

**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 ВЕБЗАСТОСУНКУ СОЦІАЛЬНОЇ МЕРЕЖІ З
ВИКОРИСТАННЯМ NODE.JS ТА REACT**

**SOCIAL NETWORK WEB APPLICATION DEVELOPMENT USING
NODE.JS AND REACT**

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

Виконала: здобувачка вищої освіти
групи ІПЗ-43
Котловець Аліна Володимирівна

Керівник:
Ліщина Наталія Миколаївна

Кваліфікаційну роботу
допущено до захисту
«__» _____ 20__ р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

Луцьк – 2026 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__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Котловець Аліні Володимирівні

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи «Розробка вебзастосунку соціальної мережі з використанням Node.js та React»

Керівник роботи: к.т.н., доцент Ліщина Н. М.

затверджені наказом закладу вищої освіти від «31» грудня 2025 р. № 541/01-02

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра «29» травня 2026 р.

3. Вихідні дані до роботи: предметна область соціальних вебзастосунків та онлайн-комунікації користувачів, сучасні технології розробки вебзастосунків (JavaScript, TypeScript, Node.js, Express, React), засоби роботи з базами даних (PostgreSQL, Prisma ORM), підходи до проєктування клієнт-серверних систем, вимоги до продуктивності, масштабованості та безпеки інформаційних систем.

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): аналіз сучасного стану соціальних вебзастосунків та існуючих платформ обміну інформацією, постановка задачі та формування вимог до розроблюваної системи, обґрунтування вибору архітектури та технологій розробки, проєктування структури вебзастосунку та його компонентів, практична реалізація клієнтської та серверної частин системи, розробка структури бази даних та організація доступу до неї, тестування та налагодження програмного забезпечення.

5. Перелік графічного матеріалу: 10 рисунків, 12 лістингів.

6. Консультанти розділів роботи

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

7. Дата видачі завдання «3» лютого 2026 р.

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 14.02.2026 р.	
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 03.03.2026 р.	
3	Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 14.03.2026 р.	
4	Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних	до 14.04.2026 р.	
5	Практична реалізація об'єкта проектування та розробка бази даних	до 20.04.2026 р.	
6	Тестування та налагодження об'єкта проектування	до 04.05.2026 р.	
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедрі	до 29.05.2026 р.	

Здобувач вищої освіти _____

Аліна КОТЛОВЕЦЬ

Керівник кваліфікаційної роботи _____

Наталія ЛІЩИНА

АНОТАЦІЯ

Котловець А. В. Розробка вебзастосунку соціальної мережі з використанням Node.js та React. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2026.

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

У першому розділі проведено аналіз предметної області соціальних вебзастосунків, розглянуто сучасні платформи обміну інформацією, проаналізовано їх функціональні можливості, переваги та недоліки, а також обґрунтовано доцільність створення власного програмного продукту. У другому розділі сформовано вимоги до розроблюваної системи, виконано проєктування архітектури вебзастосунку із використанням клієнт-серверної моделі, застосовано методи моделювання програмного забезпечення, а також обґрунтовано вибір технологій, інструментів та підходів до реалізації системи. У третьому розділі описано процес практичної реалізації вебзастосунку, зокрема розробку серверної та клієнтської частин, організацію взаємодії між ними, створення структури бази даних, реалізацію механізмів автентифікації та авторизації користувачів, забезпечення захисту інформації, а також проведено тестування і налагодження програмного забезпечення. У висновках узагальнено результати виконаної роботи, підтверджено досягнення поставленої мети та повне виконання визначених завдань.

Ключові слова: вебзастосунок, соціальна мережа, клієнт-серверна архітектура, Node.js, React, TypeScript, PostgreSQL, Prisma, автентифікація, інформаційна безпека.

ABSTRACT

Kotlovets A. V. Social network web application development using Node.js and React. Manuscript. Bachelor's qualification work in the Educational Program "Software Engineering" in the specialty "Software Engineering". Lutsk National Technical University. Lutsk, 2026.

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

The first chapter analyzes the subject area of social web applications, examines modern information exchange platforms, analyzes their functional capabilities, advantages, and disadvantages, and substantiates the feasibility of developing a custom software product. The second chapter defines the requirements for the developed system, presents the design of the web application architecture using a client-server model, applies software modeling methods, and justifies the choice of technologies, tools, and approaches for system implementation. The third chapter describes the process of practical implementation of the web application, including the development of the server and client parts, organization of their interaction, database structure design, implementation of authentication and authorization mechanisms, ensuring information security, as well as testing and debugging of the software. The conclusions summarize the results of the work, confirm the achievement of the stated goal, and the complete fulfillment of the defined tasks.

Keywords: web application, social network, client-server architecture, Node.js, React, TypeScript, PostgreSQL, Prisma, authentication, information security.

ЗМІСТ

ВСТУП	7
РОЗДІЛ 1 АНАЛІЗ ВЕБЗАСТОСУНКІВ СОЦІАЛЬНИХ МЕРЕЖ І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ	9
1.1 Аналіз сучасного стану проблеми	9
1.2 Постановка завдання на кваліфікаційну роботу бакалавра	13
РОЗДІЛ 2 СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ	15
2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проєктування програмного забезпечення	15
2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання	21
РОЗДІЛ 3 РОЗРОБКА ВЕБЗАСТОСУНКУ СОЦІАЛЬНОЇ МЕРЕЖІ З ВИКОРИСТАННЯМ NODE.JS ТА REACT	27
3.1 Практична реалізація об'єкта проєктування	27
3.2 Тестування та налагодження інформаційно-комп'ютерної системи	44
3.3 Розробка бази даних	47
3.4 Захист інформаційно-комп'ютерної системи	51
3.5 Розробка документації для програмного продукту	54
ВИСНОВКИ	57
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	59

ВСТУП

Актуальність теми. У сучасному інформаційному просторі соціальні мережі стали невід’ємною складовою цифрової комунікації та обміну даними. Вони забезпечують швидке поширення інформації, формування потоків контенту та створення середовища для взаємодії користувачів у режимі реального часу. Активний розвиток вебтехнологій і зростання обсягів даних зумовлюють підвищені вимоги до продуктивності, масштабованості та зручності використання вебзастосунків.

Сучасні платформи обміну інформацією, зокрема Twitter (X) та Threads, демонструють високий рівень розвитку інструментів взаємодії користувачів, поєднуючи можливості створення контенту, системи підписок, формування стрічки новин і пошуку даних. Водночас розвиток таких систем супроводжується ускладненням їх логіки, зокрема використанням алгоритмічного ранжування контенту, що створює додаткові вимоги до обробки даних та організації взаємодії між компонентами системи.

Незважаючи на значний рівень розвитку соціальних вебзастосунків, у даній предметній галузі залишаються актуальними певні проблеми та виклики сюди входить ефективна обробка великої кількості запитів, оптимізація завантаження контенту, зменшення навантаження на серверну частину, забезпечення стабільної роботи інтерфейсу при значних обсягах даних, а також захист персональної інформації користувачів. Окремої уваги потребує питання організації зручного та інтуїтивно зрозумілого інтерфейсу, що забезпечує швидку взаємодію користувача із системою.

Метою роботи є розробка вебзастосунку соціальної мережі для обміну даними з використанням технологій Node.js та React.

Об’єктом дослідження є процеси створення та функціонування соціальних вебзастосунків, а предметом – методи та засоби розробки клієнт-серверних систем із використанням сучасних інструментів фронтенд і бекенд розробки.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- провести аналіз предметної області та існуючих соціальних мереж;
- дослідити сучасні підходи до розробки вебзастосунків та обґрунтувати вибір технологій;
- спроектувати архітектуру вебзастосунку та визначити взаємодію його компонентів;
- розробити структуру бази даних для зберігання інформації про користувачів і контент;
- реалізувати серверну частину застосунку та взаємодію з базою даних;
- реалізувати клієнтську частину із забезпеченням зручного інтерфейсу;
- забезпечити механізми автентифікації, авторизації та захисту даних;
- провести тестування функціоналу та оцінити працездатність системи.

Галузь застосування результатів роботи охоплює інформаційні системи соціальної взаємодії, які призначені для обміну даними, публікації контенту та організації комунікації між користувачами. Виконання даної роботи пов'язане з дослідженням сучасних підходів до розробки вебзастосунків та узагальненням досвіду створення подібних систем, що визначає її взаємозв'язок з іншими роботами у галузі веброзробки, інженерії програмного забезпечення та інформаційних технологій загалом.

РОЗДІЛ 1

АНАЛІЗ ВЕБЗАСТОСУНКІВ СОЦІАЛЬНИХ МЕРЕЖ І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Сучасні вебзастосунки соціальних мереж є однією з найбільш динамічних сфер інформаційних технологій, оскільки вони безпосередньо впливають на способи комунікації, обмін інформацією та соціальну взаємодію користувачів. Як зазначається у наукових дослідженнях, соціальні мережі суттєво трансформували процеси спілкування та формування інформаційного середовища, що обумовлює необхідність їх аналізу як складних соціотехнічних систем.

Важливою характеристикою сучасних соціальних платформ є їх масовість та інтегрованість у повсякденне життя. Вони виконують не лише функцію комунікації, але й виступають середовищем формування суспільної думки, поширення інформації та організації взаємодії між користувачами. У цьому контексті дослідження підкреслюють подвійний характер впливу соціальних мереж: з одного боку, вони розширюють можливості комунікації та доступу до інформації, з іншого – можуть сприяти поширенню дезінформації, формуванню інформаційних бульбашок і маніпуляціям інформаційними потоками.

Аналіз літератури також свідчить, що вплив соціальних мереж формується як на макрорівні, пов'язаному із суспільними процесами, так і на мікрорівні – через взаємодію окремих користувачів із контентом. У цьому процесі значну роль відіграють механізми рекомендацій та поведінкові фактори, які визначають видимість інформації та характер її сприйняття [1].

Соціальні мережі функціонують як середовище постійного обміну контентом, який може мати текстову, візуальну або відеоформу. Дослідження показують, що саме форма та зміст контенту значною мірою визначають реакцію аудиторії та рівень залученості користувачів. При цьому різні типи платформ

мають різну функціональну спрямованість: одні орієнтовані на текстову комунікацію, інші – на візуальний або розважальний контент.

Важливим аспектом є також взаємодія користувачів із контентом, яка реалізується через лайки, коментарі та поширення. Саме ці механізми формують видимість інформації та визначають її вплив у межах платформи. Крім того, встановлено, що соціальні мережі створюють стійкі комунікаційні зв'язки між користувачами та джерелами інформації, що підсилює їх роль як середовища постійної взаємодії [2].

Паралельно із соціальними аспектами, у літературі значна увага приділяється технічним особливостям розробки соціальних вебзастосунків. Визначено, що такі системи повинні забезпечувати ефективну взаємодію між клієнтською та серверною частинами, обробку великої кількості запитів у реальному часі, а також стабільну роботу під навантаженням. Це зумовлює необхідність використання сучасних підходів до оптимізації запитів, асинхронної обробки даних та організації архітектури системи.

У роботах, присвячених розробці соціальних мереж як інформаційних систем, підкреслюється, що їх основними компонентами є профілі користувачів, система підписок, стрічка новин, механізми публікації контенту та засоби взаємодії між користувачами. Також наголошується на важливості забезпечення безпеки даних, конфіденційності користувачів та надійності функціонування системи.

Аналіз існуючих платформ, зокрема Twitter (X), демонструє, що сучасні соціальні мережі забезпечують високу швидкість поширення інформації та широкі можливості для взаємодії, проте водночас стикаються з проблемами інформаційного перевантаження, складності фільтрації контенту та забезпечення безпеки [3].

Таким чином, результати аналізу літературних джерел свідчать, що соціальні мережі є складними системами, у яких поєднуються технологічні, соціальні та поведінкові аспекти. Незважаючи на активний розвиток існуючих платформ, залишається актуальною потреба у створенні вебзастосунків, що

забезпечують ефективну взаємодію користувачів, зручність використання, високу продуктивність та належний рівень безпеки. Це обґрунтовує доцільність розробки власного вебзастосунку соціальної мережі, орієнтованого на сучасні вимоги до інформаційних систем.

Відповідно до наведених особливостей, можна зробити висновок, що сучасні соціальні вебзастосунки знаходяться на етапі активного розвитку, що супроводжується як розширенням функціональних можливостей, так і ускладненням їх внутрішньої структури. Це створює передумови для пошуку нових підходів до проектування систем, які поєднують ефективність обробки даних, зручність користування та прозорість взаємодій.

У сучасному цифровому середовищі існує велика кількість соціальних платформ, які можна розглядати як аналоги проєктованої системи. Найбільш відповідними і відомими серед них є Twitter (X), Threads, Instagram та Facebook. Кожна з цих платформ реалізує базові функції соціальних мереж, проте має власні особливості, що визначають характер взаємодії користувачів та підходи до організації контенту.

Однією з найбільш релевантних платформ для аналізу є Twitter (X), який орієнтований на обмін короткими текстовими повідомленнями. Основною перевагою цієї платформи є висока швидкість поширення інформації, що дозволяє користувачам оперативно реагувати на події та брати участь у глобальних обговореннях. Крім того, Twitter (X) забезпечує зручні механізми взаємодії в вигляді відповідей та лайків, а також ефективний пошук, що спрощує навігацію в інформаційному просторі. Водночас до недоліків платформи можна віднести значне інформаційне перевантаження, складність модерації контенту, а також залежність від алгоритмів рекомендацій, які можуть впливати на видимість повідомлень та формувати інформаційні бульбашки.

