “media_files” встановлено зв’язок один до багатьох – один користувач може мати багато відповідних записів, але кожен із них належить лише одному користувачу.

Також таблиці “posts” пов’язані з “comments” і “likes” типом зв’язку один до багатьох, адже пост може мати багато коментарів і вподобань. У таблиці “media_files” зв’язок з постом необов’язковий (nullable).

Повідомлення між користувачами реалізовано через два зв’язки один до багатьох – окремо для відправника і отримувача. Це дозволяє кожному користувачу надсилати та отримувати багато повідомлень. Дружні зв’язки моделюються за допомогою проміжної таблиці “friendships” з двома зовнішніми ключами на таблицю “users”, що реалізує зв’язок багато до багатьох – кожен користувач може бути як ініціатором, так і отримувачем запитів.

Таким чином, у базі даних застосовані основні типи зв’язків – один до багатьох і багато до багатьох, що забезпечує коректну структуру для

збереження інформації про користувачів, пости, коментарі, лайки, повідомлення, медіафайли, сповіщення та дружні взаємини між користувачами [10].

2.3 Проєктування архітектури вебзастосунку

Проєктування архітектури вебзастосунку є критично важливим етапом, що забезпечує належну структуру, масштабованість та підтримуваність системи. У випадку вебзастосунку соціальної мережі з використанням React, Laravel і PostgreSQL, архітектура включає наступні основні компоненти:

- клієнтська частина (Frontend);
- серверна частина (Backend);
- база даних;
- система аутентифікації та авторизації;
- реалізація реального часу (WebSocket) для повідомлень та сповіщень.

Клієнтська частина вебзастосунку соціальної мережі розробляється з використанням React – сучасної бібліотеки JavaScript для створення динамічних користувацьких інтерфейсів. Основні задачі фронтенду включають:

- відображення користувацького інтерфейсу (UI);
- обробку подій і введених даних користувача;
- маршрутизацію сторінок (за допомогою React Router);
- управління глобальним і локальним станом (через React Context або Redux);
- взаємодію з сервером через HTTP-запити (з використанням Axios або Fetch API);
- зберігання токенів доступу для авторизації (наприклад, у localStorage).

Наведемо візуальну структуру клієнтської частини на рисунку 2.4. Основні файли та їх функції:

– App.jsx або App.js: головний компонент React-застосунку, в якому відбувається відображення усього користувацького інтерфейсу;
– index.js або main.jsx: точка входу в застосунок, що ввідображує <App /> у DOM;
– router/: конфігурація React Router;
– services/: логіка роботи з API (наприклад, запити через Axios);
– components/: набір UI-компонентів;
– pages/: окремі сторінки, які відповідають певним маршрутам.

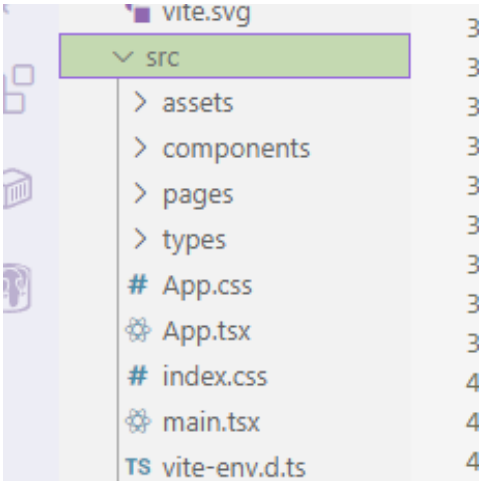


Рисунок 2.4 – Структура користувацької частини

Серверна частина вебзастосунку для відстеження даних побудована на основі Laravel. Основні компоненти: контролери, маршрути, моделі, міграції, сервіси та середовища.

Контролери у Laravel відповідають за обробку запитів та реалізацію логіки додатка, дозволяючи чітко розподіляти функціональність між різними обробниками запитів.

Маршрути вказують, які HTTP-запити мають направлятися до конкретних контролерів для обробки, і визначаються у файлах routes/web.php для вебзастосунків та routes/api.php для API-застосунків [2].

Візуальну структуру серверної частини можна побачити на рисунку 2.5.

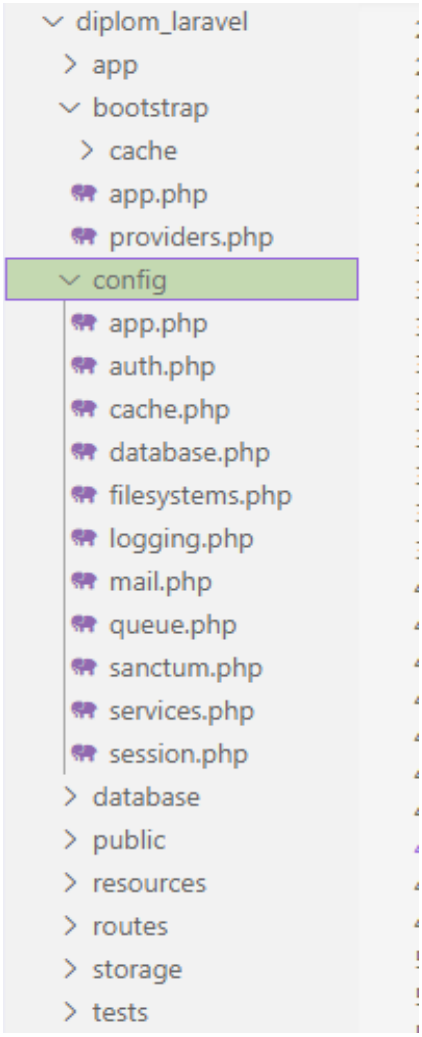


Рисунок 2.5 – Структура серверної частини

Для зберігання даних у вебзастосунку, створеному на основі Laravel, буде використовуватись база даних PostgreSQL. Візуалізуємо налаштування бази даних у Laravel на рисунку 2.6.

```
CACHE_STORE=database
#·CACHE_PREFIX=

MEMCACHED_HOST=127.0.0.1

REDIS_CLIENT=phpredis
REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379

MAIL_MAILER=log
MAIL_SCHEME=null
MAIL_HOST=127.0.0.1
```

Рисунок 2.6 – Налаштування бази даних

Наведемо основні файли та методи бекенду, які відповідають за обробку запитів, взаємодію з базою даних та зовнішніми сервісами:

- UserController.php – контролер для обробки запитів, пов’язаних з користувачами (реєстрація, авторизація, профіль);
- PostController.php – контролер для створення, редагування та відображення публікацій;
- CommentController.php – контролер для обробки коментарів до публікацій;
- LikeController.php – контролер для обробки вподобань (лайків);
- FriendshipController.php – контролер для керування заявками в друзі та списками друзів;
- MessageController.php – контролер для надсилання та отримання особистих повідомлень між користувачами;
- NotificationController.php – контролер для створення та відправлення сповіщень користувачам;
- User.php – модель, що відображає структуру таблиці користувачів;
- Post.php – модель, що описує структуру таблиці публікацій;
- Comment.php – модель, що відповідає за таблицю коментарів;
- Like.php – модель для роботи з таблицею вподобань;
- Friendship.php – модель, що відображає зв’язки між користувачами (дружбу);
- Message.php – модель для зберігання повідомлень;
- Notification.php – модель для роботи зі сповіщеннями;
- web.php – файл маршрутів для вебзапитів (наприклад, перегляд профілю через браузер);
- api.php – файл маршрутів для API-запитів з клієнтської частини (React);
- .env – файл конфігурації середовища, який містить параметри підключення до PostgreSQL, а також інші налаштування застосунку.

Також на рисунку 2.7 представлено діаграму компонентів, яка описує архітектурну структуру вебсайту соціальної мережі, реалізованого за допомогою засобів React, Laravel та PostgreSQL.

Ця діаграма ілюструє взаємозв'язки між основними функціональними модулями клієнтської та серверної частин, а також зовнішніми сервісами, які забезпечують повноцінне функціонування системи. Клієнтська частина (React) взаємодіє з сервером через APIClient, надсилаючи HTTP-запити до відповідних контролерів Laravel.

