МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Навчально-науковий інститут інформаційних технологій та бізнесу
Кафедра інформаційних технологій та аналітики даних

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавра

на тему: «**Проєктування та розробка соціальної мережі на тематику фінансів та інвестування»**

Виконала: студентка 4 курсу, групи КН-41
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Лайтер Ярина Семенівна

Керівник: *викладач кафедри інформаційних технологій та аналітики даних*
Ляховчук Сергій Васильович

Рецензент: *ФОП в галузі інформаційних технологій, викладач кафедри менеджменту та маркетингу НаУОА*
Шпак Андрій Григорович

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ
Завідувач кафедри інформаційних технологій та аналітики даних _____
(проф., д.е.н. Кривицька О.Р.)
Протокол №10 від 28 травня 2025 р.

Острог, 2025

АНОТАЦІЯ
кваліфікаційної роботи
на здобуття освітнього ступеня бакалавра

Тема: Проєктування та розробка соціальної мережі на тематику фінансів та інвестування

Автор: *Лайтер Ярина Семенівна*

Науковий керівник: викладач, фахівець-практик, Ляховчук Сергій Васильович

Захищена «……»…… 20__ року.

Пояснювальна записка до кваліфікаційної роботи: 65 с., 18 рис., 3 табл., 2 додаток, 35 джерел.

Ключові слова: соціальна мережа, фінанси, інвестування, Angular, .NET, MediatR, JWT, хаби, пости, повідомлення

Короткий зміст праці:
У роботі розглянуто процес розробки соціальної мережі, орієнтованої на фінансову тематику та інвестування. Проведено аналіз сучасних підходів до побудови соціальних платформ, виділено їхні основні функціональні елементи, обґрунтовано вибір технологій та інструментів. Система реалізована на базі клієнт-серверної архітектури з використанням Angular на клієнтській частині та ASP.NET Core з шаблоном MediatR на сервері.
У межах реалізації розроблено функціонал реєстрації та автентифікації користувачів із використанням JWT та Google OAuth, профілі користувачів, стрічку публікацій із можливістю створення постів та коментарів, систему сповіщень, приватні повідомлення, а також хаби — спільноти за інтересами з можливістю модерації. Забезпечено контейнеризацію через Docker, реалізовано багаторівневу обробку виняткових ситуацій, використано підхід CQRS для логічного поділу операцій над даними.
Система є масштабованою, модульною та розширюваною, має потенціал подальшого розвитку як платформи для фінансової взаємодії користувачів.

_____

ANNOTATION
of qualification paper
for bachelor's degree

Theme: Design and Development of a Financial- and Investment-Themed Social Network

Author: Yaryna Semenivna Laiter

Scientific supervisor: Lecturer, Practicing Specialist, Liakhovchuk Serhii Vasylovych

Defended «……» …… of 20__ року.

Explanatory note to the qualification work: 65 pages, 18 figures, 3 tables, 2 appendix, 35 sources.

Keywords: social network, finance, investment, Angular, .NET, MediatR, JWT, hubs, posts, messaging

Summary of the paper:
This thesis presents the process of designing and developing a social networking platform focused on finance and investment topics. The work includes an overview of current approaches to building social platforms, highlights key functional components, and justifies the choice of technologies and tools. The system is implemented based on a client-server architecture using Angular for the frontend and ASP.NET Core with the MediatR pattern on the backend.
The implemented functionality includes user registration and authentication via JWT and Google OAuth, user profiles, a news feed with post creation and commenting features, a notification system, private messaging, and thematic hubs with moderation capabilities. The solution supports containerization via Docker, robust exception handling, and applies the CQRS pattern for separating read and write operations.
The system is scalable, modular, and extensible, with strong potential for further development into a full-fledged financial interaction platform.

_____

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

Скорочення / Термін	Розшифрування / Назва	Опис
DTO	Data Transfer Object	Об’єкт для передачі даних між шарами системи.
ORM	Object-Relational Mapping	Технологія взаємодії з базами даних через об’єктну модель.
HTTP	Hypertext Transfer Protocol	Протокол передавання гіпертексту.
SQL	Structured Query Language	Мова структурованих запитів для роботи з базами даних.
JSON	JavaScript Object Notation	Текстовий формат для обміну даними.
Docker	—	Платформа для створення та запуску ізольованих середовищ (контейнерів).
JWT	JSON Web Token	Формат передачі зашифрованих токенів для автентифікації та авторизації.
API	Application Programming Interface	Інтерфейс прикладного програмування; взаємодія між компонентами.
REST	Representational State Transfer	Архітектурний стиль для вебсервісів на базі HTTP.
CI/CD	Continuous Integration / Continuous Delivery	Методології безперервної інтеграції та доставки ПЗ.
Angular	—	Фронтенд-фреймворк для створення SPA від Google.
PostgreSQL	—	Реляційна об’єктно-орієнтована СУБД з відкритим кодом.
TypeScript	—	Мова програмування на базі JavaScript зі статичною типізацією.

Azure	—	Хмарна платформа Microsoft для розгортання й масштабування сервісів.
AutoMapper	—	Бібліотека для автоматичного мапінгу між об'єктами.
.NET	—	Кросплатформна платформа від Microsoft для серверної розробки.