Платформа Threads, яка була створена як альтернатива текстовим соціальним мережам, орієнтована на простішу та більш інтуїтивну взаємодію користувачів. До її переваг можна віднести мінімалістичний інтерфейс, тісну інтеграцію з іншими сервісами, зокрема Instagram, а також акцент на текстовому

контенті та обговореннях. Це дозволяє користувачам швидко створювати та поширювати публікації без надмірного перевантаження функціональністю. Однак платформа має і певні обмеження, зокрема відносно обмежені функціональні можливості, а також залежність від екосистеми інших сервісів.

Instagram є прикладом соціальної мережі, орієнтованої на візуальний контент. Його основною перевагою є зручність публікації фотографій та відео, наявність інтерактивних форматів, таких як сторіс та прямі трансляції, а також ефективні алгоритми рекомендацій, які забезпечують високу залученість користувачів. Крім того, платформа активно інтегрує елементи комерції, що дозволяє використовувати її не лише для комунікації, а й для бізнесу. Разом з тим, до недоліків можна віднести залежність видимості контенту від алгоритмів, що ускладнює прогнозування охоплення, а також зміщення фокусу від текстового спілкування до візуального контенту, що обмежує можливості для користування.

Facebook є однією з найбільш універсальних соціальних платформ, яка поєднує в собі різноманітні функції, включаючи публікацію постів, створення груп і спільнот, обмін повідомленнями та організацію подій. Його перевагами є широкий функціонал, велика аудиторія та можливість створення різноманітних форматів взаємодії між користувачами. Водночас, складність інтерфейсу, значна кількість рекламного контенту та велика кількість функцій, що може негативно впливати на користувацький досвід. Крім того, як і в інших великих соціальних мережах, існує проблема перевантаження інформацією та необхідності ефективної фільтрації контенту.

Окрім розглянутих платформ, варто також звернути увагу на особливості реалізації стрічки постів та механізмів взаємодії з контентом, які є ключовими елементами сучасних соціальних мереж. У більшості систем стрічка формується або за допомогою складних алгоритмів ранжування, або за хронологічним принципом. Алгоритмічний підхід дозволяє персоналізувати контент, однак ускладнює передбачуваність його відображення та знижує прозорість для користувача. Натомість хронологічна стрічка забезпечує простоту реалізації,

зрозумілість логіки відображення та контроль над інформаційним потоком. Крім того, значну роль відіграють механізми взаємодії, такі як лайки, відповіді які формують активність користувачів та впливають на поширення контенту в межах платформи.

Аналіз наведених платформ показує, що, незважаючи на спільність базових функцій, кожна соціальна мережа має власну спеціалізацію та особливості реалізації користувацької взаємодії. Це свідчить про відсутність універсального рішення, яке б однаково ефективно задовольняло всі потреби користувачів. Сучасні соціальні мережі часто орієнтовані або на конкретний тип контенту, або на певний формат взаємодії, що обмежує їх гнучкість.

Додатково варто зазначити, що більшість сучасних соціальних платформ функціонують у межах складних екосистем, що включають інтеграцію з іншими сервісами, рекламними інструментами та аналітичними системами. Це підвищує їх функціональність, однак ускладнює структуру та знижує прозорість роботи системи для кінцевого користувача у результаті чого часто не має повного контролю над відображенням контенту, що може впливати на якість взаємодії та рівень довіри до платформи. Крім того, спільними недоліками більшості існуючих платформ є складність архітектури, залежність від алгоритмів персоналізації, ризики інформаційного перевантаження. Це створює передумови для пошуку нових підходів до розробки соціальних вебзастосунків, які б поєднували простоту використання, ефективність взаємодії та контрольованість контенту.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

У відповідності до поставленої мети кваліфікаційної роботи бакалавра, а також з урахуванням аналізу сучасного стану розробки соціальних вебзастосунків, важливо визначити перелік основних завдань, вирішення яких забезпечить досягнення поставленої мети. Дані завдання охоплюють як

теоретичні аспекти дослідження предметної області, так і практичну реалізацію вебзастосунку соціальної мережі.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- провести аналіз предметної області та існуючих соціальних мереж з метою визначення їх функціональних особливостей, переваг та недоліків, а також виявлення напрямів для покращення користувацького досвіду;
- дослідити сучасні підходи до розробки вебзастосунків соціальних мереж, зокрема клієнт-серверну архітектуру, принципи побудови інтерактивних інтерфейсів та організації обробки запитів;
- спроектувати архітектуру вебзастосунку, визначити основні компоненти системи, їх функціональне призначення та взаємодію між клієнтською і серверною частинами;
- розробити структуру бази даних для зберігання інформації про користувачів, пости та взаємодії між ними, забезпечивши цілісність і узгодженість даних;
- реалізувати серверну частину застосунку, включаючи обробку HTTP-запитів, організацію бізнес-логіки та взаємодію з базою даних;
- реалізувати клієнтську частину вебзастосунку із використанням компонентного підходу, забезпечивши зручний, інтуїтивно зрозумілий та адаптивний інтерфейс користувача;
- забезпечити реалізацію механізмів автентифікації та авторизації користувачів, а також базових засобів захисту даних;
- провести тестування функціоналу системи, виявити та усунути можливі недоліки, а також оцінити її працездатність у типових сценаріях використання.

РОЗДІЛ 2

СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

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

Для подальшої розробки програмного забезпечення соціальної мережі необхідним етапом є визначення підходів до моделювання системи та формування її вимог. Щоб забезпечити якісний та надійний підхід до визначення аспектів подальшої розробки доцільно використовувати сучасні методи проєктування, що дозволяють описати структуру програмного продукту, його функціональні можливості та взаємодію компонентів на логічному рівні.

З метою моделювання програмної системи обрано підхід, заснований на використанні клієнт-серверної прототипу, який передбачає розподіл функціональності між клієнтською та серверною частинами. Такий підхід є доцільним, оскільки він забезпечує масштабованість системи, можливість незалежного вдосконалення її компонентів, а також ефективну обробку інформації. Клієнтська частина відповідає за взаємодію з користувачем та відображення інформації, тоді як серверна частина забезпечує обробку запитів, бізнес-логіку та взаємодію з базою даних.

Формування та аналіз вимог до програмного забезпечення визначено на основі аналізу предметної області, дослідження існуючих соціальних платформ та потреб потенційних користувачів системи. Основними методами аналізу було порівняння аналогів, функціональна декомпозиція системи та узагальнення типових сценаріїв взаємодії користувача із соціальними мережами. Це допомагає сформувати цілісне уявлення про необхідний функціонал системи та її основні характеристики.

Для формалізації вимог та побудови логічної структури системи застосовано підхід, який передбачає виділення основних сутностей системи, визначення зв'язків між ними та опис їх поведінки. Такий метод дозволяє визначити ключові компоненти системи на етапі проєктування, сформувати їх

взаємодію та забезпечити узгодженість між окремими частинами програмного забезпечення.

У результаті проведеного аналізу було сформовано специфікацію вимог до програмного забезпечення, яка визначає функціональні можливості системи, її поведінку та обмеження. Специфікація вимог є основою для подальшого проєктування та реалізації програмного продукту, оскільки вона забезпечує чітке розуміння того, які задачі повинна вирішувати система та яким чином вона повинна взаємодіяти з користувачами.

З функціональної точки зору розроблювана система повинна забезпечувати повний цикл роботи з інформацією, який включає створення, зміну та відображення даних. Користувачу потрібно мати можливість вводити інформацію у систему у вигляді постів, відповідей та персональних даних профілю, після чого ці дані зберігаються у базі даних і можуть бути змінені або доповнені у подальшому. Важливою складовою є можливість пошуку інформації за запитом користувача, що дозволить отримувати релевантні результати та швидко знаходити необхідний контент. Система також повинна забезпечувати відображення інформації у вигляді стрічки постів, що формується у відповідь на запити клієнтської частини, а також підтримувати взаємодію користувачів із контентом.

Особлива увага приділяється забезпеченню безпеки та цілісності даних у системі, що необхідно досягти шляхом розмежування прав доступу між різними типами користувачів, щоб обмежити виконання певних дій лише для авторизованих осіб. Крім того, система повинна передбачати перевірку введених даних перед їх обробкою, що дозволить уникнути некоректної або небезпечної інформації. Збереження даних у базі варто організувати таким чином, щоб забезпечити їх узгодженість та цілісність навіть при одночасній роботі великої кількості користувачів. Такий підхід дозволить гарантувати надійність роботи системи та захист персональної інформації.

Для формалізованого подання функціональних можливостей розроблюваного вебзастосунку доцільно використати UML-діаграму варіантів

використання. Така діаграма дозволяє відобразити основних учасників системи та визначити набір дій, які вони можуть виконувати під час взаємодії з програмним забезпеченням.

У межах розроблюваної соціальної мережі основними акторами є гість та авторизований користувач. Гість має обмежений доступ до функціоналу системи, оскільки може лише переглядати відкритий контент, профілі користувачів і виконувати пошук. Авторизований користувач отримує розширені можливості, зокрема створення постів, взаємодію з контентом, редагування власного профілю та використання системи підписок. На рисунку 2.1 наведено UML-діаграму варіантів використання, яка відображає основні сценарії взаємодії користувачів із системою.



Рисунок 2.1 – Діаграма варіантів використання вебзастосунку соціальної мережі

Наведена діаграма демонструє розмежування прав доступу між неавторизованим та авторизованим користувачем. Такий підхід є доцільним, оскільки частина функцій системи повинна бути доступною для загального перегляду, тоді як дії, пов'язані зі створенням контенту, взаємодією з постами та зміною персональних даних, мають виконуватися лише після автентифікації.

Для детального аналізу взаємодії між компонентами системи варто використати UML-діаграму послідовності, яка дозволяє відобразити порядок обміну повідомленнями між елементами програмного забезпечення у процесі виконання конкретних сценаріїв. На відміну від діаграми варіантів використання, яка відображає загальні можливості системи, діаграма послідовності дозволяє детально проаналізувати логіку роботи клієнт-серверної архітектури, а також взаємодію між користувачем, клієнтською частиною, сервером та базою даних.

На рисунку 2.2 можна побачити діаграму послідовності, яка використовується для моделювання основних сценаріїв взаємодії користувача із системою, а саме для створення посту та взаємодії з контентом шляхом встановлення лайків та написання коментарів. Діаграма демонструє процес взаємодії між користувачем, клієнтською частиною, серверною частиною та базою даних у межах виконання основних функціональних сценаріїв системи.

Деталізація взаємодії між компонентами системи на рівні послідовності дозволяє не лише зрозуміти загальний порядок виконання операцій, але й виявити потенційні вузькі місця в обробці запитів. Зокрема, така модель дає змогу оцінити навантаження на серверну частину, кількість звернень до бази даних та особливості передачі даних між клієнтом і сервером. Це є важливим етапом проєктування, оскільки дозволяє оптимізувати логіку обробки запитів, зменшити кількість зайвих операцій та підвищити загальну ефективність роботи вебзастосунку. Крім того, використання діаграми послідовності спрощує подальшу реалізацію функціоналу, оскільки кожен крок взаємодії може бути безпосередньо відображений у вигляді відповідних викликів API та обробників на серверній стороні.

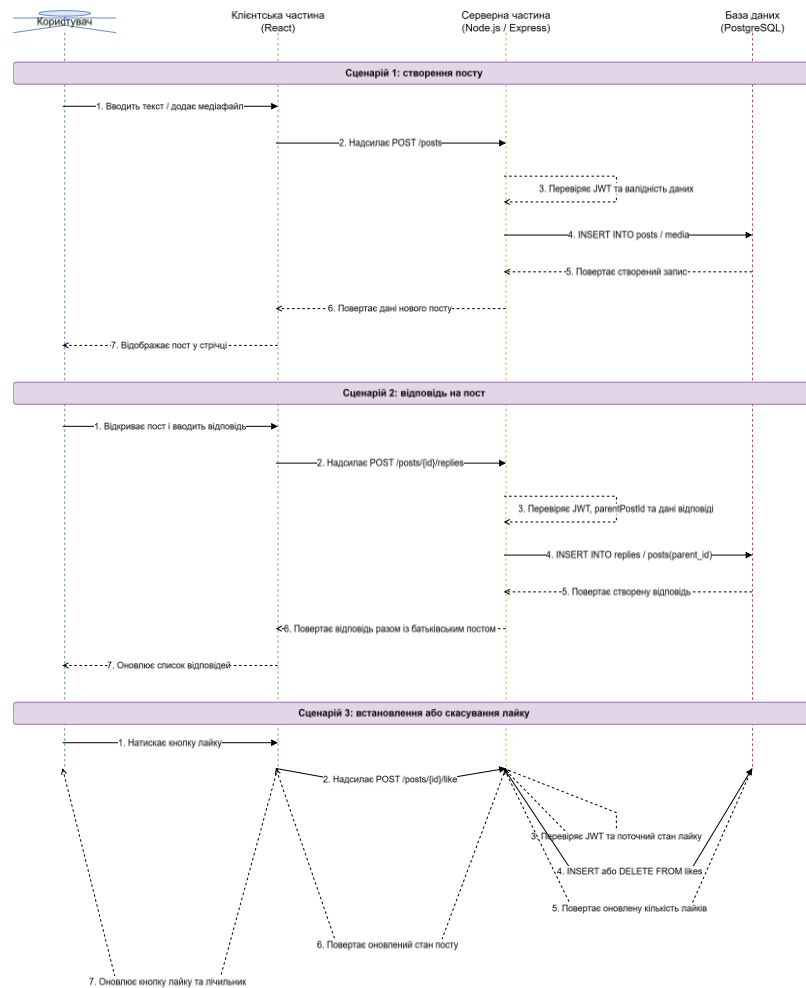


Рисунок 2.2 – Діаграма послідовності взаємодії користувача із системою

У сценарії створення посту користувач ініціює відповідну дію через інтерфейс клієнтського застосунку, після чого формується запит до серверної частини. Сервер обробляє отримані дані, виконує операцію збереження інформації у базі даних та повертає результат виконання запиту клієнту. Клієнтська частина, у свою чергу, оновлює інтерфейс та відображає новостворений пост у стрічці.