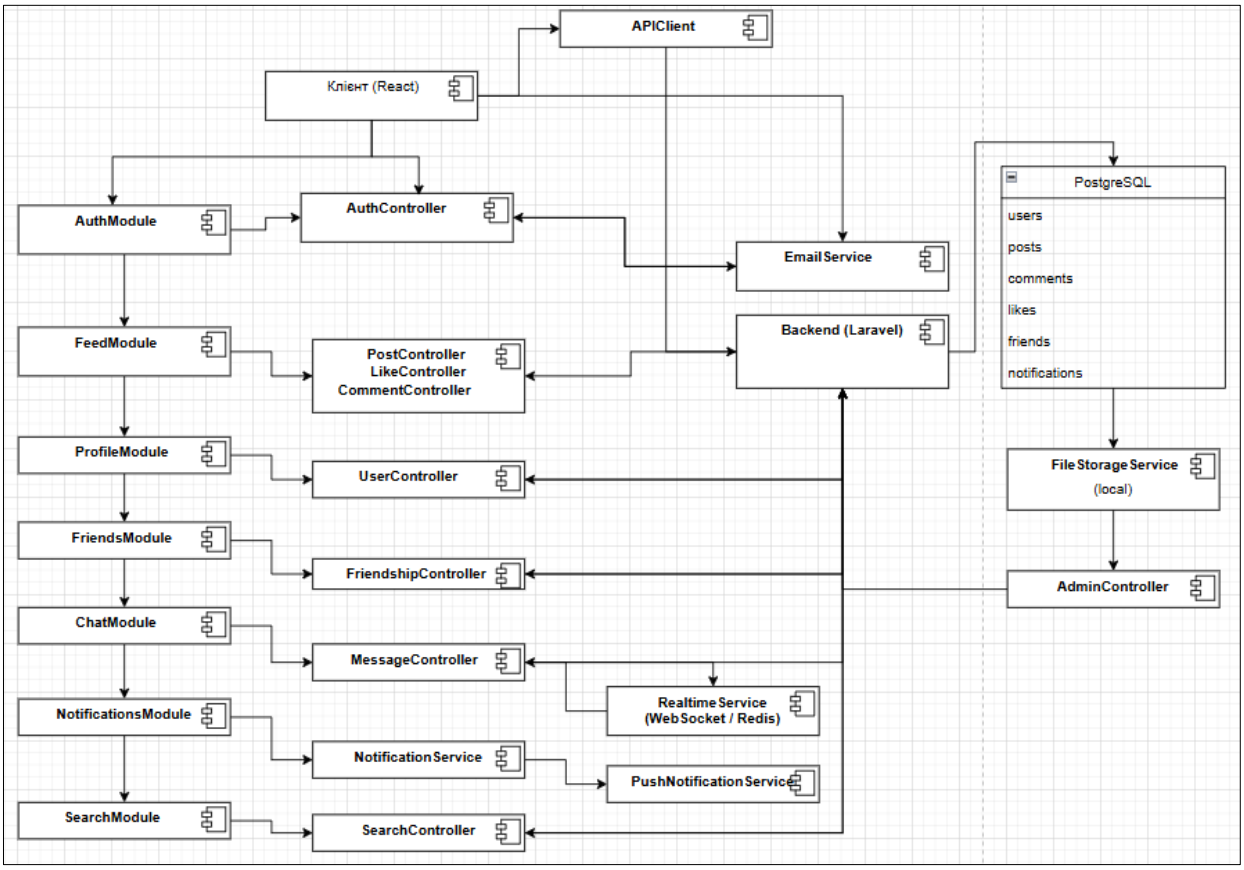


Рисунок 2.7 – Діаграма компонентів

Основна функціональність інтерфейсу розділена на окремі модулі:

– AuthModule – модуль аутентифікації;
– FeedModule – модуль стрічки новин (дописи, лайки, коментарі);
– ProfileModule – модуль профілю користувача;
– FriendsModule – модуль управління дружніми зв'язками;
– ChatModule – модуль особистих повідомлень;

– NotificationsModule – модуль сповіщень;
– SearchModule – модуль пошуку користувачів або контенту.

Серверна частина (Laravel) реалізує контролери для обробки логіки відповідних модулів:

– AuthController – обробляє вхід, реєстрацію та валідацію користувачів;
– PostController, LikeController, CommentController – обробка дописів, лайків і коментарів;
– UserController – керування профілями;
– FriendshipController – керування дружніми зв’язками;
– MessageController – обробка повідомлень між користувачами;
– NotificationService – логіка формування сповіщень;
– SearchController – реалізація пошуку;
– AdminController – керування адміністративною частиною системи.

Для забезпечення інтерактивності та зручності у вебсайті соціальної мережі доцільно використовувати зовнішні сервіси, такі як сервіси для надсилання електронних листів (наприклад, SendGrid), обробки зображень (Cloudinary) або пуш-сповіщень (Firebase Cloud Messaging).

Такі API дозволяють значно розширити функціональність проєкту без необхідності розробляти ці сервіси з нуля. Інтеграція сторонніх сервісів дає змогу автоматично надсилати сповіщення користувачам про нові повідомлення, заявки в друзі, вподобання або коментарі, а також забезпечити зручне зберігання та обробку медіаконтенту. Це дозволяє зробити вебсайт більш динамічним, зручним і привабливим для користувачів [9].

2.4 Проєктування серверної частини

Налаштування серверної частини вебзастосунку є важливим етапом, що впливає на стабільність, безпеку та продуктивність системи. Основні

конфігураційні файли зазвичай розміщуються в каталозі config, а також у файлі .env, який зберігає змінні середовища [6].

Ключові файли налаштування:

– .env – головний файл конфігурації, в якому зберігаються конфіденційні дані та змінні середовища: параметри підключення до бази даних, API-ключі, системні налаштування тощо;
– config/app.php – файл, що містить базові налаштування застосунку: назва, URL, часовий пояс, мови, підключені сервіси;
– config/database.php – відповідає за налаштування з'єднання з базою даних, включаючи тип СУБД, хост, ім'я користувача, пароль та інші параметри;
– config/cache.php – визначає механізми кешування, тип кешу (наприклад, file, redis), його драйвери та поведінку;
– config/session.php – містить налаштування для роботи з сесіями користувачів: драйвери зберігання, тривалість, безпека, шифрування;
– config/logging.php – параметри ведення журналів: рівні логування, типи логів (файл, база даних, Slack тощо), структура та місця збереження.

Ці конфігураційні файли забезпечують гнучкість та контроль над поведінкою Laravel-додатку в різних середовищах (розробка, тестування).

На рисунку 2.8 наведено приклад налаштування файлу ".env".

```
MEMCACHED_HOST=127.0.0.1

REDIS_CLIENT=phpredis
REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379

MAIL_MAILER=log
MAIL_SCHEME=null
MAIL_HOST=127.0.0.1
MAIL_PORT=2525
MAIL_USERNAME=null
MAIL_PASSWORD=null
```

Рисунок 2.8 – Налаштування файлу ".env"

Налаштування серверної частини є ключовим етапом, що забезпечує стабільну, безпечну та ефективну роботу вебзастосунку. Воно визначає базові параметри, такі як підключення до бази даних, кешування, робота з сесіями, логування тощо, а також дозволяє адаптувати ці параметри відповідно до конкретних умов експлуатації.

Під час налаштування серверної частини необхідно враховувати такі важливі аспекти:

- безпека – необхідно передбачити шифрування конфіденційних даних, обмеження прав доступу, захист від типових вразливостей, таких як SQL-ін’єкції чи XSS-атаки;
- доступність – система має бути здатною витримувати високі навантаження та стабільно обробляти запити у реальному часі;
- масштабованість – важливо забезпечити можливість розширення серверної частини, наприклад, за допомогою балансування навантаження чи горизонтального масштабування;
- моніторинг та логування – впровадження системи моніторингу та журналювання дозволяє контролювати стан системи, швидко виявляти та реагувати на помилки або збої;
- резервне копіювання – регулярне створення резервних копій бази даних та важливих файлів є критично важливим для забезпечення надійного відновлення після збоїв;
- оптимізація продуктивності – необхідно правильно налаштувати кешування, оптимізувати запити до бази даних та зменшити надлишкове навантаження на сервер.

Правильне налаштування серверної частини допоможе забезпечити стабільну та безпечну роботу вебзастосунку, знижуючи ймовірність виникнення проблем та забезпечуючи задоволення користувачів з його використання [11].

На рисунку 2.9 наведено приклад діаграми розгортання, яка демонструє взаємодію між основними компонентами серверної архітектури вебзастосунку соціальної мережі. Зокрема, діаграма ілюструє, як клієнтські запити,

сформовані у браузері користувача, передаються до вебсервера, де обробляються за допомогою PHP-інтерпретатора та фреймворку Laravel.