ВСТУП

Із розвитком цифрових технологій дедалі більше аспектів людської діяльності переходять в онлайн-простір, і фінансова сфера — не виняток. Водночас більшість сучасних платформ, що стосуються інвестування та фінансової грамотності, орієнтовані на досвідчених користувачів, створені для іноземного ринку або є надто загальними. Вони часто не забезпечують комфортного середовища для щоденного спілкування, обміну думками та підтримки інтересу до теми. Це створює запит на новий формат тематичної соціальної мережі, що поєднує знайомі користувачам механізми — стрічку публікацій, особисті повідомлення, підписки, тематичні хаби — з контентом, сфокусованим на фінансовій тематиці.

Запропонований у межах дослідження програмний продукт спрямований на створення такої соціальної мережі для користувачів, зацікавлених у фінансах, інвестуванні та економіці в широкому сенсі. Очікується, що платформа стане місцем формування активної спільноти, де користувачі зможуть ділитися досвідом, навчатися, обговорювати актуальні теми та надихати одне одного.

Актуальність теми полягає в потребі створення сучасної спеціалізованої соціальної платформи, яка б відповідала вимогам користувачів щодо функціональності, доступності та інтерактивності, а також підтримувала розвиток фінансової культури та грамотності.

Метою дослідження є розробка сучасного вебзастосунку — соціальної мережі, що забезпечує користувачам можливість взаємодіяти між собою, приєднуватися до тематичних хабів, публікувати інформаційні чи розважальні матеріали, вести дискусії, підписуватись на інших користувачів і отримувати сповіщення про важливі події.

Для досягнення цієї мети поставлено такі завдання:

- провести аналіз предметної області та існуючих аналогів;
- спроєктувати архітектуру інформаційної системи з урахуванням принципів масштабованості та безпеки;
- обрати інструменти та методи реалізації клієнтської і серверної частини;

- реалізувати основні модулі системи та провести їх тестування.

Об’єктом дослідження є вебзастосунок — спеціалізована соціальна мережа, що включає клієнтську та серверну частини і дозволяє інтерактивну взаємодію між користувачами.

Предметом дослідження виступають інструментальні засоби проєктування та реалізації подібних систем: вебфреймворки, бібліотеки для побудови інтерфейсу, методи організації зберігання та обробки даних, технології автентифікації та обміну повідомленнями, а також засоби забезпечення масштабованості, надійності й підтримки зручного користувацького досвіду.

Методи дослідження включають аналіз конкурентного середовища, структурне та об’єктно-орієнтоване проєктування, а також сучасні практики розробки програмного забезпечення для реалізації, тестування й розгортання системи.

РОЗДІЛ 1. АНАЛІЗ ТЕМАТИКИ ТА ПЛАНУВАННЯ СОЦІАЛЬНОЇ МЕРЕЖІ

1.1. Сучасні підходи до створення соціальних мереж

Проєктування сучасних соціальних мереж — це складний інженерний процес, що охоплює архітектурні, інфраструктурні та UX-рішення. З розвитком цифрових технологій і зростанням очікувань користувачів такі платформи мають бути не лише функціональними, а й масштабованими, продуктивними, доступними з будь-якого пристрою та здатними обробляти тисячі одночасних запитів із мінімальними затримками. Усі ці вимоги впливають на вибір архітектурного підходу, стеку технологій, структури бази даних та стратегії підтримки і розгортання продукту.

Одним із найпоширеніших архітектурних підходів є мікросервісна архітектура, яка забезпечує ізольовану реалізацію окремих бізнес-функцій. Це дозволяє масштабувати кожен компонент незалежно, адаптуватися до навантажень і швидко вносити зміни без ризику для всієї системи. Наприклад, система може мати окремі сервіси для авторизації, обробки публікацій, сповіщень, повідомлень, пошуку тощо. Такий підхід, хоч і ускладнює оркестрацію, дає гнучкість у розгортанні та обслуговуванні. Світові гравці, такі як Twitter, початково реалізований на Ruby on Rails, з часом перейшов до мікросервісної архітектури на Java і Scala для підвищення масштабованості та гнучкості [1].

Ще одним важливим аспектом є система кешування, яка дозволяє зменшити кількість запитів до бази даних і значно прискорити відображення популярного контенту. Кешування можна реалізувати як на стороні сервера (через Redis, Memcached), так і на клієнтській стороні через локальні сховища. У великих мережах, таких як Reddit, кешування відіграє критичну роль у забезпеченні швидкодії при роботі зі стрічкою, профілями, списками підписок тощо — платформа активно використовує Redis і Memcached для кешування стрічки публікацій і профілів користувачів [2]. В межах мого проєкту кешування реалізовано на рівні API-шару та Angular-сервісів з урахуванням оптимізації запитів і повторного використання даних.

Сховище даних у сучасних соціальних мережах зазвичай гібридне. Для транзакційних операцій (авторизація, пости, коментарі, повідомлення) зручно використовувати реляційні бази даних (наприклад, PostgreSQL або MySQL), які гарантують консистентність даних. Для масштабного зберігання зображень, логів або нечітко структурованих даних часто залучають NoSQL-рішення — MongoDB, Cassandra, DynamoDB. Наприклад, платформа Quora працює з MySQL у поєднанні з індексацією й кешуванням для підвищення швидкості доступу до інформації [3].