Аналогічним чином реалізується сценарій залишення відповіді, де взаємодія включає передачу даних про батьківський пост та збереження відповідної інформації у базі даних. Це дозволяє забезпечити зв'язок між постами та їх відповідями, формуючи ієрархічну структуру контенту.

У сценарії взаємодії з контентом через додавання лайку користувач ініціює дію, яка передається на сервер для обробки. Сервер відслідковує поточний стан

та відповідно оновлює дані у базі та клієнтська частина відображає актуальні зміни.

Результати моделювання показують, що структура системи орієнтована на два основні сценарії використання: пасивне споживання контенту та активну взаємодію із соціальною платформою. Перший сценарій охоплює перегляд постів, профілів та пошук інформації. Другий передбачає створення публікацій, залишення відповідей, використання лайків, підписку на інших користувачів та редагування власного профілю.

Обране рішення дозволяє чітко визначити межі доступу до функціоналу системи та забезпечити логічну організацію користувацької взаємодії. Крім того, така модель спрощує подальше проектування клієнтської та серверної частин вебзастосунку, оскільки кожен варіант використання може бути пов'язаний з окремими інтерфейсними елементами, API-запитами та операціями з базою даних.

З точки зору організації взаємодії користувача із системою важливим є забезпечення зручного та інтуїтивно зрозумілого інтерфейсу. Структура вебзастосунку передбачає логічне розділення функціональних елементів, що дозволяє користувачу швидко орієнтуватися у системі та виконувати необхідні дії без зайвих ускладнень. Інформаційні потоки в системі організовані таким чином, що взаємодія між клієнтською та серверною частинами відбувається через обмін запитам та відповідями, що забезпечує оперативне оновлення даних та відображення актуального стану системи. Такий підхід дозволяє ефективно обробляти дії користувача та підтримувати безперервність роботи вебзастосунку.

Результати моделювання свідчать про те, що обрана клієнт-серверна архітектура відповідає поставленим очікуванням та забезпечує чіткий розподіл відповідальності між компонентами системи, а також дозволяє ефективно організувати обробку запитів користувачів. Такий підхід сприяє масштабованості системи, спрощує підтримку та забезпечує можливість подальшого розширення функціоналу.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Для реалізації клієнтської частини вебзастосунку можуть бути використані різні сучасні JavaScript-фреймворки, серед яких найбільш поширеними є React та Angular. Angular пропонує комплексний підхід до розробки, включаючи вбудовані інструменти для маршрутизації, роботи з формами та управління станом, однак характеризується більш складною структурою та високим порогом входження. React, у свою чергу, базується на компонентному підході, що дозволяє гнучко будувати інтерфейс користувача, повторно використовувати елементи та ефективно оновлювати стан застосунку [4].

У даній роботі для розробки клієнтської частини було обрано бібліотеку React у поєднанні з мовою TypeScript. Такий вибір обумовлений високою продуктивністю, гнучкістю архітектури та можливістю створення динамічних інтерфейсів. Використання TypeScript дозволяє підвищити надійність коду та зменшити ймовірність помилок під час розробки за рахунок статичної типізації, що особливо важливо при розробці складних вебзастосунків з великою кількістю взаємодій [5].

Для реалізації серверної частини вебзастосунку можуть бути використані різні технології, серед яких поширеними є Node.js та фреймворки на інших мовах програмування, зокрема Django для Python або Spring для Java. Django пропонує високий рівень абстракції та велику кількість вбудованих засобів для розробки вебзастосунків, що дозволяє швидко створювати функціональні системи, однак він менш гнучкий у побудові нестандартної логіки взаємодії в реальному часі. Spring, у свою чергу, забезпечує високу продуктивність та масштабованість, проте характеризується складністю налаштування та більшою кількістю допоміжних інструментів, що ускладнює процес розробки.

У даній роботі для реалізації серверної частини було обрано платформу Node.js із використанням фреймворку Express. Такий вибір зумовлений можливістю використання єдиної мови програмування як на клієнтській, так і на

серверній стороні, що спрощує розробку та підтримку системи. Крім того, Node.js ефективно працює з великою кількістю одночасних запитів завдяки асинхронній моделі обробки, що є важливим для соціальних вебзастосунків із високим рівнем взаємодії користувачів [6]. Використання Express дозволяє гнучко організувати структуру серверної частини та реалізувати необхідні API для взаємодії з клієнтським застосунком.

Реалізація функціональних можливостей системи здійснюється із використанням сучасних підходів до розробки вебзастосунків. Зокрема, клієнтська частина побудована на основі компонентного підходу, що дозволяє розділити інтерфейс на незалежні елементи та забезпечити повторне використання логіки. Взаємодія з серверною частиною реалізується через REST API, що дозволяє організувати чіткий обмін даними між клієнтом і сервером. Серверна частина, у свою чергу, використовує асинхронну обробку запитів, що забезпечує ефективну роботу системи при великій кількості одночасних користувачів.

Для організації зберігання даних у вебзастосунках можуть використовуватися як реляційні, так і нереляційні системи керування базами даних. До нереляційних рішень належить, зокрема, MongoDB, яка дозволяє зберігати дані у вигляді документів та забезпечує гнучкість структури. Такий підхід є зручним при роботі з неструктурованими або динамічними даними, однак може ускладнювати забезпечення цілісності даних та виконання складних запитів.

Реляційні системи керування базами даних, зокрема PostgreSQL, забезпечують чітко визначену структуру даних, підтримку зв'язків між таблицями та можливість виконання складних запитів. Це дозволяє ефективно організувати зберігання взаємопов'язаних даних, що є характерним для соціальних мереж, де існують зв'язки між користувачами, постами та взаємодіями.

У даній роботі для зберігання даних було обрано систему PostgreSQL. Такий вибір спричинений необхідністю забезпечення цілісності даних,

підтримки зв'язків між сутностями та можливості виконання складних запитів. Крім того, використання реляційної бази даних дозволяє більш чітко структурувати інформацію та забезпечити стабільну роботу системи при зростанні обсягів даних [7].

Після вибору основних інструментів та технологій важливим етапом є їх практичне застосування для реалізації функціональних можливостей системи. Обрані засоби розробки дозволяють ефективно вирішувати задачі обробки даних, організації взаємодії користувачів та забезпечення стабільної роботи вебзастосунку.

У процесі розробки системи буде реалізовано низку алгоритмів, що забезпечить виконання основних функціональних сценаріїв. Зокрема, алгоритм створення посту передбачає отримання даних від користувача, їх передачу на серверну частину, перевірку коректності введеної інформації та подальше збереження у базі даних з наступним поверненням результату клієнту. Аналогічно, алгоритм залишення відповіді включає обробку зв'язку з батьківським постом та збереження відповідної структури даних, що дозволяє формувати ієрархію постів.

Окремо реалізовано алгоритм взаємодії з контентом, зокрема встановлення лайків, який передбачає ідентифікацію користувача, перевірку наявності попередньої взаємодії та оновлення відповідних даних у базі. Такий підхід дозволяє забезпечити коректність відображення кількості взаємодій та уникнути дублювання дій користувача.

Важливим елементом системи є також алгоритм формування стрічки постів, який базується на отриманні даних із бази у хронологічному порядку з урахуванням поступового підвантаження інформації. Це дозволяє оптимізувати обсяг переданих даних та забезпечити безперервну взаємодію користувача із системою.

Для відображення логічної структури даних розроблюваної системи доцільно використати UML-діаграму класів. Вона дозволяє визначити основні сутності вебзастосунку, їх атрибути та зв'язки між ними. Такий підхід є

важливим на етапі проєктування, оскільки дає змогу узгодити структуру програмного забезпечення з функціональними вимогами системи.

У межах розроблюваної соціальної мережі основними сутностями є користувач, пост, лайк, підписка та медіафайл. Кожна з цих сутностей відповідає окремому елементу предметної області та бере участь у реалізації основних сценаріїв роботи системи. На рисунку 2.3 наведено UML-діаграму класів, яка демонструє структуру основних сутностей системи та зв'язки між ними.

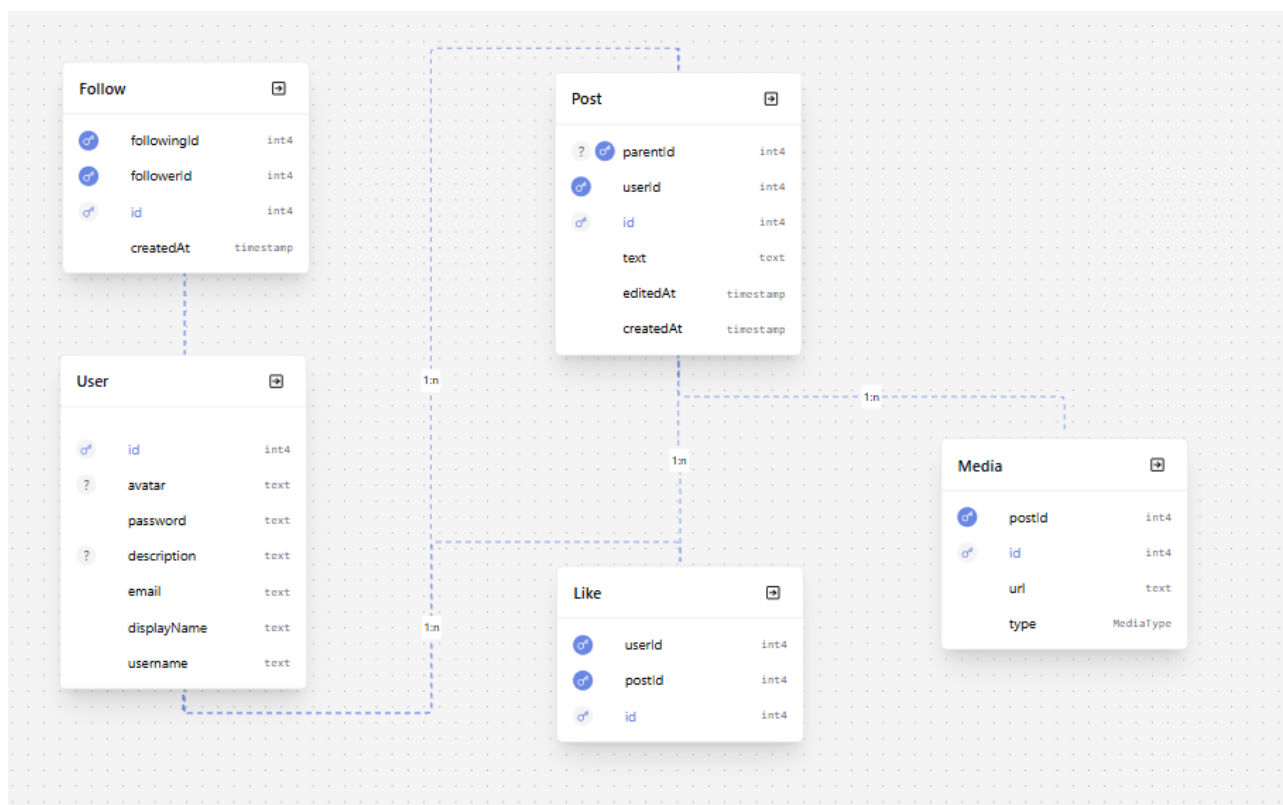


Рисунок 2.3 – Діаграма класів вебзастосунку соціальної мережі

Наведена діаграма класів демонструє логічну структуру основних об'єктів системи. Центальною сутністю є клас **User**, який описує користувача соціальної мережі та містить основні дані профілю, зокрема ім'я користувача, відображуване ім'я, електронну пошту, опис, аватар. Користувач може створювати пости, ставити лайки та брати участь у системі підписок.

Клас **Post** відповідає за збереження публікацій користувачів. Він містить текст посту, дату створення, дату редагування, посилання на автора, а також

зв'язки з відповідями, лайками та медіафайлами. Наявність поля батьківського посту дозволяє реалізувати механізм відповідей, коли один пост може бути пов'язаний з іншим як реакція або коментар. Це забезпечує можливість побудови гілок обговорення в межах системи.

Клас Like використовується для фіксації взаємодії користувача з постом. Він пов'язує конкретного користувача з конкретною публікацією, що дозволяє визначати, які пости були вподобані користувачем. Обмеження унікальності для пари користувач-пост запобігає повторному встановленню лайку одним і тим самим користувачем на одну й ту саму публікацію.

Клас Follow описує зв'язок підписки між користувачами. Він містить інформацію про користувача, який підписується, та користувача, на якого оформлено підписку. Така структура дозволяє реалізувати механізм підписників і підписок, а також формувати стрічку контенту на основі користувачів, за якими стежить авторизований користувач.

Клас Media відповідає за збереження інформації про медіафайли, прикріплені до постів. Він містить посилання на відповідний пост, тип медіафайлу та адресу його зберігання. Для визначення типу медіафайлу використовується перелік MediaType, який дозволяє розмежовувати зображення та відео.

Таким чином, побудована діаграма класів відображає основні сутності системи та їх взаємозв'язки. Запропонована структура є доцільною для реалізації соціальної мережі, оскільки вона охоплює ключові функції застосунку, а саме роботу з користувачами, створення контенту, взаємодію з постами, систему підписок та підтримку мультимедійних матеріалів.