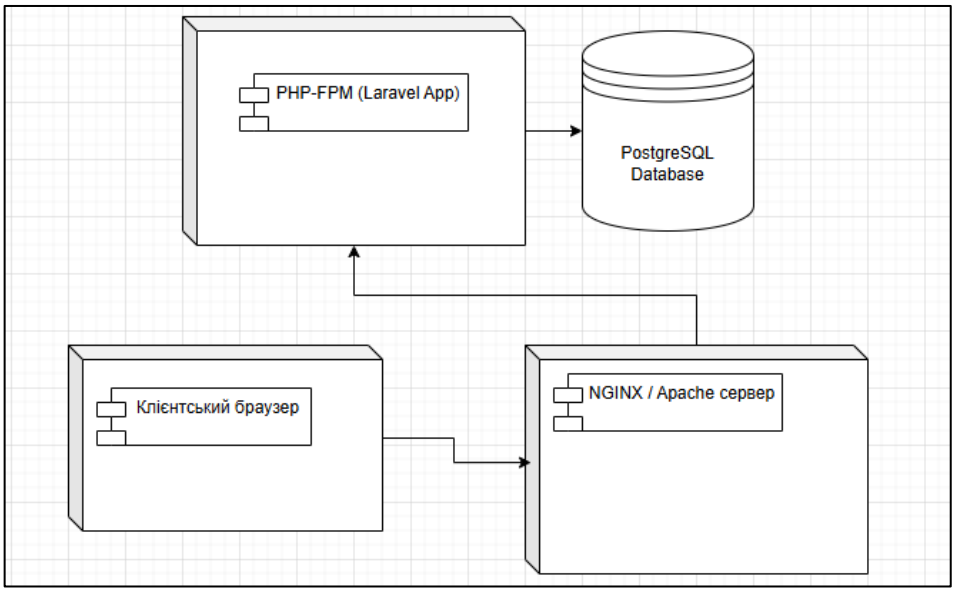


Рисунок 2.9 – Діаграма розгортання

Клієнтський браузер є точкою входу користувача до системи. За допомогою інтерфейсу, реалізованого на React, користувач формує запити, які надсилаються на сервер для обробки. Ці запити охоплюють такі дії, як реєстрація, авторизація, публікація постів або перегляд стрічки новин.

Вебсервер (NGINX або Apache) приймає запити з клієнтської сторони та перенаправляє їх на PHP-інтерпретатор. Його завдання – забезпечити стабільну маршрутизацію, обробку статичних ресурсів і передачу запитів до серверної частини застосунку.

PHP-FPM разом із Laravel-застосунком відповідає за обробку бізнес-логіки. Контролери приймають запити, сервіси виконують необхідні обчислення або обробку, а моделі забезпечують доступ до бази даних. На цьому етапі реалізується логіка створення постів, керування користувачами, взаємодія з повідомленнями тощо [12].

PostgreSQL є основним сховищем усіх даних. У ньому зберігається інформація про акаунти користувачів, повідомлення, публікації, коментарі та інші сутності, необхідні для роботи соціальної мережі.

Laravel WebSockets забезпечує двосторонню комунікацію в режимі реального часу. Цей компонент використовується для реалізації чатів, сповіщень, онлайн-статусів користувачів та інших функцій, які вимагають миттєвого оновлення інформації без перезавантаження сторінки.

На рисунку 2.10 наведено діаграму класів серверної частини, яка ілюструє основні сутності вебзастосунку соціальної мережі та взаємозв'язки між ними. Зокрема, зображено класи User, Post, Comment, Like, Message, FriendRequest та PasswordResetToken.

Кожна сутність має відповідні атрибути (наприклад, id, text, created_at) та зв'язки з іншими класами через асоціації, позначені стрілками навігації з відповідною кардинальністю.

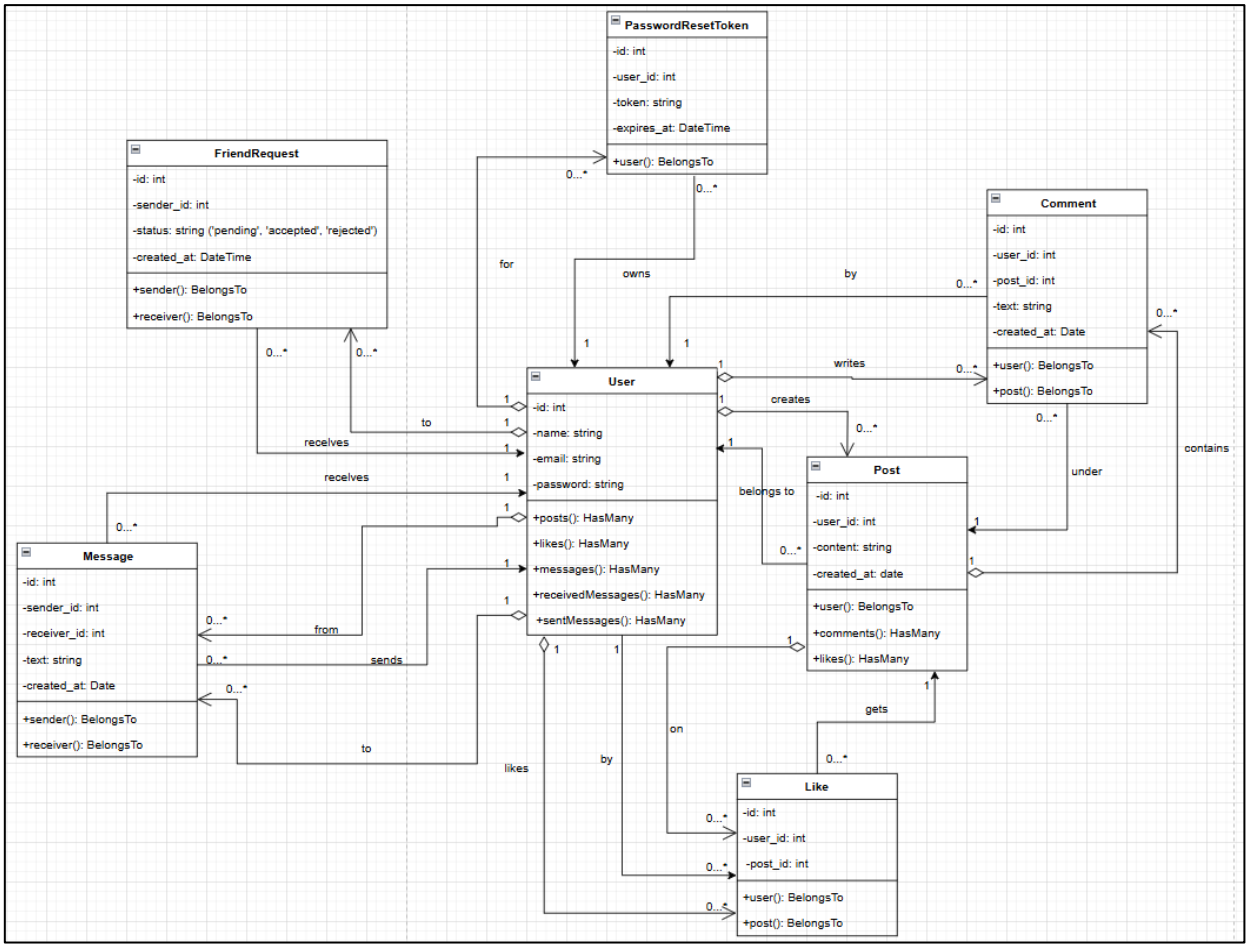


Рисунок 2.10 – Діаграма класів серверної частини

Клас User є центральним у структурі системи, оскільки з ним пов'язані всі інші сутності: користувачі можуть створювати пости, залишати коментарі,

ставити вподобання (лайки), надсилати повідомлення, а також отримувати або надсилати запити у друзі.

Асоціації типу «один до багатьох» позначаються стрілкою від батьківського класу до дочірнього з кардинальністю «1» біля батьківського класу та «*» біля дочірнього.

Ця діаграма дозволяє унаочнити логічну структуру бази даних серверної частини, що реалізована засобами Laravel, та забезпечує чітке розуміння зв’язків між компонентами, які лежать в основі бізнес-логіки системи. Вона також служить основою для подальшого фізичного проєктування бази даних та реалізації ORM-зв’язків у коді [13].

2.5 Висновки до розділу 2

У другому розділі, присвяченому проєктуванню вебсайту соціальної мережі, визначено перелік функціональних і нефункціональних вимог до системи, що дало можливість чітко визначити її основні можливості та критерії ефективності. Проведено аналіз ключових сценаріїв взаємодії користувачів із системою за допомогою діаграми прецедентів, яка окреслює основні ролі та дії користувачів.

Створено логічну структуру вебзастосунку, а також представлено структуру бази даних, що детально описує взаємозв’язки між таблицями, такими як users, posts, friends, comments, messages тощо. Це дозволяє забезпечити ефективне зберігання, обробку та доступ до даних.

Загалом, проведене проєктування сформувало цілісну архітектурну основу для подальшої реалізації вебсайту соціальної мережі, яка відповідає сучасним вимогам до продуктивності, масштабованості та зручності користування.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Реалізація серверної частини

У рамках розробки вебсайту соціальної мережі було прийнято рішення реалізувати серверну логіку за допомогою фреймворку Laravel, що забезпечує ефективне управління бізнес-логікою, маршрутизацією та роботою з базою даних PostgreSQL без необхідності ручного адміністрування серверних процесів.