Користувацький інтерфейс (UI) здебільшого реалізується за допомогою односторінкових застосунків (SPA) на фреймворках Angular, React або Vue.js. Це дозволяє досягти максимальної інтерактивності, мінімізуючи навантаження на сервер. Такі клієнти працюють через REST API або GraphQL, що дозволяє чітко розділити клієнтську й серверну логіку. Зокрема, в моєму проєкті використано Angular CLI як основу клієнта, де реалізовано маршрутизацію, сервіси для взаємодії з API, захист приватних маршрутів, а також реактивні форми й локальний стан.

На рівні асинхронної обробки дій користувачів, як-от надсилання повідомлень, створення сповіщень або фонове збереження даних, у великих системах застосовують системи черг повідомлень — Apache Kafka, RabbitMQ, Azure Service Bus. Ці інструменти дозволяють розвантажити основний потік запитів і забезпечити стійкість до пікових навантажень. У моєму проєкті використано подієву модель на сервері — кожна ключова дія користувача (наприклад, коментування чи підписка) генерує подію, що обробляється окремим обробником за допомогою MediatR.

Ще одним важливим елементом сучасного підходу є CI/CD (Continuous Integration / Continuous Deployment). Це підхід, що автоматизує перевірку, тестування й розгортання змін. Інструменти на кшталт GitHub Actions, GitLab CI/CD, Azure Pipelines або Jenkins дозволяють зменшити кількість помилок при оновленні та підтримувати стабільність продукту. Навіть у невеликих проєктах CI/CD-практики значно підвищують якість і передбачуваність оновлень [4]. У проєкті інтегровано GitHub Actions для виконання перевірок коду, тестів та автоматичного публікування образів у Docker Registry.

Безпека також є невід’ємним компонентом сучасних підходів. Це охоплює як автентифікацію й авторизацію, так і захист від CSRF, XSS та інших атак. Більшість систем використовують токенну автентифікацію (JWT), шифрування даних у транзиті (HTTPS), контроль доступу через ролі й політики. У проєкті застосовано власну реалізацію JWT-автентифікації та інтеграцію з Google OAuth, а також механізми захисту маршрутів на Angular і контролерів на .NET API.

Таким чином, сучасні соціальні мережі — це високонавантажені, складні системи, що поєднують найкращі практики у сфері розробки, безпеки, взаємодії з даними та інтерфейсу. Навіть у невеликих проєктах важливо орієнтуватись на ці підходи, адаптуючи їх відповідно до масштабу та цілей, — адже саме ці рішення формують основи надійного й зручного продукту.

1.2. Особливості проєктування тематичної мережі про фінанси

Проєктування соціальної мережі, присвяченої фінансовій тематиці, потребує комплексного підходу, що враховує як технологічні рішення, так і поведінкові характеристики цільової аудиторії. На відміну від універсальних платформ, таких як Facebook, Instagram або Twitter, тематичні соціальні мережі мають вужчий фокус, проте дозволяють будувати більш якісну взаємодію між учасниками спільноти, об’єднаними спільними інтересами, цілями та рівнем залученості. Це особливо актуально в контексті фінансів — сфери, яка водночас вимагає високої обізнаності, критичного мислення і довіри між учасниками.

Фінансова тематика була обрана не випадково. За даними Організації економічного співробітництва та розвитку (OECD), рівень базової фінансової грамотності серед дорослого населення залишається на низькому рівні: лише 52% мають елементарні знання у сфері фінансів [5]. Це створює критичний попит на освітні платформи, які були б доступними, зручними та спрямованими не лише на навчання, а й на побудову спільноти. Водночас зростає інтерес до особистого фінансового планування, інвестування, криптовалют, облігацій внутрішньої

державної позики (ОВДП) та інших інструментів, що потребують простого та безпечного середовища для обговорення.

У світовому контексті вже існують численні ресурси, присвячені фінансам — зокрема, Reddit-спільноти, біржові додатки з чатами, тематичні блоги або канали на YouTube [6]. Проте більшість із них орієнтовані на англомовну аудиторію та враховують реалії інших країн: іншу правову базу, оподаткування, доступ до бірж або банківських сервісів. У той же час українському користувачу часто бракує платформи, яка враховувала б локальний контекст — мову, податкове законодавство, доступні інструменти інвестування, економічну ситуацію під час війни, банківські обмеження та загальну нестабільність.

Проєктована мережа ставить за мету стати майданчиком для взаємодії таких груп: початківців, які хочуть навчитися вести бюджет чи робити перші кроки в інвестуванні; досвідчених користувачів, які шукають майданчик для обміну аналітикою чи дискусій; лідерів думок, що створюють авторський контент на тему економіки, бізнесу та фінансових технологій. Соціальна динаміка платформи має поєднувати вже знайомі механіки (стрічка публікацій, система підписок, персональні повідомлення, сповіщення, можливість коментування), з функціями, властивими саме фінансовій тематиці. Наприклад, це може бути підтримка форматів публікацій зі вставками графіків, таблиць, біржових даних, обговорення новин в рамках хабів за напрямами (інвестування, податки, заощадження тощо).