Для більш загального представлення архітектури системи доцільно також розглянути її на рівні компонентів. У цьому випадку система може бути представлена як сукупність клієнтської частини, серверного застосунку та бази даних, які взаємодіють між собою через визначені інтерфейси. Такий підхід дозволяє наочно відобразити розподіл відповідальності між основними частинами системи та спрощує розуміння її внутрішньої організації.

На рисунку 2.4 наведено діаграму компонентів, яка демонструє взаємодію основних складових системи.

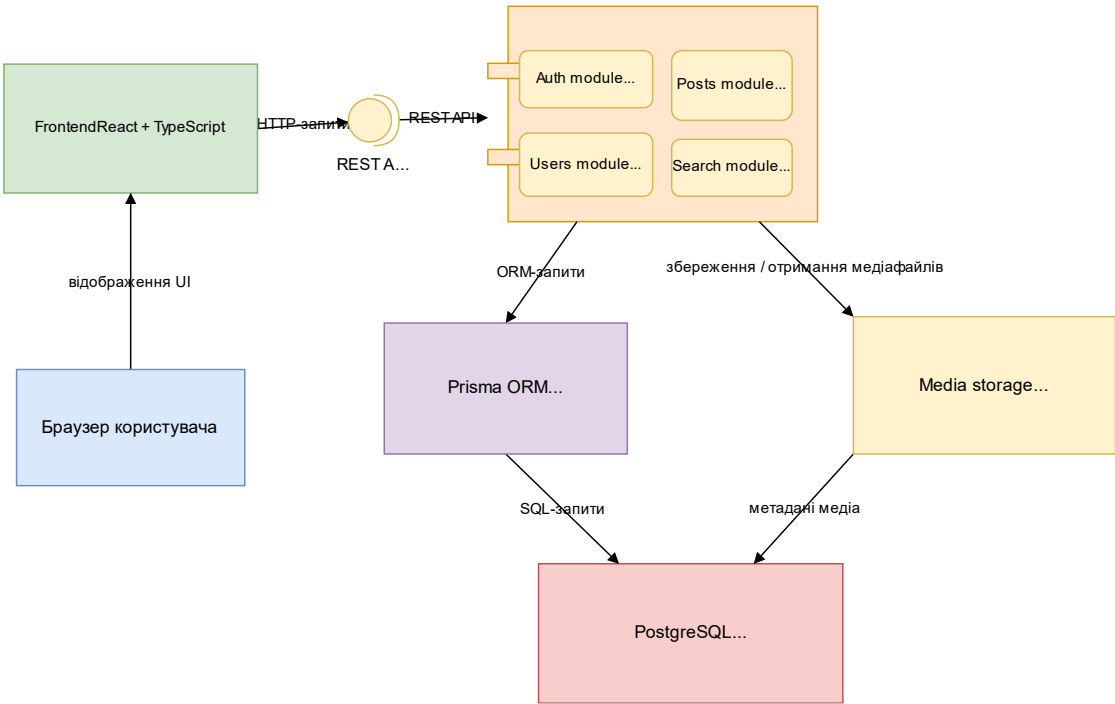


Рисунок 2.4 – Діаграма компонентів вебзастосунку соціальної мережі

РОЗДІЛ 3

РОЗРОБКА ВЕБЗАСТОСУНКУ СОЦІАЛЬНОЇ МЕРЕЖІ З ВИКОРИСТАННЯМ NODE.JS ТА REACT

3.1 Практична реалізація об'єкта проєктування

У межах даної роботи практична реалізація вебзастосунку соціальної мережі здійснювалась із використанням сучасних технологій розробки клієнт-серверних систем. У процесі створення програмного забезпечення було реалізовано основні функціональні можливості, описані у попередніх розділах, за допомогою використання мови програмування TypeScript як на серверній, так і на клієнтській частині.

Архітектура застосунку побудована за принципом розділення відповідальності між компонентами системи, що дозволяє забезпечити гнучкість та масштабованість. Основна логіка обробки запитів реалізована на серверній частині з використанням платформи Node.js та фреймворку Express [8], тоді як клієнтська частина відповідає за взаємодію з користувачем та відображення даних. Підхід з розділеною відповідальністю дозволив реалізувати серверну логіку у вигляді окремого застосунку, який відповідає за обробку HTTP-запитів, взаємодію з базою даних, перевірку вхідних даних, роботу з файлами та формування відповідей для клієнтської частини.

Серверна частина застосунку організована як окремий модуль проєкту у ньому налаштовується застосунок на базі Express, підключаються middleware та маршрути, що забезпечує керування обробкою запитів. Для структуризації коду його поділено на логічні частини: routes, controllers, services, middleware, utils та types. Такий підхід дозволяє розділити рівні відповідальності та спрощує підтримку і масштабування застосунку.

Директорія routes визначає HTTP-ендпоінти та відповідає за направлення запитів до відповідних обробників, що відповідає підходам побудови серверних застосунків на базі Express [9]. Каталог controllers реалізує обробку запитів і формування відповідей, виступаючи проміжним рівнем між маршрутизацією та

бізнес-логікою [10]. Основна логіка роботи з даними винесена у каталог `services`, що дозволяє зменшити навантаження на контролери та забезпечити повторне використання коду.

Папка `middleware` містить обробники, які виконують додаткові дії під час обробки запиту, зокрема перевірку токена, валідацію даних та обробку помилок. Це дозволяє реалізувати послідовний ланцюг обробки запитів і забезпечити модульність логіки [11]. Допоміжні функції винесені у каталог `utils`, що зменшує дублювання коду та спрощує його підтримку. Окремо використовується каталог `types`, який містить типи TypeScript і забезпечує узгодженість структури даних між модулями [12].

Узагальнено, обробка запиту відбувається за типовою схемою де запит проходить через маршрутизатор, за потреби викликається `middleware`, передається до контролера, який використовує відповідний сервіс для роботи з даними і результат повертається клієнту. Така архітектура є зрозумілою, масштабованою та відповідає сучасним підходам до розробки вебзастосунків. Структуру каталогів серверної частини проєкту наведено на рисунку 3.1.

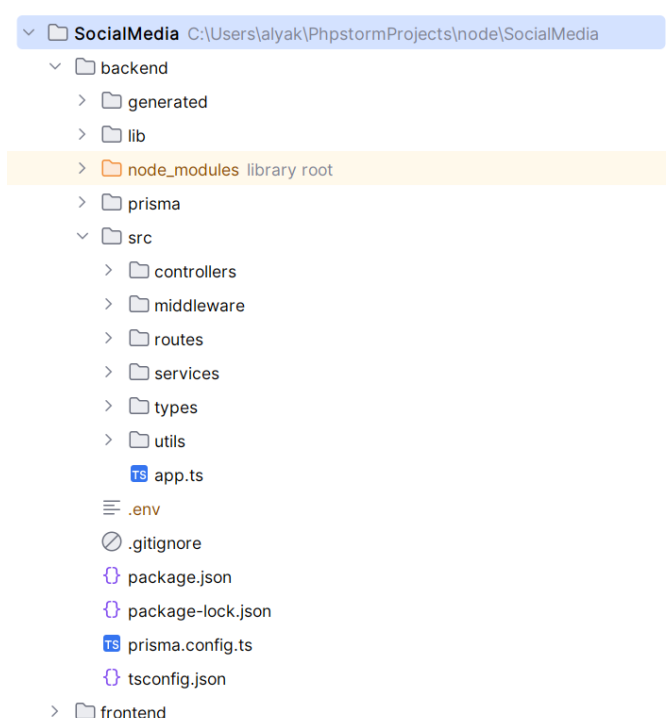


Рисунок 3.1 – Структура каталогів серверної частини вебзастосунку

Для кращого розуміння практичної реалізації описаної архітектури доцільно розглянути взаємодію її основних компонентів на прикладі конкретного функціонального сценарію. Це дозволяє перейти від загального опису структури до аналізу коду та простежити, яким чином здійснюється обробка запиту на різних рівнях застосунку. У якості прикладу обрано операцію редагування посту, оскільки вона включає обробку вхідних даних, перевірку прав доступу, роботу з файлами та взаємодію із сервісним рівнем. Відповідний фрагмент контролера наведено в лістингу 3.1.

Лістинг 3.1 – Фрагмент контролера редагування посту

```
export async function editPost(req: Request, res: Response) {
  const { text, keepMediaIds } = req.body;
  const parsedKeepIds: number[] = keepMediaIds ? JSON.parse(keepMediaIds) : [];
  const postId = parseId(req.params.postId, "Post id");
  const userId = getUserId(req);

  const postFromDb = await postServices.getPostById(postId, userId);
  if (!postFromDb) throw new AppError(404, "Post with this id not found");
  if (postFromDb.userId !== userId) throw new AppError(403, "You not author of this post");
  const files = (req.files as Express.Multer.File[]) ?? [];
  if (parsedKeepIds.length + files.length > 2) throw new AppError(400, "Maximum 2 media per post");

  const newMedia = await uploadFiles(files, "SocialMediaFiles");
  const post = await postServices.updatePost(postId, text, parsedKeepIds, newMedia);
  res.status(200).json({ post });
}
```

кінець лістингу 3.1

На початковому етапі коду відбувається отримання вхідних даних із тіла запиту, зокрема тексту посту та списку ідентифікаторів медіа, які необхідно зберегти. Оскільки дані можуть передаватися у вигляді рядка, виконується їх додаткова обробка та приведення до відповідного формату. Окрему увагу приділено обробці медіа.

Контролер отримує список файлів, переданих у запиті, та перевіряє обмеження на їх кількість. У випадку перевищення допустимого значення формується помилка, що запобігає некоректному використанню функціоналу

системи. Після проходження всіх перевірок виконується завантаження нових медіа та виклик відповідного методу сервісу для оновлення посту та формування HTTP-відповіді, яка містить оновлений об'єкт посту у форматі JSON для подальшого оброблення його на клієнтській частині щоб дані оновлювались самостійно без потреби перезавантаження сторінки.

Таким чином, контролер реалізує повний цикл обробки запиту: від отримання та перевірки даних до взаємодії з бізнес-логікою та повернення результату клієнту. Подібна структура дозволяє забезпечити чітке розмежування відповідальності між різними рівнями застосунку та спрощує підтримку коду.

Для доповнення розглянутого прикладу доцільно проаналізувати спосіб організації виклику контролерів у межах маршрутизації застосунку. Це дозволяє простежити, яким чином визначається відповідність між HTTP-запитами та функціями їх опрацювання, а також як формується послідовність виконання додаткових обробників. Відповідний фрагмент маршрутизатора, що забезпечує роботу з постами, наведено в лістингу 3.2.

Лістинг 3.2 – Фрагмент маршрутизатора для роботи з постами

```
import express from "express";
import { validateFields } from "../middleware/validateFields";
import { verifyToken } from "../middleware/verifyToken";
import { optionalAuth } from "../middleware/optionalAuth";
import { postValidation } from "../services/postValidation";
import { addPost, editPost, deletePost, getPosts, getUserPost, getPostReplies, getPostById,
  getFollowingPosts, getSearchPosts } from "../controllers/postController";
import { upload } from "../utils/multer";
import { validateFiles } from "../middleware/validateFile";

const postRouter = express.Router();

postRouter.get("/following", verifyToken, getFollowingPosts);
postRouter.get("/search", verifyToken, getSearchPosts);
postRouter.delete("/:postId", verifyToken, deletePost);
postRouter.put("/:postId", upload.array("media", 2), validateFiles, verifyToken, postValidation,
  validateFields, editPost);
postRouter.post("/:postId/replies", upload.array("media", 2), validateFiles, verifyToken,
  postValidation, validateFields, addPost);
postRouter.post("/", upload.array("media", 2), validateFiles, verifyToken, postValidation,
  validateFields, addPost);
postRouter.get("/:postId/replies", optionalAuth, getPostReplies);
```

```
postRouter.get("/:postId", optionalAuth, getPostById);  
postRouter.get("/user/:userId", optionalAuth, getUserPost);  
postRouter.get("/", optionalAuth, getPosts);
```

```
export default postRouter;
```

кінець лістингу 3.2

Наведений фрагмент демонструє налаштування HTTP-маршрутів для виконання основних операцій із постами, зокрема створення, редагування, видалення та отримання даних. Кожен маршрут визначає шлях запиту, HTTP-метод та послідовність обробників, які будуть виконані під час обробки запиту. Особливістю реалізації є використання `middleware`, які підключаються до маршрутів перед викликом контролера. Наприклад, для операцій, що потребують авторизації, використовується перевірка токена користувача, а для обробки файлів застосовується спеціалізований `middleware`, який відповідає за завантаження медіаконтенту також тут використовуються механізми валідації вхідних даних, що дозволяє забезпечити їх коректність ще до передачі в контролер.

Послідовність виклику `middleware` формує чіткий ланцюг обробки запиту, у якому кожен етап виконує окрему функцію. Такий підхід дозволяє розділити логіку на незалежні компоненти та повторно використовувати їх у різних маршрутах. Загалом використання маршрутизатора забезпечує централізоване управління доступними ендпоінтами системи, спрощує організацію коду та дозволяє гнучко налаштовувати обробку запитів залежно від їх типу та призначення.

Забезпечення коректної обробки помилок, які можуть виникати під час виконання запитів є важливим аспектом у процесі розробки серверної частини вебзастосунку і наявність централізованого механізму для цього дозволяє уникнути дублювання коду, спростити логіку контролерів та забезпечити єдиний формат відповідей для клієнтської частини (ліст. 3.3). У застосунку реалізовано глобальний обробник помилок, який підключається як `middleware` у кінці обробки запитів. Такий підхід відповідає рекомендаціям фреймворку Express та

дозволяє перехоплювати всі виняткові ситуації, що виникають під час виконання контролерів або інших middleware.