Клієнтська частина, створена на React, взаємодіє з бекендом через RESTful API, реалізоване на Laravel. Запити від користувачів надходять з React-додатку до API Laravel, який обробляє їх відповідними контролерами.

Схема взаємодії виглядає таким чином:

– користувач ініціює дію через React-інтерфейс;
– запит надсилається до Laravel API;
– API виконує бізнес-логіку, взаємодіє з базою даних PostgreSQL, створює, оновлює, видаляє або отримує дані;
– результат повертається назад до React-додатку і відображається користувачу.

Таке розділення забезпечує масштабованість системи, зручність розробки і підтримки, а також високу продуктивність вебсайту соціальної мережі.

Розглянемо декілька контролерів Laravel, які використовуються у застосунку для обробки запитів та реалізації бізнес-логіки. На рисунку 3.1 наведено приклад методу контролера для отримання даних з бази даних PostgreSQL.

Функція виконує роль частини серверної логіки у вебзастосунку соціальної мережі. Її основне призначення – обробляти HTTP-запити для роботи з постами, взаємодіяти з базою даних PostgreSQL та повертати необхідні дані клієнтській частині у форматі JSON.

Функція виконує такі кроки:

– приймає HTTP-запит від клієнтської частини, наприклад, для отримання, створення, оновлення або видалення постів;
– здійснює валідацію вхідних даних (для методів створення і оновлення), перевіряючи наявність обов’язкових полів і коректність їх форматів;
– виконує операції з базою даних PostgreSQL через ORM Eloquent: отримує записи з таблиці posts, створює нові записи, оновлює або видаляє **існуючі.**

```php
<?php

namespace App\Http\Controllers;

use App\Models\Post;
use Illuminate\Http\Request;

class PostsController extends Controller
{
    public function index()
    {
        // Отримуємо всі пости з бази даних
        $posts = Post::all();

        // Повертаємо дані у форматі JSON
        return response()->json($posts);
    }
    public function store(Request $request)
        $validated = $request->validate([
            'title' => 'required|string|max:255',
            'content' => 'required|string',
            'user_id' => 'required|integer|exists:users,id',
        ]);

        $post = Post::create($validated);

        return response()->json($post, 201);
    }
    public function show($id)
    {
        $post = Post::findOrFail($id);

        return response()->json($post);
    }
```

Рисунок 3.1 – Контролер для отримання даних з бази даних

Контролер PostsController забезпечує повноцінну роботу з постами у вебзастосунку соціальної мережі, реалізуючи основні операції створення, читання, оновлення та видалення даних. Завдяки використанню вбудованих

механізмів валідації та ORM Eloquent, контролер гарантує коректність введених даних і ефективну взаємодію з базою даних PostgreSQL.

Таким чином, він є ключовим компонентом серверної частини, який відповідає за реалізацію бізнес-логіки та забезпечення надійного обміну інформацією між клієнтським інтерфейсом і базою даних.

На рисунку 3.2 наведено приклад контролера Laravel CommentsController, який реалізує функції додавання, отримання, оновлення та видалення коментарів.

```php
<?php

namespace App\Http\Controllers;

use App\Models\Comment;
use Illuminate\Http\Request;

class CommentsController extends Controller
{
    public function index()
    {
        $comments = Comment::all();
        return response()->json($comments);
    }

    public function store(Request $request)
    {
        $validated = $request->validate([
            'post_id' => 'required|integer|exists:posts,id',
            'user_id' => 'required|integer|exists:users,id',
            'content' => 'required|string',
        ]);

        $comment = Comment::create($validated);
        return response()->json($comment, 201);
    }

    public function show($id)
    {
        $comment = Comment::findOrFail($id);
        return response()->json($comment);
    }

    public function update(Request $request, $id)
    {
        $comment = Comment::findOrFail($id);

        $validated = $request->validate([
            'content' => 'sometimes|required|string',
        ]);

        $comment->update($validated);
        return response()->json($comment);
    }
```

Рисунок 3.2 – Контролер для взаємодії з коментарями

Контролер CommentsController відповідає за керування коментарями у вебзастосунку соціальної мережі. Його функціонал включає отримання списку всіх коментарів, створення нового коментаря, отримання конкретного коментаря за ідентифікатором, оновлення вмісту коментаря та видалення коментаря з бази даних.

Метод index повертає всі коментарі у форматі JSON. Метод store приймає дані нового коментаря, виконує їх валідацію (перевіряє наявність посилань на пост і користувача, а також вміст коментаря), створює запис у базі і повертає створений коментар з кодом відповіді 201. Метод show отримує коментар за вказаним ID, а update дозволяє оновити текст коментаря після перевірки коректності даних. Метод destroy видаляє коментар за ID та повертає відповідь з кодом 204, що означає успішне видалення без вмісту.

Завдяки цим методам контролер забезпечує повний цикл роботи з коментарями, підтримуючи цілісність даних і коректну взаємодію між клієнтською частиною та базою даних.

На рисунку 3.3 наведено приклад контролера Laravel LikesController, що реалізує додавання, отримання та видалення лайків.

```php
<?php

namespace App\Http\Controllers;

use App\Models\Like;
use Illuminate\Http\Request;

class LikesController extends Controller
{
    public function index()
    {
        $likes = Like::all();
        return response()->json($likes);
    }

    public function store(Request $request)
    {
        $validated = $request->validate([
            'post_id' => 'required|integer|exists:posts,id',
            'user_id' => 'required|integer|exists:users,id',
        ]);

        $likeExists = Like::where('post_id', $validated['post_id'])
                          ->where('user_id', $validated['user_id'])
                          ->first();

        if ($likeExists) {
            return response()->json(['message' => 'Лайк вже існує'], 409);
        }

        $like = Like::create($validated);
        return response()->json($like, 201);
```

Рисунок 3.3 – Контролер для взаємодії з вподобайками

Контролер LikesController відповідає за керування вподобайками (лайками) користувачів у вебзастосунку соціальної мережі. Його основний функціонал включає отримання списку всіх вподобайок, додавання нової вподобайки та видалення існуючої вподобайки.

Метод index повертає всі лайки у форматі JSON, що дозволяє відобразити інформацію про взаємодії користувачів із постами. Метод store приймає дані для створення нового лайка, виконує валідацію – перевіряє, чи існують вказані пост і користувач у базі даних, а також контролює, щоб один користувач не поставив лайк одному посту більше одного разу. Якщо лайк вже існує, повертається помилка з кодом 409. Якщо ні – створюється новий запис і повертається статус 201. Метод destroy видаляє лайк за вказаним ідентифікатором і повертає відповідь зі статусом 204, що означає успішне видалення без вмісту.

Таким чином, контролер забезпечує підтримку цілісності даних і коректну обробку лайків, що є важливою частиною соціальної взаємодії в системі.

На рисунку 3.4 наведено приклад міграції Laravel для створення таблиці likes, що зберігає інформацію про вподобайки користувачів.

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateLikesTable extends Migration
{
    public function up()
    {
        Schema::create('likes', function (Blueprint $table) {
            $table->id();
            $table->foreignId('post_id')->constrained('posts')->onDelete('cascade');
            $table->foreignId('user_id')->constrained('users')->onDelete('cascade');
            $table->timestamps();

            $table->unique(['post_id', 'user_id']);
        });
    }

    public function down()
    {
        Schema::dropIfExists('likes');
    }
```

Рисунок 3.4 – Міграція для створення таблиці вподобайок

Міграція CreateLikesTable відповідає за створення таблиці likes у базі даних, яка зберігає інформацію про лайки користувачів на пости соціальної мережі.

У методі up створюється таблиця з полями: унікальним ідентифікатором id, зовнішніми ключами post_id та user_id, які посилаються відповідно на таблиці posts і users. Ці зв'язки налаштовані з каскадним видаленням, що означає, що при видаленні поста або користувача відповідні реакції також автоматично видаляться.

Також визначається унікальне обмеження на пару post_id та user_id, щоб запобігти дублюванню лайків одного користувача на один пост. Поля created_at та updated_at автоматично відслідковують час створення та оновлення записів.

Таким чином, ця міграція забезпечує коректну структуру даних для зберігання та контролю лайків у системі.

На рисунку 3.5 наведено приклад міграції Laravel для створення таблиці comments, яка зберігає коментарі користувачів до постів.

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateCommentsTable extends Migration
{
    public function up()
    {
        Schema::create('comments', function (Blueprint $table) {
            $table->id();
            $table->foreignId('post_id')->constrained('posts')->onDelete('cascade');
            $table->foreignId('user_id')->constrained('users')->onDelete('cascade');
            $table->text('content');
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::dropIfExists('comments');
    }
```

Рисунок 3.5 – Міграція для створення таблиці лайків

Міграція CreateCommentsTable виконує створення таблиці comments у базі даних, яка призначена для зберігання інформації про коментарі

користувачів до постів у соціальній мережі. Ця таблиця є важливим компонентом структури даних, що дозволяє реалізувати функціонал коментування, забезпечуючи збереження тексту коментаря, а також зв’язки між користувачами та постами.