Цільова аудиторія мережі — це молоді українці віком від 18 до 35 років, які прагнуть підвищити свою фінансову обізнаність, беруть участь в економічному житті, цікавляться новинами бізнесу та інвестицій, а також готові ділитися досвідом з іншими [7]. Водночас платформа має залишатись інклюзивною — вона повинна бути доступною як для тих, хто щойно дізнається, що таке інфляція чи депозит, так і для людей, які вже мають свій інвестиційний портфель.

Особливу увагу під час проєктування було приділено тематичній організації контенту — хаби дозволяють групувати користувачів і матеріали за інтересами (наприклад, "Бюджет для студентів", "Криптовалюти", "Оподаткування ФОП", "Фінансові новини", "Фінанси під час війни"). Така структура не лише спрощує

навігацію, але й створює ефект "місця належності", де користувач почувається частиною мікроспільноти.

Також важливим аспектом стало питання довіри та модерації. У фінансовій сфері поширена проблема дезінформації, маніпуляцій та шахрайства. Саме тому важливо реалізувати багаторівневу модерацію: автоматичні фільтри, ручне втручання адміністраторів хабів, а також системи репутації або позначки надійності для користувачів. Також буде запроваджено механізми скарг на контент, перевірку джерел інформації у публікаціях та маркування неперевірених тверджень.

Оскільки платформа орієнтована на український ринок, значну увагу приділено локалізації: весь інтерфейс — українською мовою, з адаптацією термінології, яка відповідає реаліям українського фінансового середовища (наприклад, замість "IRA account" — "депозит", "ОВДП", "єПідтримка", тощо). Також буде підтримуватися робота з українськими банками, податковими кодами, формами реєстрації бізнесу, що робить платформу унікальною в порівнянні з західними аналогами.

З технологічного боку, архітектура системи передбачає масштабованість та гнучкість, оскільки передбачено як ріст кількості користувачів, так і збільшення функціональності. Запропонована реалізація використовує сучасні підходи — поділ на модулі, адаптивний інтерфейс, RESTful API з аутентифікацією, серверну пагінацію контенту, зберігання зображень у хмарі, а також підтримку DTO для зручності обміну даними між клієнтом і сервером.

У перспективі платформа може інтегрувати інструменти для автоматизованого збору фінансових даних (наприклад, через API банків або бірж), реалізовувати освітні курси, функції менторства, сервіси аналізу портфелів тощо. Однак базова ідея залишається сталою — створити українську, тематичну, безпечну і зручну соціальну мережу про фінанси, яка б об’єднала користувачів різного рівня досвіду в єдину спільноту.

Таким чином, розробка такої платформи не обмежується лише технічним викликом. Це також соціальна ініціатива, яка відповідає актуальному запиту українського суспільства на підвищення рівня фінансової культури, підтримку

молоді в умовах економічної турбулентності та створення середовища, де можна безпечно обговорювати складні теми, вчитись і ділитись досвідом.

1.3. Аналіз існуючих рішень

На сучасному етапі розвитку інформаційного суспільства соціальні мережі дедалі частіше перетворюються з платформ для особистої комунікації на повноцінні середовища для професійного обміну знаннями, формування спільнот за інтересами та поширення аналітичного або освітнього контенту. Особливо це стосується таких чутливих і складних сфер, як особисті фінанси, інвестування, макроекономіка чи біржова аналітика. У цьому контексті аналіз існуючих цифрових платформ, які певною мірою реалізують подібні функції, є ключовим кроком для виявлення практичних рішень, які вже довели свою ефективність, а також недоліків, що мають бути враховані при створенні нової системи. Основна увага в даному розділі приділяється таким сервісам, як Twitter, Reddit, Quora, Stocktwits і Facebook-групи — як найбільш релевантним з огляду на фінансову тематику, наявність активної користувацької бази та відкритість до публічного обговорення.

Twitter посідає особливе місце в екосистемі платформ, де ведеться обговорення фінансових ринків, економічної політики та інвестиційних стратегій. Завдяки короткому формату повідомлень, а також можливості супроводжувати публікації графіками, гіперпосиланнями та тегами, платформа забезпечує швидке поширення новин, прогнозів і мікроаналітики. Активна присутність аналітиків фондового ринку, фінансових журналістів і впливових блогерів створює динамічне середовище, в якому нова інформація циркулює практично в реальному часі. Втім, швидкість не завжди дорівнює достовірності: Twitter часто потерпає від дезінформації, маніпулятивних меседжів та інсайдерських витоків сумнівної якості. Крім того, платформа не передбачає зручного механізму ведення тривалих, тематично впорядкованих дискусій, що робить її менш придатною для систематизованого обміну знаннями. Відсутність чіткої категоризації, низький рівень модерування та

ризик створення «інформаційної бульбашки» через алгоритмічну стрічку лише посилюють ці недоліки [8].