Лістинг 3.3 – Глобальний обробник помилок

```
import type { Request, Response, NextFunction } from "express";
import { AppError } from "../utils/AppError";

export function errorHandler(err: unknown, req: Request, res: Response, next: NextFunction) {
  console.log("Error: ", err);
  let status = 500;
  let errors: string[] = ["Something went wrong"];

  if (err instanceof AppError) {
    status = err.statusCode;
    errors = err.errorsText;
  }

  else if (err instanceof Error) {
    errors = [err.message];
  }

  res.status(status).json({ errors });
}
```

кінець лістингу 3.3

Наведений фрагмент коду реалізує універсальний механізм обробки помилок. Функція приймає об'єкт помилки, параметри запиту та відповіді, після чого визначає тип помилки та формує HTTP відповідь де на початковому етапі встановлюється загальний статус 500, який відповідає внутрішній помилці сервера, це робиться щоб запобігти випадкам відсутності коду помилки і кращого розуміння що могло піти не так. Далі виконується перевірка типу помилки. Якщо помилка є екземпляром користувацького класу `AppError`, з неї отримується код статусу та текст повідомлення. Це дозволяє явно керувати помилками, що виникають у бізнес-логіці, та повертати клієнту інформативні повідомлення.

Після реалізації серверної частини важливим етапом стала розробка клієнтської частини вебзастосунку, яка відповідає за взаємодію користувача із системою. Саме клієнтська частина забезпечує відображення постів, профілів

користувачів, форм створення контенту, сторінок авторизації, пошуку та інших елементів інтерфейсу. Вона реалізована з використанням React та TypeScript. Основою побудови інтерфейсу є компонентний підхід, за якого окремі частини сторінки виділяються в незалежні компоненти. Це дозволяє повторно використовувати однакові елементи інтерфейсу, спрощує підтримку коду та робить структуру застосунку більш зрозумілою.

У межах frontend частини доцільно виділити кілька основних груп елементів такі як сторінки, компоненти, контексти, типи та допоміжні модулі для взаємодії з API. Сторінки відповідають за відображення основних розділів вебзастосунку у вигляді головної сторінки, профілю користувача, сторінки пошуку, сторінки окремого посту, сторінки входу та реєстрації. Компоненти використовуються для побудови повторюваних елементів інтерфейсу, наприклад постів, елементів користувачів, кнопок, форм або навігаційних панелей.

Для керування спільними даними між різними частинами застосунку використовуються контексти React. Вони дозволяють зберігати інформацію, яка потрібна багатьом компонентам, без необхідності передавати її через велику кількість проміжних елементів. У цьому застосунку контексти використовуються, зокрема, для зберігання інформації про поточного користувача та списку постів.

Для кращого розуміння роботи варто розглянути окремі фрагменти реалізації клієнтської частини, які демонструють використання компонентів, контекстів, типізації та взаємодії з серверним API. Для організації взаємодії із серверною частиною у застосунку реалізовано окремий модуль API-клієнта, який централізує виконання HTTP-запитів та забезпечує єдиний підхід до обміну даними між frontend та backend частинами системи.

Такий підхід дозволяє уникнути дублювання логіки запитів у різних компонентах, спростити підтримку коду та забезпечити узгодженість у способах обробки відповідей і помилок. Демонстрацію реалізації API-клієнта можна побачити в лістингу 3.4.

Лістинг 3.4 – Реалізація API-клієнта для виконання HTTP-запитів

```

export default async function Client( link: string,
  method: "GET" | "POST" | "PUT" | "DELETE", body?: string | FormData, signal?:AbortSignal):
  Promise<any> {
  try {
    const token = localStorage.getItem("token");
    const options: RequestInit = {method};
    const headers: Record<string, string> = {};
    if(token) headers["Authorization"] = `Bearer ${token}`;
    if (body && !(body instanceof FormData)) headers["Content-Type"] = "application/json";
    options.headers = headers;
    if(body && method !== "GET") options.body = body;
    if(signal)options.signal = signal;
    const response = await fetch(`${baseUrl}${link}`, options);
    return response.json();
  } catch (e: any){
    let error: string = "Something went wrong!";
    if (e.name === "AbortError") {
      return undefined;
    } else if(typeof e === "string"){
      error = e;
    } else if (e instanceof TypeError) {
      error = e.message;
    }
    return {errors: [error]};
  }
}

```

кінець лістингу 3.4

Наведена функція приймає шлях, метод, тіло та додаткові параметри запиту, після чого формує його за допомогою API fetch. На початковому етапі отримується токен авторизації з локального сховища браузера якщо токен присутній, він додається до заголовків запиту у форматі Authorization, що дозволяє серверній частині ідентифікувати користувача. Такий підхід забезпечує підтримку авторизованих запитів без необхідності передавати токен вручну в кожному компоненті. Вона також містить логіку визначення базової адреси серверної частини залежно від режиму роботи застосунку. Це дозволяє використовувати різні сервери для розробки та середовища без зміни основного коду.

Далі формується об'єкт параметрів запиту, у якому визначається HTTP-метод, заголовки та, за необхідності, тіло запиту. У випадку передачі даних у

форматі JSON додатково встановлюється відповідний заголовок Content-Type. При цьому передбачена підтримка передачі файлів у вигляді об'єкта FormData, що дозволяє використовувати один і той самий клієнт для різних типів запитів.

Після виконання запиту здійснюється обробка відповіді сервера у форматі JSON. У випадку виникнення помилки передбачено централізований механізм її обробки, який дозволяє повернути узгоджений формат повідомлення. Окремо враховано ситуацію скасування запиту, що не вважається помилкою та обробляється окремо. Таким чином, використання централізованого API-клієнта дозволяє забезпечити єдиний підхід до взаємодії з сервером, спростити реалізацію компонентів клієнтської частини та підвищити узгодженість обробки даних у всьому застосунку.

Під час реалізації важливо було виконати оптимізацію отримання даних для стрічки постів. Оскільки соціальна мережа передбачає постійне збільшення кількості контенту, отримання всіх записів одним запитом є неефективним і може створювати надмірне навантаження на серверну частину, базу даних та клієнтський інтерфейс. Для вирішення цієї проблеми у застосунку реалізовано поступове завантаження даних частинами.

На серверній частині для цього використовується курсорна пагінація, яка базується на ідентифікаторі останнього отриманого посту (ліст. 3.5). Під час першого запиту система повертає обмежену кількість записів, а під час наступних запитів передає cursorId, після чого сервер отримує пости з ідентифікатором, меншим за переданий. Це дозволяє послідовно завантажувати старіші публікації без повторного отримання вже відображених даних.

Лістинг 3.5 – Реалізація курсорної пагінації на серверній частині

```
getPosts: async(cursorId: number | null, userId: number | null): Promise<PostWithUser[]>=>
  prisma.post.findMany({
    where: {
      parentId: null,
      ...(cursorId ? {id: {lt: cursorId}} : {})
    },
    include: {
      media: {},
    }
  })
```

```

user:{
  select: {id:true, username: true, displayName:true, avatar:true}
},
likes: userId ? {
  where: { userId },
  select: { id: true }
} : false,
_count: { select: { replies: true, likes: true } }
},
orderBy: {
  id: "desc"
},
take: 20
}),

```

кінець лістингу 3.5

Наведений фрагмент демонструє отримання постів із бази даних де у запиті використовується умова `id: { lt: cursorId }`, яка дозволяє отримати записи, створені раніше від останнього завантаженого посту. Дані сортуються за спаданням ідентифікатора, а параметр `take: 20` обмежує кількість записів, що повертаються за один запит. Такий підхід зменшує обсяг даних, які передаються між сервером і клієнтом, та забезпечує стабільнішу роботу стрічки.

Крім самих постів, сервер одразу повертає пов'язані дані, необхідні для відображення стрічки: інформацію про автора, медіафайли, кількість лайків і відповідей. Якщо користувач авторизований, додатково виконується перевірка, чи вподобав він конкретний пост. Це дозволяє клієнтській частині одразу відобразити актуальний стан елементів інтерфейсу без виконання додаткових запитів.

На клієнтській частині поступове завантаження реалізовано за допомогою механізму нескінченного прокручування. Під час прокручування сторінки система визначає, чи наблизився користувач до нижньої частини документа. Якщо це відбулося і попередній запит уже завершено, формується новий запит із передаванням `cursorId` останнього елемента у списку. Важливо зазначити що під час оновлення списку виконується фільтрація вже наявних елементів за ідентифікатором. Це дозволяє уникнути дублювання постів у стрічці, якщо певні записи вже були завантажені раніше. Завдяки цьому інтерфейс залишається

стабільним, а користувач отримує безперервний досвід перегляду контенту без ручного переходу між сторінками.

Таким чином, поєднання курсорної пагінації на серверній частині та нескінченного прокручування на клієнтській частині дозволяє оптимізувати отримання даних, зменшити навантаження на сервер і базу даних, а також покращити зручність взаємодії користувача зі стрічкою постів. Це особливо важливо для соціальної мережі, де кількість контенту постійно збільшується, а швидкість відображення даних безпосередньо впливає на якість користувацького досвіду.

Для керування спільним станом клієнтської частини у вебзастосунку використано механізм контекстів React. Такий підхід дозволяє зберігати дані, які необхідні багатьом компонентам, в одному місці та отримувати до них доступ без багаторазового передавання через props. Це особливо важливо для даних, які впливають на роботу значної частини інтерфейсу, зокрема інформації про список постів. Його використання дозволяє централізовано зберігати стрічку публікацій та оновлювати її після виконання певних дій користувача, наприклад після створення нового посту приклад коду наведено в лістингу 3.6.

Лістинг 3.6 – Реалізація контексту постів

```
import { createContext, useContext, useState } from "react";
import type { Post } from "../types/Post.ts";

type PostsContextType = {
  posts: Post[],
  setPosts: (posts: Post[]) => void,
  addPost: (post: Post) => void,
  activeVideoId: number | null,
  setActiveVideoId: (id: number | null) => void
}

const PostsContext = createContext<PostsContextType | null>(null);
export function PostsProvider({ children }: { children: React.ReactNode }) {
  const [posts, setPosts] = useState<Post[]>([]);
  const [activeVideoId, setActiveVideoId] = useState<number | null>(null);
  function addPost(post: Post) {
    setPosts(prev => [post, ...prev]);
  }
  return(
    <PostsContext.Provider value={{ posts, setPosts, addPost, activeVideoId,
```

```

setActiveVideoId}}>
  {children}
</PostsContext.Provider>
)
}
export function usePosts(){
  const ctx = useContext(PostsContext);
  if(!ctx) throw new Error("usePosts must be used within PostsProvider");
  return ctx;
}

```

кінець лістингу 3.6

У наведеному фрагменті створюється контекст `PostsContext`, який містить масив постів, функцію його оновлення, функцію додавання нового посту та стан активного відео. Використання окремої функції `addPost` дозволяє додавати новий пост на початок списку, що відповідає логіці хронологічної стрічки, де новіші публікації відображаються вище.

Окремим елементом контексту є стан `activeVideoId`, який використовується для керування активним відео у стрічці. Це дозволяє контролювати поведінку медіаконтенту та уникати ситуацій, коли кілька відео одночасно відтворюються в різних частинах сторінки. Такий підхід покращує користувацький досвід та робить роботу з мультимедійним контентом більш передбачуваною.

Для доступу до контексту постів реалізовано власний хук `usePosts`. Він перевіряє, чи використовується контекст усередині відповідного провайдера, і в разі порушення цієї умови генерує помилку. Це дозволяє швидше виявляти помилки під час розробки та робить структуру взаємодії компонентів більш контрольованою.

Застосування контекстів у клієнтській частині дозволило спростити обмін даними між компонентами, зменшити кількість повторюваного коду та забезпечити узгоджене оновлення інтерфейсу. Такий підхід є доцільним для соціальної мережі, оскільки значна частина функціоналу залежить від спільних даних: стану авторизації, поточного користувача, списку постів та мультимедійного контенту.

Після реалізації загальної структури клієнтської частини важливим етапом стала побудова окремих інтерфейсних компонентів. У React компонент є самостійною частиною системи він може містити власну логіку, обробку подій та відображення даних. Такий підхід дозволяє розділяти інтерфейс на менші частини, повторно використовувати їх у різних місцях застосунку та спростувати подальшу підтримку коду.

Одним із прикладів такого компонента є Menu (ліст. 3.7), який відповідає за відображення навігаційного меню на мобільних та планшетних пристроях. Компонент використовує контекст поточного користувача для визначення стану авторизації та залежно від цього змінює набір доступних елементів інтерфейсу. Якщо користувач не авторизований, відображаються кнопки переходу на головну сторінку, сторінку входу та сторінку реєстрації. Якщо користувач авторизований, замість окремого мобільного меню відображається компонент Sidebar, який містить навігацію для роботи із застосунком.

Лістинг 3.7 – Реалізація компонента мобільного меню

```
import {useCurrentUserContext} from "../context/CurrentUserContext.tsx";
import homeIcon from "../assets/home.png"
import "../components/ui/Sidebar.css";
import {Sidebar} from "../ui/Sidebar.tsx";
import {useNavigate} from "react-router-dom";

export function Menu(){
  const {currentUser, isLoading} = useCurrentUserContext();
  const navigate = useNavigate();
  return(
    <>
      {!currentUser && !isLoading &&(
        <div className="mobile-menu">
          <button onClick={()=>navigate("/")}>
            className="button sidebar-btn">
              <img src={homeIcon} alt="home button icon"/>
            </button>
          <div className="mobile-menu-btn-box">
            <button onClick={()=>navigate("/login")}>
              className="button button-md button-primary">Login</button>
            <button onClick={()=>navigate("/register")}>
              className="button button-md button-outline">Register</button>
            </div>
          </div>
        </div>
      )}
    </>
  );
}
```

```

    })
    {currentUser && (
      <Sidebar/>
    })
  </>
)
}

```

кінець лістингу 3.7

Наведений фрагмент демонструє використання умовного рендерингу в React. Завдяки перевірці значень `currentUser` та `isLoading` компонент визначає, який саме інтерфейс необхідно показати користувачу. Для переходу між сторінками використовується хук `useNavigate`, що дозволяє здійснювати навігацію без перезавантаження сторінки. Такий підхід відповідає принципам побудови односторінкових застосунків, де зміна сторінок відбувається за рахунок роботи клієнтської частини.