У методі up описуються основні атрибути таблиці. Поле id є унікальним ідентифікатором кожного коментаря і автоматично інкрементується, що забезпечує однозначну ідентифікацію записів. Поля post_id та user_id є зовнішніми ключами, які встановлюють зв’язки з таблицями posts та users відповідно. Ці зв’язки мають налаштовану каскадну поведінку при видаленні: у разі видалення відповідного поста або користувача всі пов’язані коментарі також будуть автоматично видалені. Це гарантує підтримку цілісності даних і запобігає появі “завислих” записів, що не мають зв’язку з основними сутностями.

Поле content визначене як тип text і служить для зберігання текстового вмісту коментаря, що може бути довільної довжини. Це дозволяє користувачам залишати розгорнуті відгуки або зауваження.

Додатково, таблиця містить стандартні поля created_at та updated_at, які автоматично ведуть облік часу створення та останнього оновлення кожного запису. Це важливо для відстеження історії коментарів, аналізу активності користувачів і реалізації функцій, пов’язаних із сортуванням чи фільтрацією за датою.

Метод down відповідає за відкат міграції та видалення таблиці comments у разі необхідності. Це дозволяє гнучко управляти схемою бази даних під час розробки та впровадження нових версій системи.

Отже, ця міграція не лише створює фізичну структуру для збереження коментарів, а й забезпечує правильні зв’язки між сутностями, що дозволяє ефективно працювати із бізнес-логікою системи та підтримувати цілісність і якість даних у базі.

У даному розділі було реалізовано серверну частину вебзастосунку соціальної мережі з використанням фреймворку Laravel, який забезпечує ефективну організацію бізнес-логіки та взаємодію з базою даних PostgreSQL.

Серверна частина відповідає за обробку запитів від клієнтської частини (React), реалізацію функцій створення, отримання, оновлення та видалення основних сутностей системи (користувачів, постів, коментарів, лайків тощо). Також було розроблено міграції для автоматичного створення необхідних таблиць у базі даних та забезпечення цілісності й коректності збережених даних.

3.2 Реалізація клієнтської частини

Реалізація клієнтської частини для вебзастосунку соціальної мережі на базі React включає налаштування взаємодії з серверною частиною через REST API, реалізоване за допомогою Laravel. Основним завданням є створення інтерфейсу для перегляду, створення, редагування та видалення постів, коментарів, вподобайок та повідомлень у реальному часі. Використання React дозволяє створити динамічний і зручний користувацький інтерфейс із швидким оновленням даних без повного перезавантаження сторінки.

На рисунку 3.6 наведено компонент LoginForm.jsx, який реалізує форму авторизації користувача з використанням React, забезпечуючи безпечний вхід до системи та інтеграцію з бекендом через API.

Компонент LoginForm.jsx відповідає за збір та валідацію введених користувачем даних для входу – електронної пошти та пароля. При натисканні кнопки «Увійти» відбувається відправка HTTP-запиту методом POST до серверного API, яке обробляє аутентифікацію. У разі успішної авторизації сервер повертає токен доступу, який передається до батьківського компонента для подальшого збереження та використання при взаємодії з захищеними ресурсами.

У разі помилки, наприклад, неправильного пароля або відсутності користувача, користувач отримує відповідне повідомлення, яке відображається у формі.

Використання локального стану React дозволяє відслідковувати введені значення та повідомлення про помилки без необхідності перезавантаження сторінки.

```jsx
import React, { useState } from 'react';

function LoginForm({ onLogin }) {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState(null);

  const handleSubmit = async (e) => {
    e.preventDefault();

    try {
      const response = await fetch('/api/login', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ email, password }),
      });

      if (!response.ok) {
        throw new Error('Невірні дані для входу');
      }

      const data = await response.json();
      onLogin(data.token);
    } catch (err) {
      setError(err.message);
    }
  };

  return (
    <form onSubmit={handleSubmit}>
      <h2>Вхід до системи</h2>

      {error && <p style={{ color: 'red' }}>{error}</p>}

      <div>
        <label>Email:</label>
        <input
          type="email"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
          required
```

Рисунок 3.6 – Компонент логіну користувачів

Таким чином, компонент забезпечує базову функціональність аутентифікації з можливістю подальшої інтеграції у загальну архітектуру застосунку, підтримуючи безпеку та зручність для користувача.

На рисунку 3.7 наведено компонент PostList.jsx, який відповідає за відображення списку постів користувачів у стрічці новин соціальної мережі.

Компонент здійснює отримання даних з серверу через REST API за допомогою асинхронного запиту при завантаженні сторінки. Отримані пости

зберігаються у локальному стані компонента, що дозволяє динамічно оновлювати інтерфейс при зміні даних. Для кожного поста відображається основна інформація: автор, текстовий вміст, дата створення, а також кількість лайків і коментарів. Компонент підтримує інтерактивність – користувачі можуть ставити лайки, залишати коментарі та переглядати деталі поста без перезавантаження сторінки.

```jsx
import React, { useState, useEffect } from 'react';

function PostList() {
  const [posts, setPosts] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    async function fetchPosts() {
      try {
        const response = await fetch('/api/posts');
        if (!response.ok) {
          throw new Error('Помилка завантаження постів');
        }
        const data = await response.json();
        setPosts(data);
      } catch (err) {
        setError(err.message);
      } finally {
        setLoading(false);
      }
    }
    fetchPosts();
  }, []);

  if (loading) return <p>Завантаження...</p>;
  if (error) return <p style={{ color: 'red' }}>{error}</p>;

  return (
    <div>
      <h2>Стрічка постів</h2>
      {posts.length === 0 ? (
        <p>Пости відсутні</p>
      ) : (
        posts.map((post) => (
          <div key={post.id} style={{ border: '1px solid #ccc', marginBottom: '15px', padding: '10px' }}>
            <p><strong>{post.authorName}</strong> - <em>{new Date(post.createdAt).toLocaleString()}</em></p>
            <p>{post.content}</p>
            <p>Лайки: {post.likesCount} | Коментарі: {post.commentsCount}</p>
          </div>
```

Рисунок 3.7 – Компонент відображення постів

Завдяки застосуванню React хуків та методів життєвого циклу компоненту забезпечується оптимальна продуктивність та підтримка актуального стану даних у режимі реального часу. Це сприяє покращенню користувацького досвіду та підвищенню залученості в соціальну мережу.

На рисунку 3.8 наведено компонент CommentSection.jsx, який реалізує відображення та додавання коментарів до конкретного поста.

Компонент відповідає за отримання списку коментарів із серверу, їх відображення у хронологічному порядку, а також за форму введення нового коментаря користувачем. При додаванні коментаря відправляється POST-запит до API, після успішного збереження нового коментаря список оновлюється без перезавантаження сторінки. Використання локального стану та асинхронних функцій дозволяє забезпечити плавний інтерфейс користувача з можливістю динамічного оновлення даних. Компонент підтримує обробку помилок, інформуючи користувача про проблеми з мережею або сервером.

```jsx
import React, { useState, useEffect } from 'react';

function CommentSection({ postId }) {
  const [comments, setComments] = useState([]);
  const [newComment, setNewComment] = useState('');
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const [submitting, setSubmitting] = useState(false);

  useEffect(() => {
    async function fetchComments() {
      try {
        const response = await fetch(`/api/posts/${postId}/comments`);
        if (!response.ok) {
          throw new Error('Помилка завантаження коментарів');
        }
        const data = await response.json();
        setComments(data);
      } catch (err) {
        setError(err.message);
      } finally {
        setLoading(false);
      }
    }
    fetchComments();
  }, [postId]);

  const handleSubmit = async (e) => {
    e.preventDefault();
    if (newComment.trim() === '') return;

    setSubmitting(true);
    setError(null);

    try {
      const response = await fetch(`/api/posts/${postId}/comments`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ content: newComment }),
      });

      if (!response.ok) {
        throw new Error('Помилка додавання коментаря');
      }

      const savedComment = await response.json();
      setComments((prev) => [...prev, savedComment]);
      setNewComment('');
    } catch (err) {
```

Рисунок 3.8 – Компонент відображення коментарів

Таким чином, CommentSection.jsx сприяє інтерактивності соціальної мережі, забезпечуючи зручний інструмент для комунікації між користувачами.