Reddit пропонує інший підхід до побудови інформаційної взаємодії: його структура базується на тематичних підрозділах — сабреддітах, кожен з яких фокусується на конкретному питанні чи спільному інтересі. У фінансовому контексті особливо популярними є r/WallStreetBets, орієнтований на високоризиковий трейдинг і спекулятивні стратегії, та r/PersonalFinance, який більше стосується повсякденного фінансового планування. Reddit надає користувачам інструменти для створення довгих, аргументованих дописів, а також підтримує деревовидну структуру коментарів, що забезпечує логічну послідовність обговорення. Голосування за публікації дозволяє виводити на перший план найбільш релевантний або корисний контент. Водночас відкрита анонімність, обмежені можливості верифікації авторів і неоднорідність рівня експертизи створюють ґрунт для поширення неперевірених порад, суб'єктивних суджень або навмисної маніпуляції. Часто трапляються приклади «групового мислення», коли окремі популярні думки підтримуються без належної критичності, що у фінансовому середовищі може мати ризиковані наслідки [9].

Quora є платформою, орієнтованою на запитально-відповідну модель, яка стимулює користувачів до формулювання конкретних питань і надання змістовних відповідей. У цьому контексті вона нагадує енциклопедичну систему, що доповнюється особистим досвідом або експертними судженнями користувачів. У галузі фінансів Quora демонструє високу якість відповідей, особливо в розділах, що стосуються загальних економічних понять, моделей інвестування, банківських продуктів або фіскальної політики. Проте платформа в меншій мірі сприяє формуванню сталої спільноти: між користувачами відсутня інфраструктура для постійної комунікації, не передбачено гнучких інструментів категоризації тем або підтримки багаторівневих обговорень. Крім того, платформа не створює ефекту "соціального простору", що знижує рівень залучення користувачів.

Stocktwits є спеціалізованою платформою, орієнтованою виключно на ринок цінних паперів. Її головна функція полягає у поширенні коротких повідомлень із

біржовими символами (тікерами), що дозволяє відслідковувати публічну активність та настрої навколо конкретних активів. Такий підхід підходить для миттєвого аналізу ринкових трендів, однак він відносно мало придатний для освітнього контенту, стратегічного планування чи обговорення фінансових продуктів, що не пов'язані з біржею. Структура платформи навмисно спрощена, що знижує гнучкість в адаптації під ширші інформаційні цілі. Наявність великої кількості початківців без досвіду веде до надмірного ентузіазму, часто не підкріпленого знанням ринку чи об'єктивними аналітичними критеріями [10].

Facebook-групи, незважаючи на відносно просту технічну реалізацію, залишаються потужним інструментом формування спільнот в українському середовищі. У багатьох випадках групи на фінансову тематику функціонують як майданчики для консультацій, обміну новинами та особистими історіями. Проте їх функціональність залишається обмеженою у порівнянні зі спеціалізованими платформами. Алгоритмічна стрічка не завжди забезпечує релевантність відображення контенту, модерація є несистемною, а відсутність структурованої навігації між темами ускладнює пошук конкретної інформації. Частим є також засилля комерційного спаму, надмірної реклами або шахрайських схем. Попри все це, саме Facebook-групи демонструють значний попит на україномовний, локалізований і прикладний фінансовий контент, що свідчить про незадоволений запит аудиторії.

Аналіз зазначених платформ дозволяє виокремити кілька важливих висновків. З одного боку, існує чіткий попит на доступний і динамічний фінансовий контент, підтриманий взаємодією з іншими користувачами. З іншого боку, жодна з платформ не забезпечує повноцінної комбінації таких характеристик, як тематична гнучкість, локалізація, структурованість, ефективна модерація, а також створення простору для сталого зростання знань користувачів. Це відкриває можливість для створення нової цифрової екосистеми, в межах якої буде реалізовано як інструменти для активної участі — стрічка новин, хаби за інтересами, повідомлення — так і механізми зниження ризиків — автентифікація, прозора модерація, освітній контент, орієнтований на українську фінансову реальність.

Метою розробки є не просте відтворення існуючих функцій, а створення якісно нового середовища, де користувачі зможуть не лише ділитися думками, а й навчатися, аналізувати, планувати та формувати відповідальні фінансові стратегії. Така платформа покликана стати містком між швидким інформаційним потоком сучасних соціальних мереж і потребою в глибшому, критичному та осмисленому ставленні до фінансів як особистої, так і суспільної теми.

1.4. Визначення функціональних вимог до системи

Після проведення аналізу особливостей тематичних соціальних мереж і розгляду прикладів наявних рішень стає можливим формулювання загальних функціональних вимог до розроблюваної інформаційної системи. Враховуючи специфіку предметної області, а також потребу в забезпеченні інтуїтивно зрозумілого та ефективного інтерфейсу для взаємодії користувачів, визначення таких вимог є ключовим етапом проєктування майбутнього вебзастосунку.

Функціональність системи повинна охоплювати весь цикл користувацької взаємодії — від початкового входу на платформу до створення, перегляду й обговорення контенту, що публікується учасниками спільноти. Першочерговою є можливість реєстрації та авторизації користувачів із підтримкою як базового механізму на основі логіна й пароля, так і сторонньої автентифікації через зовнішні сервіси. Після входу користувач повинен мати доступ до персоналізованого профілю, який відображає його активність, підписки, власні публікації та інші пов’язані дані.