В процесі реалізації одним з головних аспектів була візуальна складова застосунку. Для соціальної мережі інтерфейс має велике значення, оскільки саме через нього користувач взаємодіє з основними функціями системи: переглядає стрічку постів, створює публікації, переходить до профілів інших користувачів, виконує пошук та користується системою підписок. Надмірно складний або перевантажений інтерфейс може погіршувати досвід роботи із системою, тому під час розробки було обрано мінімалістичний підхід до побудови UI.

Візуальне оформлення вебзастосунку орієнтоване на швидкий доступ до основного функціоналу та зручне сприйняття контенту. Основну увагу приділено стрічці постів, оскільки вона є центральним елементом соціальної мережі. Елементи інтерфейсу згруповані таким чином, щоб користувач міг швидко перейти до створення посту, перегляду профілю, пошуку або навігації між основними розділами системи. Приклад інтерфейсу головної сторінки вебзастосунку наведено на рисунку 3.2.

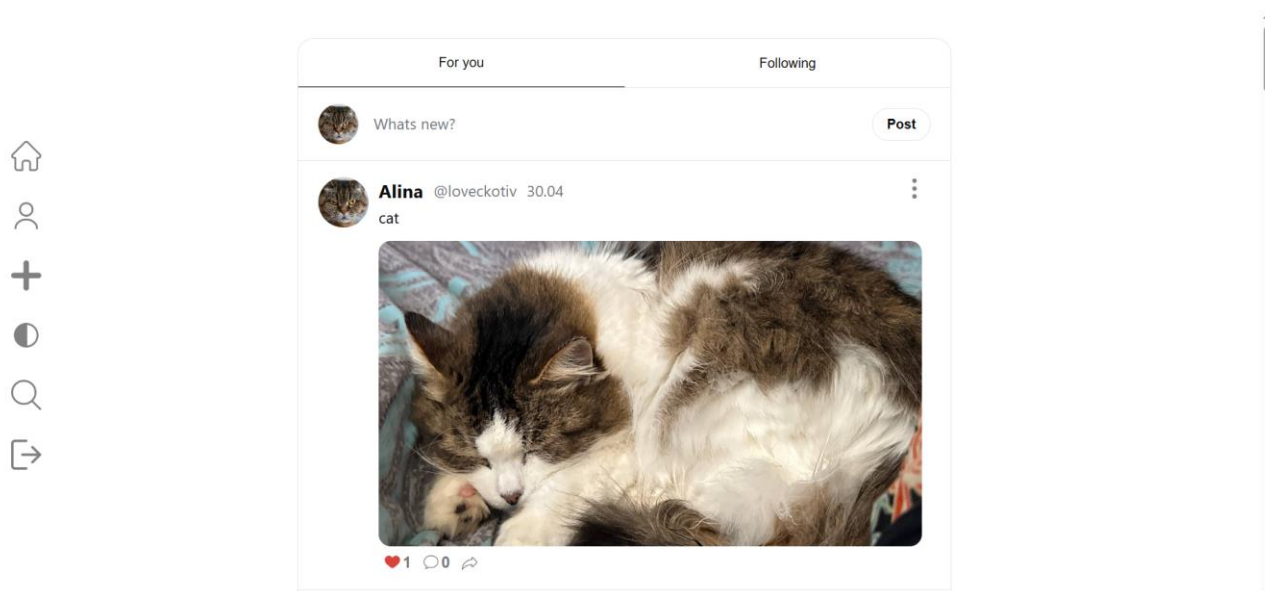


Рисунок 3.2 – Інтерфейс головної сторінки вебзастосунку

Окрім головної сторінки, важливим елементом інтерфейсу є сторінка профілю, яка забезпечує перегляд персональної інформації та взаємодію з контентом конкретного користувача. Відповідно на цій сторінці відображається основна інформація про нього, а також реалізовано можливість переходу між різними типами контенту.

Інтерфейс сторінки профілю організовано таким чином, щоб користувач міг швидко отримати доступ до необхідної інформації та взаємодіяти з нею. Зокрема, передбачено можливість підписки на інших користувачів, перегляду списків підписників і підписок, а також редагування власного профілю у випадку, якщо сторінка належить користувачу.

Особливістю реалізації є відображення контенту у вигляді вкладок або окремих списків, в залежності від типу даних які юзер хоче переглянути це можуть бути всі пости які він додав, вподобав або на які відповів, що дозволяє розділити різні типи активності та зробити інтерфейс більш структурованим та інтуїтивно зрозумілим. Такий підхід сприяє зменшенню когнітивного навантаження на користувача та полегшує навігацію між різними видами контенту. Приклад інтерфейсу сторінки профілю користувача наведено на рисунку 3.3.



Рисунок 3.3 – Інтерфейс сторінки профілю користувача

Важливим аспектом реалізації клієнтської частини є адаптивність інтерфейсу. Оскільки соціальні мережі активно використовуються не лише на настільних комп'ютерах, але й на смартфонах та планшетах, інтерфейс повинен коректно відображатися на екранах різного розміру та забезпечувати зручну взаємодію незалежно від пристрою. Для цього у застосунку використано CSS-стилі та медіа-запити, які дозволяють змінювати розташування, розміри та поведінку елементів залежно від ширини екрана.

У процесі розробки враховано особливості взаємодії користувачів із мобільними пристроями, зокрема обмежений розмір екрана та використання сенсорного керування. Це зумовило необхідність оптимізації інтерфейсу шляхом спрощення навігації, зменшення кількості одночасно відображуваних елементів та підвищення їх доступності. Елементи керування, такі як кнопки та навігаційні панелі, мають достатній розмір для зручного натискання, що покращує користувацький досвід.

У мобільній версії застосунку навігаційні елементи адаптуються до меншого простору екрана. Зокрема, меню, яке на великих екранах може бути розташоване збоку, на мобільних пристроях трансформується у нижню навігаційну панель. Такий підхід дозволяє розмістити основні елементи

керування в зоні досяжності великого пальця, що відповідає сучасним практикам проєктування мобільних інтерфейсів.

Крім того, змінюється структура відображення контенту. Це забезпечує зручність перегляду та прокручування контенту без необхідності масштабування сторінки. Таким чином, адаптивний підхід дозволяє забезпечити коректну роботу вебзастосунку на різних пристроях, покращити зручність використання та підвищити доступність функціоналу для користувачів. Приклад адаптивного відображення інтерфейсу наведено на рисунку 3.4.

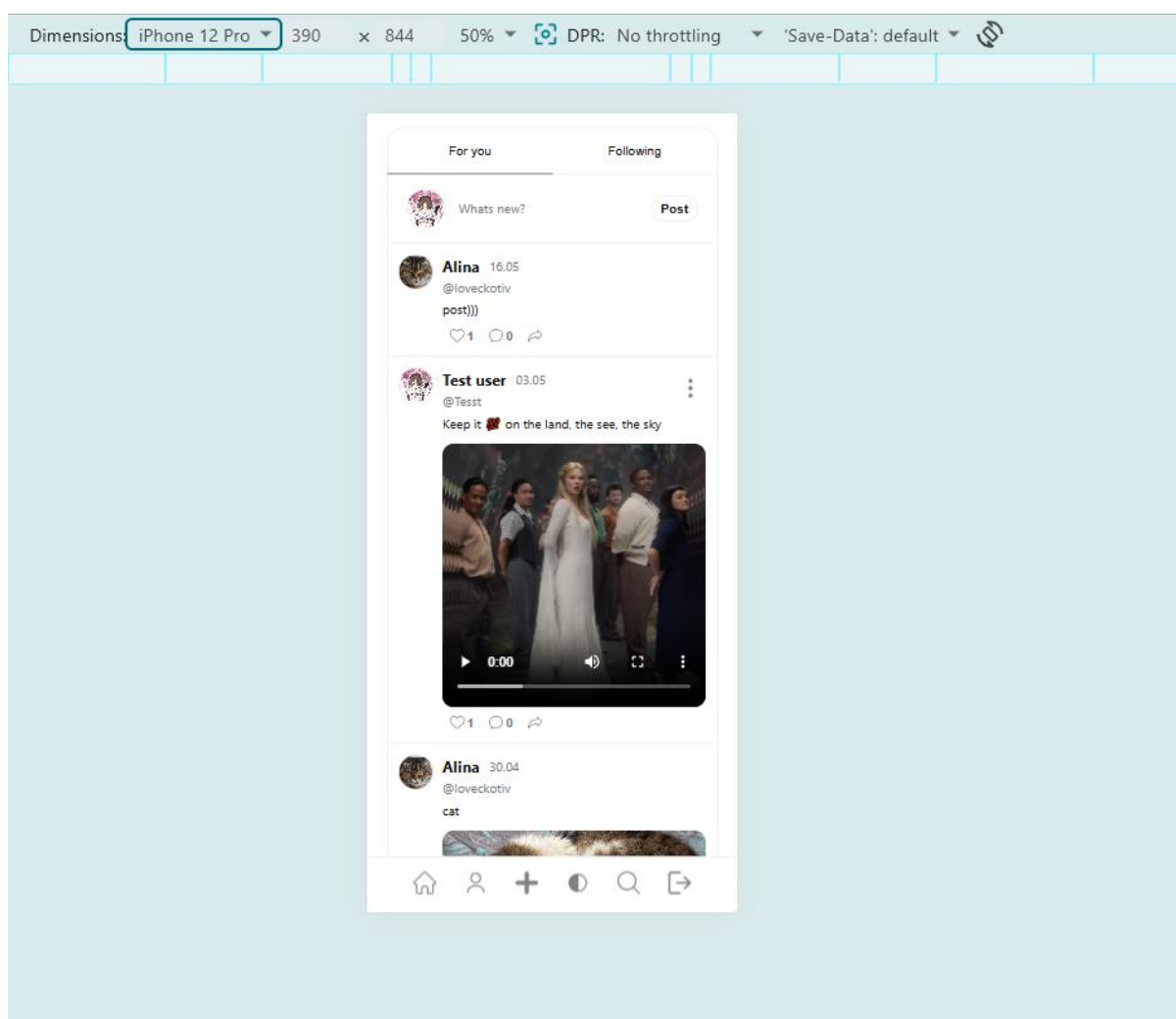


Рисунок 3.4 – Адаптивне відображення інтерфейсу на мобільному пристрої

Таким чином, практична реалізація клієнтської частини охоплює не лише побудову окремих компонентів, але й організацію зручної взаємодії користувача із системою. Використання компонентного підходу, контекстів,

централізованого API-клієнта та адаптивного інтерфейсу дозволило створити клієнтську частину, яка є зручною для підтримки, розширення та використання на різних пристроях.

3.2 Тестування та налагодження інформаційно-комп'ютерної системи

Для забезпечення коректної роботи вебзастосунку важливим етапом є тестування та налагодження програмного забезпечення. Цей етап дозволяє виявити помилки, перевірити коректність реалізації функціоналу та забезпечити стабільний цикл виконання системи. Тестування проводилось протягом усього процесу розробки та охоплювало як серверну, так і клієнтську частини застосунку.

Основна увага була приділена функціональному тестуванню, яке передбачає перевірку відповідності реалізованого функціоналу встановленим вимогам. Зокрема, тестувались основні сценарії взаємодії користувача із системою: реєстрація, авторизація, створення та редагування постів, взаємодія з контентом, пошук, перегляд профілів та робота зі стрічкою.

Для тестування серверної частини застосунку використовувався інструмент Postman, який дозволяє формувати HTTP-запити та аналізувати відповіді сервера [13]. Використання Postman дало можливість перевірити коректність роботи API незалежно від клієнтської частини, що значно спростило процес налагодження.

Для прикладу, на рисунку 3.5 наведено виконання запиту реєстрації користувача, який включає передачу необхідних даних у тілі запиту та отримання відповіді від сервера. У представленому випадку сервер повертає повідомлення про помилку у зв'язку з тим, що користувач із вказаною електронною адресою вже існує в системі. Така поведінка свідчить про наявність реалізованих механізмів валідації та перевірки унікальності даних, що є важливим аспектом забезпечення коректності роботи системи.

Отриманий результат демонструє, що серверна частина обробляє некоректні або дубльовані дані відповідно до визначених правил, повертаючи структуровану відповідь із описом помилки. Це дозволяє клієнтській частині коректно інформувати користувача про причину відмови у виконанні операції та забезпечує більш зрозумілу взаємодію із системою.