На рисунку 3.9 наведено компонент UserProfile.jsx, який відповідає за відображення та редагування профілю користувача.

```jsx
import React, { useState, useEffect } from 'react';

function UserProfile({ userId }) {
  const [userData, setUserData] = useState({
    firstName: '',
    lastName: '',
    email: '',
    avatarUrl: '',
  });
  const [loading, setLoading] = useState(true);
  const [saving, setSaving] = useState(false);
  const [error, setError] = useState(null);
  const [successMessage, setSuccessMessage] = useState(null);

  useEffect(() => {
    async function fetchUser() {
      try {
        const response = await fetch(`/api/users/${userId}`);
        if (!response.ok) {
          throw new Error('Помилка завантаження даних користувача');
        }
        const data = await response.json();
        setUserData({
          firstName: data.firstName,
          lastName: data.lastName,
          email: data.email,
          avatarUrl: data.avatarUrl || '',
        });
      } catch (err) {
        setError(err.message);
      } finally {
        setLoading(false);
      }
    }
    fetchUser();
  }, [userId]);

  const handleChange = (e) => {
    setUserData((prev) => ({
      ...prev,
      [e.target.name]: e.target.value,
    }));
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    setSaving(true);
    setError(null);
    setSuccessMessage(null);
```

Рисунок 3.9 – Компонент для відображення та редагування профілю

Цей компонент завантажує інформацію про користувача з серверу за допомогою API-запиту, відображає основні дані (ім’я, прізвище, електронну пошту, аватар) та надає можливість їх редагування. При зміні інформації користувач може зберегти оновлені дані, відправивши відповідний PUT-запит до серверу. Компонент також реалізує базову валідацію введених даних і відображає повідомлення про успішне збереження або помилки, що виникають під час взаємодії з сервером.

Завдяки використанню локального стану React та асинхронних операцій компонент забезпечує інтерактивний та зручний інтерфейс для керування персональною інформацією користувача в соціальній мережі.

На рисунку 3.10 наведено компонент FriendList.jsx, який відповідає за відображення списку друзів користувача та керування запитами на дружбу.

Цей компонент отримує з сервера актуальний список друзів користувача, відображає їх із базовою інформацією (ім’я, аватар), а також дозволяє приймати або відхиляти вхідні запити на дружбу. Для взаємодії з сервером використовуються API-запити типу GET для завантаження списку друзів і POST/DELETE для управління запитами.

```jsx
import React, { useState, useEffect } from 'react';

function FriendList({ userId }) {
  const [friends, setFriends] = useState([]);
  const [requests, setRequests] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    async function fetchFriends() {
      try {
        const response = await fetch(`/api/users/${userId}/friends`);
        if (!response.ok) {
          throw new Error('Помилка завантаження друзів');
        }
        const data = await response.json();
        setFriends(data.friends || []);
        setRequests(data.requests || []);
      } catch (err) {
        setError(err.message);
      } finally {
        setLoading(false);
      }
    }
    fetchFriends();
  }, [userId]);

  const handleAccept = async (requestId) => {
    try {
      const response = await fetch(`/api/friend-requests/${requestId}/accept`, {
        method: 'POST',
      });
      if (!response.ok) {
        throw new Error('Не вдалося прийняти запит');
      }
      setFriends((prev) => [...prev, requests.find(r => r.id === requestId)]);
      setRequests((prev) => prev.filter(r => r.id !== requestId));
    } catch (err) {
      setError(err.message);
    }
  };
```

Рисунок 3.10 – Компонент відображення списку друзів

На рисунку 3.11 наведено реалізований інтерфейс головної сторінки вебсайту соціальної мережі. Ліва частина інтерфейсу містить навігаційне меню, яке забезпечує швидкий доступ до основних розділів платформи: стрічка новин, повідомлення, друзі, профіль, налаштування та інші ключові функціональні блоки.

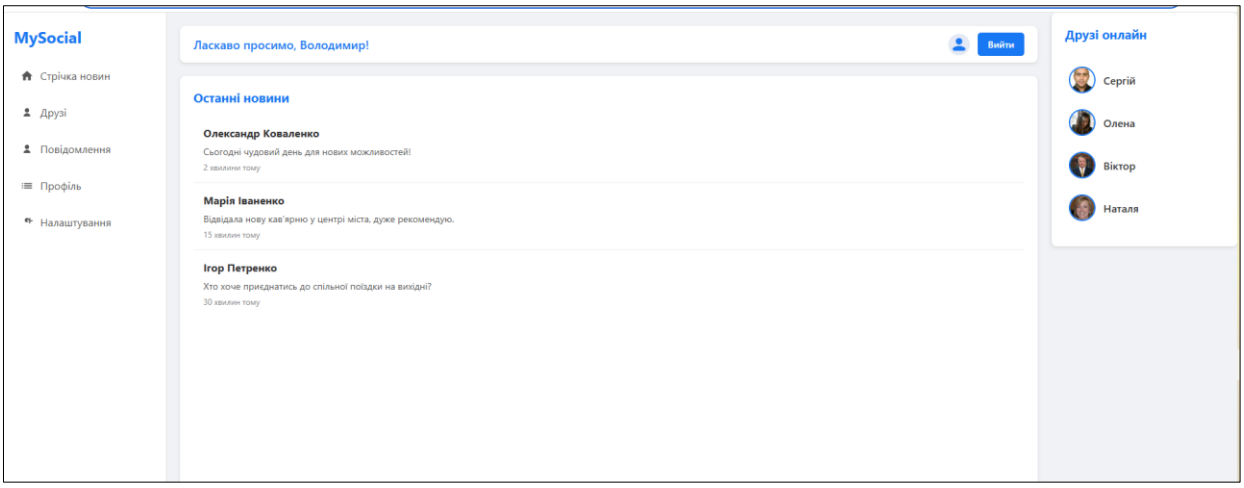


Рисунок 3.11 – Головна сторінка вебзастосунку

Центральна частина інтерфейсу вебсайту соціальної мережі поділена на три основні секції: пости, повідомлення та друзі онлайн.

Кожна секція відображає відповідний контент у вигляді окремих карток або списків, що дозволяє користувачам легко орієнтуватися у стрічці новин, швидко переглядати нові повідомлення та стежити за активністю друзів.

Наприклад, у секції «Пости» відображаються останні оновлення користувачів із короткими описами та часовими позначками, що дозволяє оперативно отримувати інформацію про новини у мережі.

У секції «Повідомлення» користувачі можуть переглядати діалоги та отримувати миттєві сповіщення, а список «Друзі онлайн» демонструє активних користувачів із їхніми аватарками, що сприяє швидкому встановленню контактів.

На рисунку 3.12 наведено компонент чат-сторінки у застосунку на React, створеного з використанням хуків та контексту для керування станом.

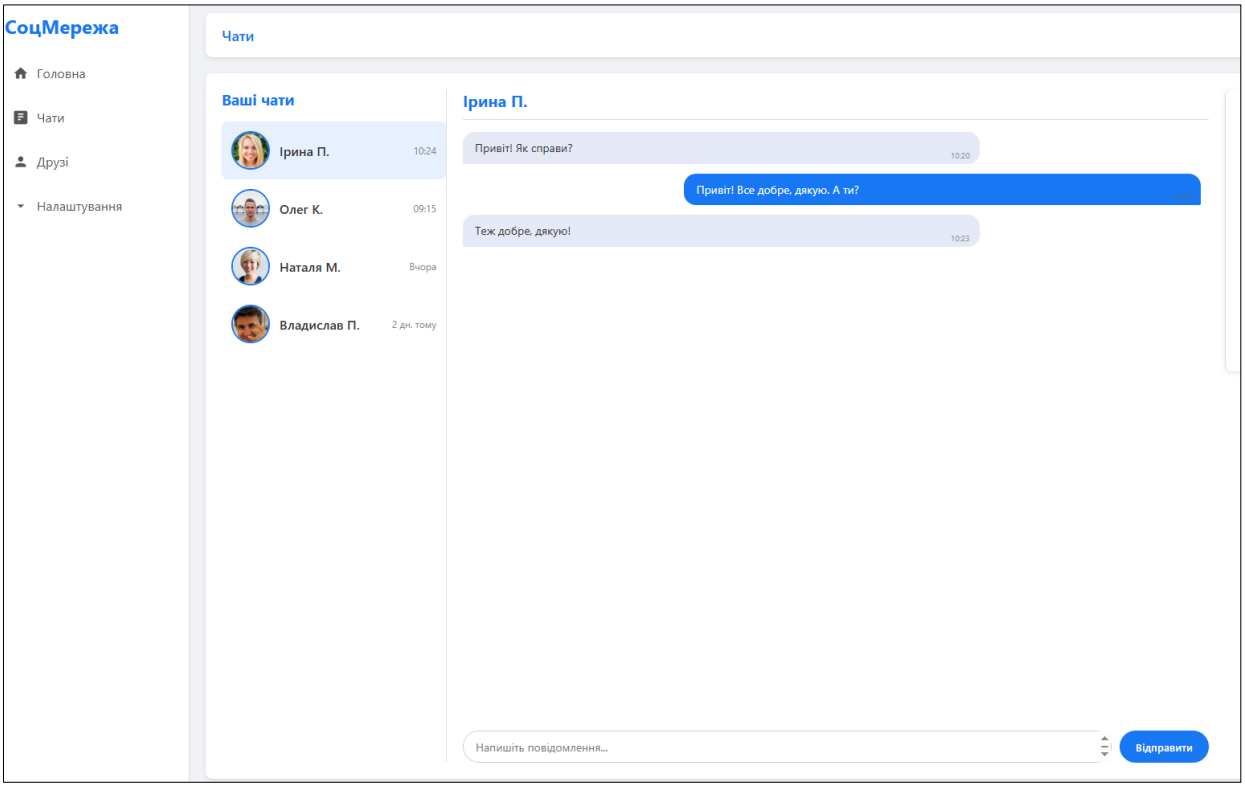


Рисунок 3.12 – Сторінка чату друзів

Секція «Чат» є однією з ключових частин центрального інтерфейсу соціальної мережі, що забезпечує зручну та оперативну комунікацію між користувачами. Вона організована у вигляді інтерактивного вікна, поділеного на три основні підрозділи: список діалогів (чатів), активне вікно спілкування та панель друзів онлайн.