Основною одиницею взаємодії в системі є публікація — контент, що містить текстові матеріали, аналітику або дописи з елементами розважального чи публіцистичного характеру. Користувачі повинні мати можливість створювати такі публікації з використанням розширеного текстового редактора, який підтримує базове форматування, вставлення зображень і забезпечує очистку HTML-коду від зайвих елементів для збереження узгодженого стилю контенту. Крім того, платформа

має забезпечувати відображення стрічки публікацій як загального характеру, так і на основі підписок користувача, із можливістю сортування та пагінації для покращення продуктивності й зручності перегляду.

Комунікаційна складова є важливим компонентом проєкту. Вона передбачає реалізацію системи коментарів, що підтримує вкладені відповіді, а також можливість надсилання приватних повідомлень між користувачами. Коментарі повинні оновлюватися динамічно, забезпечуючи безперервний досвід обговорення, а можливість їх видалення має бути обмежена лише до автора або адміністратора контенту. Платформа також повинна надсилати сповіщення про взаємодії, що стосуються користувача, як-от нові підписники, відповіді в коментарях або прийняття до хабу, до якого було подано запит.

Окрему увагу слід приділити функціональності тематичних хабів, що відіграють роль структурованих груп за інтересами. Користувач повинен мати змогу приєднуватися до таких спільнот, надсилати запити на вступ, а адміністратори — модерувати склад учасників і контент, що публікується в межах хабу. Такий підхід дозволяє підтримувати тематичну спрямованість платформи та підвищувати якість обговорень.

Усі вищезгадані можливості мають бути реалізовані з дотриманням принципів інформаційної безпеки, стійкості до навантажень та розширюваності. Система повинна підтримувати масштабування як на рівні даних, так і на рівні сервісів, що є критично важливим для потенційного зростання аудиторії. Крім того, застосування сучасних вебтехнологій у клієнтській і серверній частинах має забезпечити швидкий відгук інтерфейсу, мінімізацію затримок при взаємодії з даними та адаптивне відображення для різних типів пристроїв.

Загалом, формалізовані функціональні вимоги виступають основою для подальшого технічного проєктування, дозволяючи чітко визначити рамки розробки, структуру компонентів і необхідні інтерфейси, а також забезпечити відповідність очікуванням кінцевих користувачів і цілям, поставленим у межах даного дослідження.

РОЗДІЛ 2. СИСТЕМНЕ ПРОЄКТУВАННЯ ТА ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ

2.1. Вибір технологічного стеку та засобів розробки

Під час створення веборієнтованої інформаційної системи — соціальної мережі, орієнтованої на тематику фінансів та інвестування — постала необхідність обрати такі інструменти та технології, які б одночасно забезпечували масштабованість, безпеку, високу продуктивність, зручну інтеграцію між клієнтською та серверною частинами, а також були б придатними для подальшого розгортання й підтримки системи. Оскільки мова йде про платформу, яка потенційно має обслуговувати велику кількість користувачів, активно обмінюватися даними, підтримувати складні зв’язки між об’єктами та інтегрувати збереження файлів, вибір стеку мав бути обґрунтованим і стратегічним.

Клієнтську частину застосунку реалізовано з використанням Angular — потужного фронтенд-фреймворку, який забезпечує строго структуровану архітектуру, модульність та широкий набір інструментів «з коробки» [11]. На відміну від бібліотек типу React або Vue, Angular пропонує повноцінне рішення для побудови масштабованих односторінкових застосунків, включаючи маршрутизацію, інжекцію залежностей, систему форм та інші засоби, що спрощують розробку і підтримку проєкту [13]. Саме ця особливість дозволила з самого початку впровадити модульну організацію застосунку, що у подальшому суттєво спростило його масштабування.

Використання TypeScript дозволило реалізувати статичну типізацію, інкапсуляцію логіки в окремих компонентах, а також полегшити налагодження завдяки компіляторній перевірці. Angular також надає інструменти для реалізації захисту сторінок, реактивних форм, перехоплення HTTP-запитів — усе це сприяло ефективній реалізації ключового функціоналу соціальної мережі: автентифікації,

персоналізованої стрічки публікацій, навігації між сторінками хабів, повідомленнями та профілями.

Застосування Angular CLI стало додатковою перевагою, оскільки значно прискорило розробку завдяки автоматизації створення компонентів та перевірці коду [12] .

У серверній частині використовувався фреймворк ASP.NET Core Web API — сучасна платформа від Microsoft, яка підтримує кросплатформенність, асинхронність та високу продуктивність [14]. Основною причиною вибору саме цього фреймворку стала можливість побудови чітко структурованої логіки на основі шаблону Clean Architecture [15], який дозволяє відокремити бізнес-логіку від інфраструктурного коду. Це забезпечило кращу підтримуваність та тестованість системи.

Завдяки впровадженню бібліотеки MediatR була реалізована шаблонна архітектура на основі CQRS, яка дозволяє чітко розмежувати запити та команди, що виявилося зручним для складних сценаріїв взаємодії з даними [16]. AutoMapper, своєю чергою, спростив процес перетворення DTO в доменні сутності, що знизило ризики помилок при передачі даних між шарами [17]. Усе це дозволило побудувати захищене API з підтримкою авторизації через JWT-токени, де кожен запит від користувача перевіряється та обмежується за правами доступу.