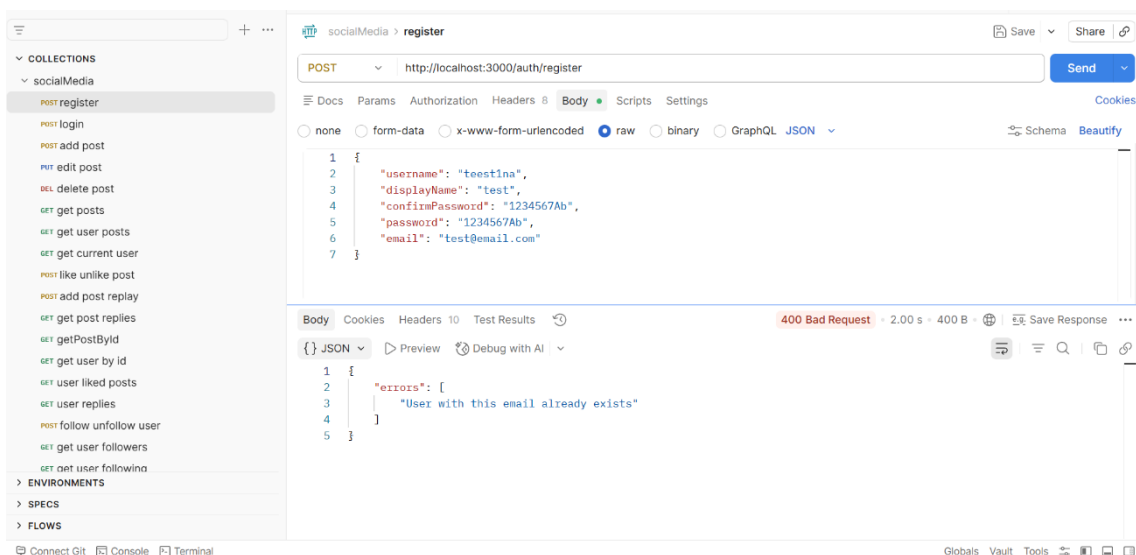


Рисунок 3.5 – Відображення виконання запиту реєстрації користувача

Перевірялась також коректність обробки вхідних даних, формування відповідей, а також обробка помилкових ситуацій. Наприклад, було перевірено сценарії введення некоректних даних, повторної реєстрації з однаковими значеннями та відсутності обов'язкових полів. Це дозволило впевнитися у правильній роботі механізмів валідації та обробки помилок.

Окрім тестування API, здійснювалась перевірка клієнтської частини застосунку. Тестування інтерфейсу включало перевірку коректності відображення компонентів, реакції на дії користувача, переходів між сторінками та роботи адаптивного інтерфейсу на різних розмірах екрана. Виконувалось тестування взаємодії між клієнтською та серверною частинами. У процесі тестування інтерфейсу було виявлено проблему відображення елементів на пристроях із малим розміром екрана. Зокрема, при використанні довгого значення імені користувача текст виходив за межі контейнера, що призводило до

порушення структури інтерфейсу та погіршення користувацького досвіду. Така ситуація виникала через відсутність обмеження ширини текстового блоку та механізмів переносу або обрізання тексту.

Для усунення виявленого недоліку було внесено зміни до стилів інтерфейсу, застосовано обмеження ширини елемента, а також використано CSS-властивості, що забезпечують коректне перенесення або обрізання тексту в межах контейнера. Це дозволило забезпечити стабільне відображення інтерфейсу незалежно від довжини імені користувача та розміру екрана пристрою. Результат виправлення наведено на рисунку 3.6, де продемонстровано вигляд інтерфейсу до та після внесення змін.

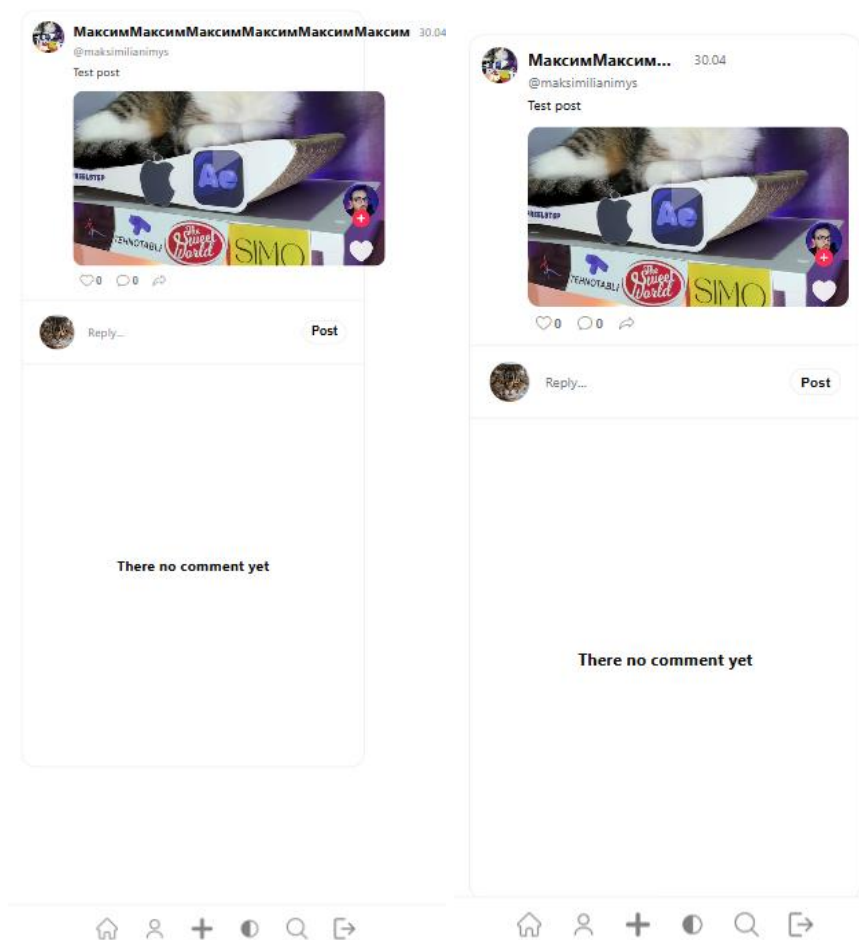


Рисунок 3.6 – Відображення елемента інтерфейсу з довгим іменем користувача до та після виправлення

3.3 Розробка бази даних

У межах розробки вебзастосунку соціальної мережі важливим етапом є проєктування та реалізація бази даних, оскільки саме вона забезпечує зберігання основної інформації системи. До таких даних належать облікові записи користувачів, пости, відповіді, лайки, підписки та медіафайли. Оскільки зазначені сутності мають велику кількість взаємозв'язків, для реалізації сховища даних було обрано реляційну модель бази даних.

Реляційна модель є обґрунтованим вибором для даної системи, оскільки дозволяє чітко описати структуру даних у вигляді таблиць та встановити зв'язки між ними. Наприклад, один користувач може створювати багато постів, один пост може містити кілька медіа, а також мати багато лайків і відповідей. Крім того, система підписок передбачає зв'язок між користувачами, де один користувач може підписуватися на багатьох інших користувачів і водночас мати власних підписників.

Для взаємодії з базою даних у роботі використано Prisma ORM [14]. Prisma дозволяє описувати структуру бази даних у вигляді моделей компонентів. Такий підхід спрощує виконання операцій створення, читання, оновлення та видалення записів, а також дозволяє працювати з базою даних на рівні TypeScript коду без ручного написання великої кількості SQL-запитів.

Використання Prisma також дозволяє забезпечити узгодженість між структурою бази даних і кодом застосунку. Після опису моделей Prisma створює новий клієнт, завдяки якому під час розробки можна отримувати підказки щодо доступних полів, зв'язків та методів роботи з даними. Це зменшує ймовірність помилок при зверненні до бази даних і покращує підтримуваність коду.

Схема бази даних описується і визначаються моделі, їх поля, типи даних, первинні ключі, унікальні обмеження та зв'язки між сутностями. У розроблюваній системі основними моделями є User, Post, Like, Follow та Media. Вони відповідають ключовим об'єктам предметної області соціальної мережі та забезпечують реалізацію основного функціоналу застосунку.

Модель User (ліст. 3.8) використовується для зберігання інформації про користувача. Вона містить унікальний ідентифікатор, ім'я користувача, відображуване ім'я, електронну пошту, пароль, опис профілю, аватар та ознаку верифікації. Окремо в моделі визначено зв'язки з постами, лайками, підписками та підписниками, що дозволяє отримувати пов'язану з користувачем інформацію без дублювання даних.

Лістинг 3.8 – Фрагмент моделі користувача в Prisma

```
model User {
  id      Int    @id @default(autoincrement())
  username String @unique
  displayName String
  email    String @unique
  isVerified Boolean @default(false)
  description String?
  password String?
  avatar   String?
  posts    Post[]
  likes    Like[]
  following Follow[] @relation("UserFollowing")
  followers Follow[] @relation("UserFollowers")
}
```

кінець лістингу 3.8

Модель Post (ліст. 3.9) є однією з центральних моделей системи, оскільки вона відповідає за зберігання публікацій користувачів. Вона містить текст посту, дату створення, дату редагування, ідентифікатор автора та зв'язки з іншими сутностями. Поле `userId` використовується для зв'язку посту з користувачем, який його створив. Завдяки цьому можна отримати автора кожної публікації та сформувати список постів конкретного користувача.

Особливістю моделі Post є підтримка відповідей на пости. Для цього використовується поле `parentId`, яке може містити ідентифікатор батьківського посту. Якщо це поле має значення, пост розглядається як відповідь на інший пост. Така взаємопов'язана структура дозволяє реалізувати механізм відповідей без створення окремої таблиці для коментарів, що в свою чергу спрощує модель даних і дозволяє працювати з постами та відповідями як з єдиним типом сутності.

Також модель Post містить зв'язки з лайками та медіафайлами, щоб отримувати інформацію про кількість взаємодій із постом, а також прикріплені до нього зображення або відео. Такий підхід забезпечує гнучкість при формуванні стрічки публікацій та відображенні повної інформації про пост у клієнтській частині.

Лістинг 3.9 – Фрагмент моделі посту в Prisma

```
model Post {
  id      Int    @id @default(autoincrement())
  text    String
  createdAt DateTime @default(now())
  editedAt DateTime @updatedAt
  userId  Int
  user    User   @relation(fields: [userId], references: [id])
  parentId Int?
  parent  Post?  @relation("PostReplies", fields: [parentId], references: [id], onDelete: Cascade)
  replies Post[] @relation("PostReplies")
  likes   Like[]
  media   Media[]
}
```

кінець лістингу 3.9

Таким чином, опис моделей у Prisma дозволяє формалізувати структуру бази даних та забезпечити зв'язок між сутностями соціальної мережі. Використання моделей User і Post демонструє основний принцип побудови бази даних у системі: кожна сутність має власний набір полів і водночас пов'язана з іншими сутностями через визначені відношення. Це створює основу для коректного зберігання, пошуку та відображення даних у вебзастосунку.

Після опису структури моделей доцільно розглянути спосіб взаємодії серверної частини з базою даних. У розроблюваному застосунку доступ до даних реалізовано через сервісний рівень, який використовує Prisma Client. Такий підхід дозволяє зберігати логіку роботи з базою даних у окремих функціях, що підвищує читабельність коду та спрощує його повторне використання.

Сервіси виконують роль посередника між контролерами та базою даних. Контролер відповідає лише за обробку HTTP-запиту та формування відповіді,

тоді як безпосередня робота з даними виконується у сервісах. Це дозволяє розділити відповідальність між компонентами системи та зробити архітектуру більш зрозумілою та масштабованою [15].

Як приклад реалізації роботи з базою даних розглянемо створення посту (ліст. 3.10). Відповідна функція сервісу виконує операцію додавання нового запису до таблиці постів, а також створює пов'язані записи для медіафайлів. Крім того, у запиті одразу визначається, які пов'язані дані необхідно повернути, зокрема інформація про автора посту, список медіафайлів та значення кількості лайків і відповідей.

Лістинг 3.10 – Фрагмент сервісу створення посту

```
createPost: async(text:string, userId:number, postId: number | null, mediaFiles: { url: string, type:
"VIDEO" | "IMAGE" }[]):Promise<PostWithUser>=>
  prisma.post.create({
    data: {
      text,
      userId,
      parentId: postId,
      media: {
        create: mediaFiles
      }
    },

    include:{
      media: {},
      user: {
        select:{id:true, username: true, displayName:true, avatar:true}
      },
      _count: { select: { replies: true, likes: true } }
    }
  }),
```

кінець лістингу 3.10

Наведений приклад демонструє використання можливостей Prisma для виконання складних операцій над даними в одному запиті. Зокрема, створення посту супроводжується одночасним створенням пов'язаних медіафайлів, що дозволяє уникнути додаткових запитів до бази даних. Використання параметра `include` дає змогу одразу отримати всю необхідну пов'язану інформацію, що

спрощує формування відповіді на стороні сервера та зменшує кількість звернень клієнта.

Такий підхід забезпечує ефективну взаємодію з базою даних, оскільки дозволяє мінімізувати кількість запитів та централізувати логіку доступу до даних. Крім того, використання Prisma забезпечує типізацію результатів запитів, що додатково підвищує надійність роботи програми.

Узагальнюючи, можна зазначити, що розроблена бази даних із використанням Prisma забезпечує структуроване зберігання даних, ефективну реалізацію зв'язків між сутностями та зручну інтеграцію з серверною частиною застосунку. Поєднання реляційної моделі даних і сучасного ORM-інструменту створює надійну основу для подальшого розвитку системи та реалізації її функціональних можливостей.

3.4 Захист інформаційно-комп'ютерної системи

Захист інформаційно-комп'ютерної системи є важливим етапом розробки вебзастосунку соціальної мережі, оскільки система працює з персональними даними користувачів, обліковими записами, контентом та діями, які повинні виконуватися лише авторизованими користувачами. У межах розроблюваного застосунку основну увагу приділено автентифікації користувачів, надання доступу до захищених маршрутів, хешуванню паролів, перевірці JWT-токенів та валідації вхідних даних.

Одним із базових елементів захисту є хешування паролів під час реєстрації користувача. Пароль не зберігається у базі даних у відкритому вигляді, а попередньо обробляється за допомогою бібліотеки bcrypt [16]. Такий підхід дозволяє зменшити ризики витоку реальних паролів користувачів у разі несанкціонованого доступу до бази даних. Під час авторизації введений користувачем пароль порівнюється не з відкритим значенням, а з хешем, який зберігається у базі.