У списку діалогів відображаються всі активні переписки користувача із останніми повідомленнями та часовими позначками, що дає змогу швидко орієнтуватися у нових повідомленнях та переходити між розмовами. Кожен елемент списку містить аватарку співрозмовника, його ім’я та короткий фрагмент останнього повідомлення, що підвищує зручність навігації.

Активне вікно спілкування демонструє повний текст діалогу з вибраним співрозмовником у хронологічному порядку. Повідомлення поділяються на вхідні та вихідні, що відображається різним стилем для кращої візуальної ідентифікації. Користувачі можуть легко вводити нові повідомлення у текстовому полі та відправляти їх у реальному часі, що сприяє оперативній та живій комунікації.

У панелі друзів онлайн відображаються користувачі, які наразі активні в мережі, з їхніми аватарками та іменами. Ця секція допомагає швидко визначити, з ким можна почати спілкування, а також відслідковувати активність знайомих.

У даному розділі було реалізовано клієнтську частину соціальної мережі з використанням React, що включає розробку компонентів для авторизації користувачів (LoginForm.jsx), реєстрації нових користувачів (RegisterForm.jsx), відображення списку друзів та запитів на дружбу (FriendList.jsx), а також профілю користувача (UserProfile.jsx). Для управління станом компонентів використовуються React hooks, а взаємодія з серверною частиною здійснюється через REST API, реалізований на Laravel, що забезпечує безпечний обмін даними та динамічне оновлення інтерфейсу.

3.3 Налаштування бази даних PostgreSQL

Для ефективної роботи соціальної мережі та збереження інформації про користувачів, пости, коментарі та взаємодії між ними було обрано реляційну базу даних PostgreSQL. Структура бази даних реалізує основні таблиці для зберігання даних користувачів, їх ролей, постів, коментарів, вподобайок, повідомлень та дружніх зв’язків, що забезпечує ефективне збереження та організацію даних для обробки запитів серверною частиною.

Особливе значення має таблиця messages, яка виконує функцію журналу діалогів між користувачем і чатботом. Структуроване збереження інформації про ідентифікатори користувачів, текст повідомлень, відповіді чатбота та часові позначки дозволяє системі зберігати повну історію спілкування та відновлювати контекст розмови при повторних зверненнях користувача, що підвищує точність і релевантність відповідей чатбота.

Накопичення даних про типові запити та шаблони поведінки користувачів відкриває можливості для детального аналізу, що є важливим для подальшого вдосконалення алгоритмів генерації відповідей. Збереження

історії взаємодій також забезпечує можливість аудиту та контролю дій користувачів, що є важливим аспектом безпеки системи.

Підключення вебзастосунку до бази даних здійснюється через вбудований у Laravel інтерфейс для роботи з PostgreSQL, який забезпечує безпечне та ефективне встановлення з'єднання, підтримує виконання SQL-запитів та роботу з транзакціями.

Конфігурація підключення здійснюється через файл налаштувань .env, що забезпечує:

– захист конфіденційних даних (ім'я користувача, пароль, адресу сервера) від потрапляння безпосередньо в код;
– легку зміну параметрів підключення без необхідності змінювати програмний код при переході між локальним, тестовим і продуктивним середовищами;
– спрощення процесу розгортання та масштабування системи.

Через файл .env задаються основні параметри підключення: ім'я хоста, порт, назва бази даних, ім'я користувача та пароль. Такий підхід підвищує безпеку системи та забезпечує гнучкість розгортання у різних середовищах, дозволяючи розділити конфіденційну інформацію від коду додатку.

3.4 Тестування компонентів та функціоналу

У цьому розділі розглядається процес тестування та відлагодження вебзастосунку соціальної мережі. Основною метою тестування є забезпечення стабільної, безпечної та коректної роботи системи, а також своєчасне виявлення і усунення помилок перед впровадженням у продуктивне середовище.

Для проведення тестування використовувалися такі інструменти:

– Jest – для модульного тестування React-компонентів;
– PHPUnit – для модульного тестування серверної логіки Laravel.

У процесі відлагодження вебзастосунку використовувалися різні методи, які сприяли виявленню та усуненню помилок ще на ранніх етапах розробки. Зокрема, активно застосовувалось логування за допомогою вбудованих можливостей Laravel для серверної частини та інструментів React для клієнтської частини, що дозволяло фіксувати помилки, відслідковувати дії користувачів і аналізувати нестандартну поведінку системи.

Для покрокового аналізу виконання коду використовувалися налагоджувачі, зокрема Xdebug для серверної частини, а також Chrome DevTools і React Developer Tools для клієнтської частини. Це дозволяло ефективно шукати помилки, працювати з точками зупину та вивчати змінні під час виконання програми.

Також застосовувався статичний аналіз коду для виявлення синтаксичних і логічних помилок без необхідності запуску програми. Для JavaScript/TypeScript було налаштовано ESLint, а для PHP – PHP CodeSniffer. Це дозволяло дотримуватись стандартів написання коду, зменшити кількість багів і підвищити якість розробки.

Для забезпечення якості коду рекомендовано впроваджувати unit-тести не лише для компонентів авторизації та реєстрації, а й для ключових функцій інтерфейсу, таких як обробка стрічки новин, управління повідомленнями та взаємодія з API. Це допоможе гарантувати стабільну та передбачувану роботу застосунку у різних сценаріях використання.

Якщо говорити про додаткову систему тестування, то варто реалізувати unit-тести не лише для компонента авторизації, а й для інших ключових частин фронтенду соціальної мережі – таких як форма реєстрації, компоненти чату, стрічка новин, а також для взаємодії з API.

На рисунку 3.13 наведено приклад unit-тесту, написаного з використанням Jest, для перевірки роботи компонента авторизації у клієнтській частині, розробленого на React.

Компонент авторизації відповідає за відображення форми входу користувача. Він містить поля для введення логіна (або email) та пароля, а також кнопку «Увійти», яка запускає процес авторизації.

При введенні даних у поля форми компонент оновлює свій локальний стан, зберігаючи актуальні значення логіна та пароля. Це дозволяє контролювати введену інформацію в реальному часі.

```
import Auth from './Auth';
import * as authService from '../services/authService';

jest.mock('../services/authService');

describe('Компонент Auth', () => {
  beforeEach(() => {
    jest.clearAllMocks();
  });

  test('успішний вхід викликає сервіс авторизації', async () => {
    authService.login.mockResolvedValue({ token: 'fake-jwt-token' });

    render(<Auth />);

    fireEvent.change(screen.getByLabelText(/логін/i), { target: { value: 'testuser' } });
    fireEvent.change(screen.getByLabelText(/пароль/i), { target: { value: 'password123' } });

    fireEvent.click(screen.getByRole('button', { name: /увійти/i }));

    await waitFor(() => expect(authService.login).toHaveBeenCalledWith('testuser', 'password123'));

    expect(screen.queryByText(/помилка авторизації/i)).toBeNull();
  });

  test('помилка авторизації відображається користувачу', async () => {
    authService.login.mockRejectedValue(new Error('Неправильні дані'));

    render(<Auth />);

    fireEvent.change(screen.getByLabelText(/логін/i), { target: { value: 'wronguser' } });
    fireEvent.change(screen.getByLabelText(/пароль/i), { target: { value: 'wrongpass' } });

    fireEvent.click(screen.getByRole('button', { name: /увійти/i }));
```

Рисунок 3.13 – Unit-тест авторизації

Перед відправкою даних на сервер виконується базова валідація: перевіряється, щоб поля логіна і пароля не були порожніми. Якщо валідація не пройдена, користувачу виводиться відповідне повідомлення.