Для зберігання даних було обрано PostgreSQL — потужну об’єктно-реляційну систему управління базами даних з відкритим кодом [18]. На відміну від альтернатив, таких як MySQL або SQLite, саме PostgreSQL найбільше відповідає вимогам системи, яка має численні зв’язки між сутностями, що ускладнює структуру таблиць. Підтримка складних SQL-запитів, транзакційності (ACID), індексів, зовнішніх ключів, а також можливість працювати з JSON-даними забезпечили гнучкість і високу швидкодію. Особливо важливою стала сумісність PostgreSQL з ORM-фреймворком Entity Framework Core, який використовувався в середовищі .NET [19], що забезпечило прозору роботу з моделями даних, зменшило обсяг «ручного» SQL-коду і підвищило продуктивність розробки.

Контейнеризація проєкту була реалізована за допомогою Docker [20]. Його використання дозволило створити однакове середовище для розробки, тестування та розгортання. Завдяки Docker середовище розгортання стало повністю ізольованим і незалежним від локальних налаштувань, що усунуло типові помилки на кшталт «на моїй машині працює». Крім того, контейнеризація дозволила стандартизувати процес CI/CD та значно спростила масштабування системи у staging- та production-оточеннях.

З огляду на необхідність зберігання зображень, токенів доступу та інших конфіденційних даних, було прийнято рішення використовувати сервіси Microsoft Azure. Зокрема, Blob Storage став основним сховищем для медіафайлів, таких як зображення у постах, а Key Vault — для безпечного зберігання ключів доступу [21], секретів для JWT та ряжків підключення. Такий підхід дозволив дотриматися принципу найменших привілеїв та розділити логіку застосунку і зберігання файлів, що позитивно позначилося на безпеці.

У сукупності, обраний технологічний стек є результатом не лише прагнення до сучасності, а й відповіддю на конкретні вимоги до архітектури, продуктивності, гнучкості, безпеки та підтримуваності системи. Цей набір інструментів забезпечує надійну основу для масштабованої, функціонально насиченої соціальної платформи, здатної до подальшого розвитку.

2.2. Архітектура клієнт-серверної взаємодії інформаційної системи

Інформаційна система побудована на основі клієнт-серверної архітектури, яка забезпечує чітке розділення обов’язків між фронтендом та бекендом. Такий підхід дозволяє досягнути високої масштабованості, спрощує супровід коду, а також сприяє незалежному розвитку обох частин системи. Комунікація між клієнтом і сервером здійснюється через RESTful API [22] з використанням HTTP-запитів, що дає змогу чітко контролювати всі точки взаємодії.

Серверна частина реалізована на платформі ASP.NET Core Web API та побудована відповідно до принципів чистої архітектури, що передбачає чітке розділення логіки додатку на ізольовані шари з урахуванням напрямку залежностей. Уся структура серверу розбита на чотири основні шари: API, Application, Domain, Infrastructure.

У таблиці 2.1 окреслено основні архітектурні шари серверної частини інформаційної системи.

Таблиця 2.1

Структура серверної архітектури

Архітектурний шар	**Відповідальність**	**Особливості / Технології**
API	Прийом HTTP-запитів, маршрутизація, базова валідація	Делегування логіки внутрішнім сервісам, контролери
Application	Прикладна логіка, CQRS: команди й запити	MediatR, хендлери, підтримка логування й валідації
Domain	Бізнес-логіка, доменні моделі	Незалежність від технологій, чистий код
Infrastructure	Зовнішні залежності, доступ до БД та сервісів	EF Core, PostgreSQL, Azure Blob Storage

Джерело: розроблено автором

Кожен модуль системи ізольовано — для обробки постів, коментарів, сповіщень, повідомлень тощо реалізовано окремі хендлери та сервіси. Доступ до даних здійснюється через репозиторії, а мапінг між доменними моделями та DTO забезпечується бібліотекою AutoMapper. Це дає змогу уникати дублювання логіки перетворення даних і покращує підтримуваність коду.

Для забезпечення масштабованості та ефективного використання ресурсів реалізовано серверну пагінацію, сортування і фільтрацію. Наприклад, для перегляду стрічки постів клієнт надсилає запит із параметрами сторінки (номер і розмір), а сервер обробляє його за допомогою Skip() та Take() в LINQ-запитах, що дозволяє повертати тільки потрібну порцію даних.

Аутентифікація реалізована за допомогою JWT: після входу користувача сервер генерує токен, який клієнт зберігає у localStorage і прикріплює до заголовків кожного запиту [23]. Сервер перевіряє його дійсність через механізм middleware і атрибут [Authorize], забезпечуючи захист приватних ресурсів.

Оновлення сповіщень, повідомлень або інтерфейсних станів відбувається за допомогою періодичних запитів з клієнта до відповідних кінцевих точок API. Хоча WebSocket не використовується, обрана модель наразі задовольняє функціональні потреби, оскільки дозволяє контролювати частоту запитів і масштабування запитів до сервера.

Клієнтська частина реалізована за допомогою Angular, сучасного фронтенд-фреймворку, який надає потужні засоби для побудови реактивних, компонентно-орієнтованих веб-застосунків. Архітектура клієнтської частини побудована за модульною структурою, що забезпечує логічне розділення коду, високу читабельність і легкість тестування.

У таблиці 2.2 окреслено основні архітектурні шари клієнтської частини інформаційної системи.