Після успішної авторизації користувачу видається JWT-токен, який використовується для подальшої ідентифікації у системі. Токен містить базову інформацію про користувача, зокрема його ідентифікатор та ім'я користувача, і підписується секретним ключем, що зберігається у змінних середовища. Для перевірки токена у серверній частині реалізовано окремий middleware `verifyToken` (ліст. 3.11), який застосовується до маршрутів, доступних лише авторизованим користувачам. Він отримує токен із заголовка `Authorization`, перевіряє його наявність та коректність, після чого додає дані користувача до об'єкта запиту. Якщо токен відсутній або є недійсним, система повертає помилку авторизації та не передає запит до контролера.

Лістинг 3.11 – Middleware перевірки JWT-токена

```
import "dotenv/config"
import type {Request, Response, NextFunction} from "express";
import jwt from "jsonwebtoken";
import {AppError} from "../utils/AppError.js";
export function verifyToken (req: Request, res: Response, next: NextFunction){
  const authHeader = req.headers.authorization;
  const token = authHeader && authHeader.split(' ')[1];
  if(!token) throw new AppError(401, "You are unauthorized")
  if (!process.env.JWT_SECRET_KEY) throw new AppError(500, "No secret key");
  try {
    req.user = jwt.verify(token, process.env.JWT_SECRET_KEY) as {
      id: number;
      username: string;
    };
    next();
  } catch (err) {
    throw new AppError(401, "Invalid token"); } }

```

кінець лістингу 3.11

На першому етапі виконання коду з HTTP заголовка отримується токен, який передається у форматі `Bearer` і якщо токен не було передано, генерується помилка зі статусом 401, що означає відсутність авторизації. У разі успішної перевірки токена його вміст додається до об'єкта `req.user`, що дозволяє наступним контролерам отримувати інформацію про поточного користувача. Якщо перевірка завершується помилкою, генерується відповідна помилка

авторизації. Такий підхід забезпечує централізовану перевірку доступу та дозволяє повторно використовувати один і той самий механізм для різних захищених маршрутів.

Окрім автентифікації, важливою частиною захисту є авторизація, тобто перевірка прав користувача на виконання конкретної дії. Наприклад, користувач може редагувати або видаляти лише власні пости, тому в контролері додатково перевіряється, чи збігається ідентифікатор автора посту з ідентифікатором поточного користувача. Якщо ці значення не збігаються, система повертає помилку доступу. Це дозволяє запобігти зміні чужого контенту навіть у випадку, якщо користувач є авторизованим.

Ще одним важливим механізмом захисту є валідація вхідних даних. Для цього у застосунку використано набір правил перевірки, які застосовуються до даних, отриманих із тіла запиту. Вона дозволяє перевірити наявність обов'язкових полів, коректність формату електронної пошти, довжину імені користувача, складність пароля та відповідність підтвердження пароля основному значенню. Приклад коду реалізації можна побачити в лістингу 3.12.

Лістинг 3.12 – Фрагмент валідації даних під час реєстрації користувача

```
export const validateRegister = [
  displayNameValidation,
  body("username")
    .trim()
    .notEmpty().withMessage("Username is required")
    .isLength({min: 2, max: 25}).withMessage("Username should at least 2 symbols and no more
than 25 symbols"),
  body("email")
    .trim()
    .notEmpty().withMessage("Email is required")
    .isEmail().withMessage("Email should be in valid format: example@emaiservice.com"),
  passwordValidation,
  body("description")
    .customSanitizer(value => {
      if (value === "") return null;
      return value;
    })
    .optional()
    .isLength({ min: 5, max: 100 }).withMessage("Description should be at least 5 and no more
than 100 characters"),
  body("confirmPassword")
```

```

.optional()
.trim()
.notEmpty().withMessage("Confirm password is required")
.custom((value, {req}) =>{
    if(value !== req.body.password){
        throw new Error("Passwords do not match");
    }
    return true;
}))
]

```

кінець лістингу 3.12

Застосування такого підходу дозволяє відхиляти некоректні запити ще до виконання основної бізнес-логіки. Це зменшує ризик потрапляння некоректних або неповних даних до бази, спрощує обробку помилок та підвищує стабільність роботи систем, а валідація покращує взаємодію з клієнтською частиною, оскільки користувач може отримати зрозуміле повідомлення про причину помилки.

Таким чином, захист інформаційно-комп'ютерної системи реалізовано через поєднання кількох механізмів: хешування паролів, використання JWT-токенів, перевірку авторизації, розмежування доступу до дій, централізовану обробку помилок та валідацію вхідних даних. Сукупність цих підходів дозволяє забезпечити базовий рівень безпеки вебзастосунку соціальної мережі та запобігти типовим помилкам, пов'язаним із некоректним доступом або обробкою даних.

3.5 Розробка документації для програмного продукту

Важливою складовою розробки програмного продукту є створення супровідної документації, яка забезпечує можливість ефективного використання, розгортання та підтримки системи. Документація дозволяє користувачам та розробникам отримати необхідну інформацію про функціональні можливості застосунку, вимоги до середовища виконання та порядок взаємодії з системою.

Для коректної роботи вебзастосунку визначено мінімальні системні вимоги. Оскільки розробка реалізована як клієнт-серверний вебзастосунок, користувачу достатньо мати пристрій із сучасним браузером, який підтримує JavaScript та роботу з мережевими запитами. До таких браузерів належать Google Chrome, Mozilla Firefox та Microsoft Edge. Для стабільної роботи також необхідне підключення до інтернету.

Серверна частина застосунку потребує наявності середовища виконання Node.js, встановленого менеджера пакетів npm, а також системи керування базами даних PostgreSQL. Крім того, необхідно налаштувати змінні середовища, зокрема секретний ключ для генерації JWT-токенів, параметри підключення до бази даних та інші конфігураційні значення.

Процес встановлення та розгортання програмного продукту передбачає виконання кількох основних етапів. На першому етапі здійснюється клонування репозиторію з вихідним кодом застосунку. Після цього необхідно встановити залежності для серверної та клієнтської частин за допомогою менеджера пакетів. Далі виконується налаштування змінних середовища, що містять параметри конфігурації системи.

Наступним кроком є ініціалізація бази даних. Для цього використовується інструментарій Prisma, який дозволяє виконати міграції та створити необхідні таблиці відповідно до описаних моделей. Після успішного налаштування бази даних виконується запуск серверної частини застосунку, яка починає обробляти HTTP-запити. Клієнтська частина запускається окремо та забезпечує взаємодію користувача із системою через інтерфейс.

Для користувачів застосунку передбачено базові довідкові матеріали, які описують основні можливості системи. До таких можливостей належать реєстрація та авторизація, створення та редагування постів, перегляд стрічки, взаємодія з контентом через лайки та відповіді, а також робота з профілем користувача. Інтерфейс застосунку реалізовано з урахуванням принципів інтуїтивності, що дозволяє користувачу швидко освоїти основні функції без необхідності детального вивчення інструкцій.

Окремо варто зазначити, що довідкова інформація інтегрована безпосередньо у логіку інтерфейсу. Наприклад, повідомлення про помилки або підказки щодо некоректно введених даних відображаються у відповідь на дії користувача. Такий підхід дозволяє забезпечити контекстну допомогу без створення окремого файлу довідки або складної системи документації.

Таким чином, розроблена документація охоплює основні аспекти використання та розгортання програмного продукту. Вона дозволяє забезпечити зрозумілий процес встановлення системи, її налаштування та ефективного використання як кінцевими користувачами, так і розробниками.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи бакалавра було досягнуто поставлену мету, яка полягала у розробці вебзастосунку соціальної мережі для обміну даними з використанням сучасних вебтехнологій. У процесі роботи було повністю виконано поставлені завдання, що охоплюють як теоретичні аспекти дослідження предметної області, так і практичну реалізацію програмного продукту.

У процесі виконання роботи було проведено аналіз предметної області та існуючих соціальних мереж. У результаті аналізу було визначено основні функціональні можливості сучасних платформ, зокрема механізми формування стрічки, взаємодії користувачів та організації контенту, а також виявлено їхні недоліки. Це дозволило сформувані обґрунтовані вимоги до розроблюваної системи та визначити напрями її реалізації.

Було опрацьовано сучасні підходи до розробки вебзастосунків та обґрунтовано вибір технологічного стеку. Використання React і TypeScript для клієнтської частини, Node.js і Express для серверної частини, а також PostgreSQL для роботи з даними відповідає сучасним світовим тенденціям у сфері веброзробки.

У ході роботи було спроектовано архітектуру вебзастосунку, визначено основні компоненти системи та їх взаємодію. Застосування клієнт-серверної моделі дозволило розмежувати відповідальність між частинами системи, оптимізувати процес обробки запитів та забезпечити структуровану організацію програмного коду.

Було розроблено структуру бази даних, яка забезпечує ефективне зберігання інформації про користувачів, пости та взаємодії між ними. Реалізація моделі даних із використанням ORM-інструменту Prisma дозволила підвищити спростити виконання запитів та зменшити ймовірність помилок при роботі з базою даних.

У процесі реалізації серверної частини застосунку було створено набір API для обробки запитів, організовано бізнес-логіку та забезпечено взаємодію з базою даних. Було реалізовано основний функціонал системи, включаючи роботу з постами, користувачами, підписками та пошуком.

Було реалізовано клієнтську частину вебзастосунку із використанням компонентного підходу. Інтерфейс системи побудовано з урахуванням принципів зручності та інтуїтивності, а також забезпечено адаптивність для різних типів пристроїв. Це дозволило створити зручне середовище для взаємодії користувача із системою.

У межах забезпечення безпеки було реалізовано механізми автентифікації та авторизації користувачів із використанням JWT. Забезпечено перевірку прав доступу до ресурсів, а також валідацію вхідних даних, що дозволило підвищити рівень захисту інформації та запобігти несанкціонованому доступу до системи.

У процесі тестування було перевірено працездатність основного функціоналу застосунку, виявлено та усунено недоліки, що виникали під час розробки. Це дозволило підтвердити відповідність системи поставленим вимогам та забезпечити стабільну роботу у типових сценаріях використання.

Отримані результати роботи відповідають сучасним світовим тенденціям у сфері розробки вебзастосунків, зокрема використанню клієнт-серверної архітектури, компонентного підходу та ефективних механізмів обробки великих обсягів даних.

Виконана робота пов'язана з напрямками науково-дослідної діяльності у сфері вебтехнологій та інженерії програмного забезпечення і може бути використана як база для подальших досліджень у даній галузі. Отримані результати мають як практичну, так і навчально-наукову цінність.

Подальший розвиток роботи може бути спрямований на розширення функціональних можливостей застосунку, зокрема впровадження алгоритмів рекомендацій, покращення механізмів обробки контенту, оптимізацію продуктивності при високих навантаженнях та посилення засобів забезпечення безпеки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Stieglitz S. Social media and society. i-com. 2025. URL: <https://doi.org/10.1515/icom-2025-0029> (дата звернення: 15.04.2026).
2. Lozano-Blasco R., Mira-Aladrén M., Gil-Lamata M. Social media influence on young people and children: Analysis on Instagram, Twitter and YouTube. Comunicar. 2023. Vol. 31. № 74. URL: <https://doi.org/10.3916/c74-2023-10> (дата звернення: 15.04.2026).
3. Клішевич І. Є. Вебзастосунок «Соціальна мережа»: кваліфікаційна робота на здобуття освітнього ступеня «бакалавр»: спец. 122 «Комп'ютерні науки». ЧНУ ім. Петра Могили. Миколаїв, 2024. 92 с. URL: <https://krs.chmnu.edu.ua/jspui/handle/123456789/3649> (дата звернення: 16.04.2026).
4. Managing State – React. React. URL: <https://react.dev/learn/managing-state> (дата звернення: 19.04.2026).
5. Documentation – TypeScript for JavaScript Programmers. TypeScript: JavaScript With Syntax For Types. URL: <https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html> (дата звернення: 20.04.2026).
6. Asynchronous context tracking. Node.js v25.9.0 Documentation. Node.js. Run JavaScript Everywhere. URL: https://nodejs.org/docs/latest/api/async_context.html (дата звернення: 20.04.2026).
7. What Is PostgreSQL? URL: <https://www.postgresql.org/docs/current/intro-what-is.html> (дата звернення: 22.04.2026).
8. Express – Node.js web application framework. URL: <https://expressjs.com/> (дата звернення: 27.04.2026).
9. Express routing. Node.js web application framework. URL: <https://expressjs.com/en/guide/routing.html> (дата звернення: 27.04.2026).
10. Express Tutorial Part 4. Routes and controllers. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/Express_Nodejs/routes (дата звернення: 02.05.2026).

11. Using Express middleware. URL: <https://expressjs.com/en/guide/using-middleware.html> (дата звернення: 02.05.2026).

12. Documentation – Type Inference. TypeScript. URL: <https://www.typescriptlang.org/docs/handbook/type-inference.html> (дата звернення: 03.05.2026).

13. What is API Testing? A Guide to Testing APIs. Postman. The World's Leading API Platform. URL: <https://www.postman.com/api-platform/api-testing/> (дата звернення: 03.05.2026).

14. What is Prisma ORM? Prisma. URL: <https://www.prisma.io/docs/orm> (дата звернення: 07.05.2026).

15. Understanding the Service Layer in Software Architecture. Medium. URL: <https://medium.com/@navroops38/understanding-the-service-layer-in-software-architecture-df9b676b3a16> (дата звернення: 08.05.2026).

16. Bcrypt – npm. Npm. URL: <https://www.npmjs.com/package/bcrypt> (дата звернення: 11.05.2026).