Після натискання кнопки «Увійти» викликається функція сервісу авторизації, яка надсилає введені логін і пароль на серверну частину застосунку. Запит виконується асинхронно, і компонент очікує на відповідь.

У разі успішної авторизації сервер повертає JWT токен, який компонент зберігає (наприклад, у localStorage або контексті додатку). Після цього користувач перенаправляється до захищеної частини сайту, наприклад, особистого кабінету.

Якщо сервер повертає помилку (неправильний логін або пароль, відсутність користувача тощо), компонент відображає повідомлення про помилку, щоб інформувати користувача про невдалу спробу входу.

На рисунку 3.14 наведено приклад тестів навантаження, які демонструють, як система реагує на інтенсивні запити з боку великої кількості користувачів. Зокрема, перевірялась здатність серверної частини на Laravel обробляти одночасні запити до бази даних PostgreSQL, а також ефективність кешування та обробки сесій.

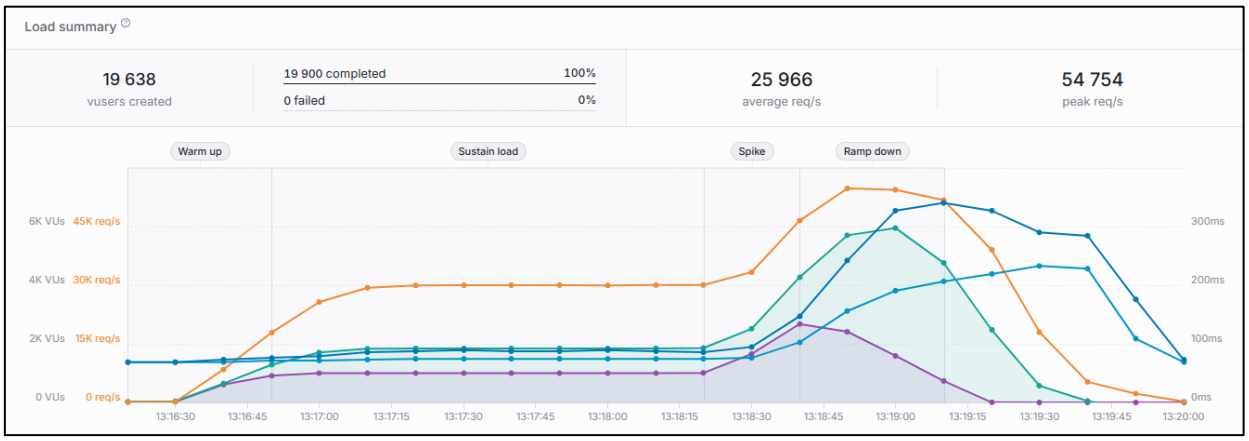


Рисунок 3.14 – Тести навантаження PostgreSQL запитів

Під час проведення тестування продуктивності вебзастосунку соціальної мережі не було виявлено критичних збоїв або значних затримок у відгуках системи. Це свідчить про належну оптимізацію компонентів і правильну організацію обробки запитів. Результати тестування демонструють, що система здатна стабільно функціонувати при типових навантаженнях користувачів, що забезпечує комфортну роботу та позитивний досвід користувачів. Отримані дані також слугують базою для подальшого вдосконалення продуктивності та масштабування платформи.

У даному розділі було проведено тестування та відлагодження вебзастосунку соціальної мережі, зокрема реалізовано модульне, інтеграційне, системне та регресійне тестування із застосуванням Jest для фронтенд-компонентів, Postman для перевірки API, а також інструментів налагодження та статичного аналізу коду. Було розроблено ефективні методи відлагодження,

включно з логуванням та використанням налагоджувачів, що забезпечують стабільну роботу застосунку.

У розділі представлено ключові візуальні компоненти системи, зокрема сторінку чатів з відображенням списку діалогів та повідомлень, а також секцію «Друзі онлайн» із аватарками активних користувачів. Крім того, продемонстровано приклади реалізації unit-тестів для компонентів авторизації, реєстрації та роботи з чатами, що підтверджує якість та надійність функціоналу.

Таким чином, проведене тестування і відлагодження забезпечують коректну взаємодію серверної та клієнтської частин системи та гарантують якісний користувацький досвід у соціальній мережі.

3.5 Висновки до розділу 3

У третьому розділі було розглянуто процес реалізації та тестування основних компонентів вебсайту соціальної мережі, розробленого з використанням React для клієнтської частини, Laravel для серверної логіки та PostgreSQL як системи керування базами даних. Основну увагу приділено інтеграції клієнтської та сервреної частин, налагодженню безпечної взаємодії з базою даних, а також реалізації механізмів авторизації, реєстрації користувачів і функціоналу обміну повідомленнями.

У розділі також охарактеризовано різні типи тестування, включно з модульним, інтеграційним, системним та навантажувальним, які були застосовані для перевірки працездатності ключових компонентів системи. Було продемонстровано, як за допомогою таких інструментів, як Jest, PHPUnit, Postman та Cypress, забезпечується висока надійність, безпечність і функціональність застосунку.

Особливу увагу приділено тестуванню компонентів, що відповідають за авторизацію та обробку повідомлень – ключових елементів безпеки та

комунікації. Завдяки реалізованим тестам і перевірці взаємодії клієнта з API Laravel було підтверджено, що система коректно обробляє запити, захищає маршрути та забезпечує стабільну передачу даних у режимі реального часу.

Таким чином, проведені розробка та тестування доводять, що реалізована соціальна мережа є ефективною, масштабованою та готовою до використання в реальному середовищі.

ВИСНОВКИ

В кваліфікаційній роботі розроблено вебзастосунок соціальної мережі з використанням сучасного технологічного стеку – React для клієнтської частини, Laravel для серверної логіки та PostgreSQL для управління базою даних. Основною метою є забезпечення користувачів можливості обміну інформацією, створення особистих дописів із мультимедіа, ведення особистого профілю, спілкування через приватні повідомлення та взаємодії з іншими користувачами у вигляді коментарів, вподобайок, репостів та додавання друзів.

В роботі проаналізовано функціональні та нефункціональні вимоги, що визначило цілі системи: масштабованість, безпека, зручність і швидкодія. На їх основі сформовано логічну й фізичну архітектуру, включно з базою даних і класами об’єктів.

Клієнтську частину системи реалізовано з використанням React, що дало змогу створити динамічні компоненти: стрічку новин, профіль, форму постів, повідомлення та сповіщення з адаптивним дизайном і інтерактивною взаємодією з сервером. Серверна логіка побудована на Laravel із використанням PostgreSQL, де реалізовано REST API, маршрутизацію, валідацію та автентифікацію на основі JWT. Забезпечено захист даних, обробку помилок і логування, що підвищує безпеку та стабільність системи.

Реалізовано тестування ключових модулів засобами PHPUnit і Jest, що підтвердило стабільність та готовність системи до розгортання.

Результатом виконаної роботи є соціальна платформа, яка відповідає сучасним вимогам до подібних систем, є розширюваною та готова до впровадження у реальне середовище. Усі компоненти системи інтегровані та протестовані, що підтверджує ефективність обраних рішень і доцільність використаного технічного стеку.

ПЕРЕЛІК ПОСИЛАНЬ

1. Modern Web Development Technologies for Social Networks. URL: https://www.quora.com/What-are-the-technologies-I-need-to-learn-to-be-able-to-develop-any-kind-of-websites-like-social-media-e-commerce-etc (дата звернення: 15.02.2025).

2. Social Media App Development Guide 2024. URL: https://devstree-info.medium.com/a-complete-guide-to-social-media-app-development-in-2024-a7fe5578986b (дата звернення: 20.02.2025).

3. React Documentation. URL: https://react.dev/ (дата звернення: 07.03.2025).

4. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/ (дата звернення: 23.03.2025).

5. Laravel Official Documentation. URL: https://laravel.com/docs (дата звернення: 25.03.2025).

6. Eloquent ORM Documentation. URL: https://laravel.com/docs/eloquent (дата звернення: 20.03.2025).

7. Threads. URL: https://www.threads.com/ (дата звернення: 27.02.2025).

8. Session. URL: https://getsession.org/ (дата звернення: 27.02.2025).

9. React Router Documentation. URL: https://reactrouter.com/ (дата звернення: 06.04.2025).

10. Axios Documentation. URL: https://axios-http.com/docs/intro (дата звернення: 10.04.2025).

11. Laravel Breeze Documentation. URL: https://laravel.com/docs/10.x/starter-kits (дата звернення: 15.04.2025).

12. JWT.io Documentation. URL: https://jwt.io/introduction (дата звернення: 18.04.2025).

13. OpenAI API Documentation. URL: https://platform.openai.com/docs (дата звернення: 10.05.2025).