Таблиця 2.2

Структура Angular-фронтенду

Шар	**Опис**	**Приклад функціоналу**
Core	Містить сервіси, глобальні конфігурації, хедери, механізми автентифікації.	Сервіс авторизації, що зчитує токен із localStorage, методи для входу та реєстрації
Shared	Включає повторно використовувані компоненти, пайпи, директиви, Guard-и, Interceptor-и.	Загальний компонент повідомлення про помилку, пайп форматування дати.
Features	Основні функціональні модулі: управління акаунтом, перегляд/створення постів, коментарі, повідомлення тощо.	Кожен модуль ізольований, має власний маршрутизатор, сторінки та сервіси для обробки бізнес-логіки.

Джерело: розроблено автором

Angular CLI використовується для генерації нових модулів, компонентів і сервісів [13], а також для зборки й деплойменту проєкту. Це дозволяє дотримуватися єдиної структури та полегшує роботу над великими проєктами.

Запити до серверу організовані через Angular HTTP-сервіси, що інкапсулюють логіку взаємодії з API. Кожен сервіс реалізує CRUD-операції для конкретного ресурсу. Наприклад, PostService відповідає за створення, редагування, отримання та видалення постів.

Фронтенд також працює з локальним станом: збереження вибраних вкладок, відкритих діалогів, фільтрів. Це дозволяє зменшити кількість запитів до сервера і покращити UX. Для управління складнішим станом використана реактивна модель через RxJS або сигнал-орієнтований підхід, підтримуваний Angular Signal Store.

Захист маршрутів реалізовано за допомогою AuthGuard, який перевіряє наявність токена перед переходом до захищених сторінок. Якщо токен недійсний або відсутній, користувач автоматично перенаправляється на сторінку входу.

Компонентний підхід Angular дозволяє ізолювати логіку відображення, повторно використовувати інтерфейсні елементи та впроваджувати lazy loading для підвищення продуктивності. Завдяки чіткій структурі додатку його легко підтримувати та масштабувати: кожен новий функціонал реалізується як окремий модуль, що знижує ризик регресій.

Завдяки поєднанню сучасної серверної архітектури на базі ASP.NET Core та ефективної клієнтської частини на Angular система має високу надійність, продуктивність і готовність до масштабування. Логічний поділ на модулі та шари сприяє підтримуваності, а впроваджені практики дозволяють адаптувати систему до змін бізнес-вимог без значного рефакторингу.

2.3. Інформаційна модель системи та структура даних

Інформаційна модель системи відображає структуроване подання основних об’єктів предметної області соціальної мережі, необхідних для забезпечення її

функціональності. Центральними сутностями моделі є користувачі, пости, коментарі, хаби, повідомлення та сповіщення, кожна з яких має унікальний ідентифікатор і набір властивостей, що описують їхні характеристики [24].

Взаємозв’язки між сутностями реалізовані через відповідні типи зв’язків:

- Користувачі є авторами постів і коментарів, надсилають повідомлення та можуть підписуватися на інших користувачів і хаби (зв’язки "один-до-багатьох" та "багато-до-багатьох").
- Пости прив’язані до конкретних авторів та, за потреби, до одного або кількох хабів, які організовують контент за темами. Також містити кілька категорій, зображень та взаємодій у вигляді лайків і коментарів [24]
- Коментарі формують ієрархічну структуру відповідей на пости або інші коментарі, підтримуючи вкладені дискусії.
- Хаби реалізовані як спільноти за інтересами. Вступ до деяких хабів може вимагати погодження заявки, для чого існує додаткова сутність [26].
- Повідомлення пов’язують відправника й отримувача, зберігаючи інформацію про вміст, час надсилання і статус прочитання, а також підтримують логіку м’якого видалення.
- Сповіщення зберігають інформацію про події, пов’язані з діями користувачів (наприклад, лайки, підписки, запити на вступ до хабів), включаючи статус їх прочитання [25].

Для підписок застосовується універсальна сутність, що дозволяє гнучко встановлювати зв’язки користувачів із іншими користувачами чи хабами.

Архітектура бази даних побудована на реляційній моделі з чітким поділом відповідальностей між сутностями та підтримкою зовнішніх ключів, що забезпечує цілісність і узгодженість даних. Така структура дозволяє ефективно масштабувати систему, додавати нові функціональні можливості та забезпечувати швидкий доступ до необхідної інформації.

Всі описані сутності та їх атрибути, а також детальна схема бази даних наведені у Додатку А.

Ця інформаційна модель слугує основою для реалізації користувацьких сценаріїв та бізнес-логіки системи, що детально розглядаються у наступному розділі.

2.4. Користувацькі сценарії функціонування системи

Користувацькі сценарії визначають типові дії, які може виконувати користувач у межах інформаційної системи. Вони відображають функціональні можливості програмного продукту з точки зору кінцевого користувача та сприяють розумінню логіки взаємодії з інтерфейсом. В контексті розробленої фінансово-інвестиційної соціальної мережі, низка основних сценаріїв реалізована для забезпечення повноцінної участі користувачів у системі, включно з реєстрацією, створенням контенту, взаємодією із публікаціями, управлінням спільнотами та комунікацією.

Першим та базовим сценарієм є процес реєстрації нового користувача.

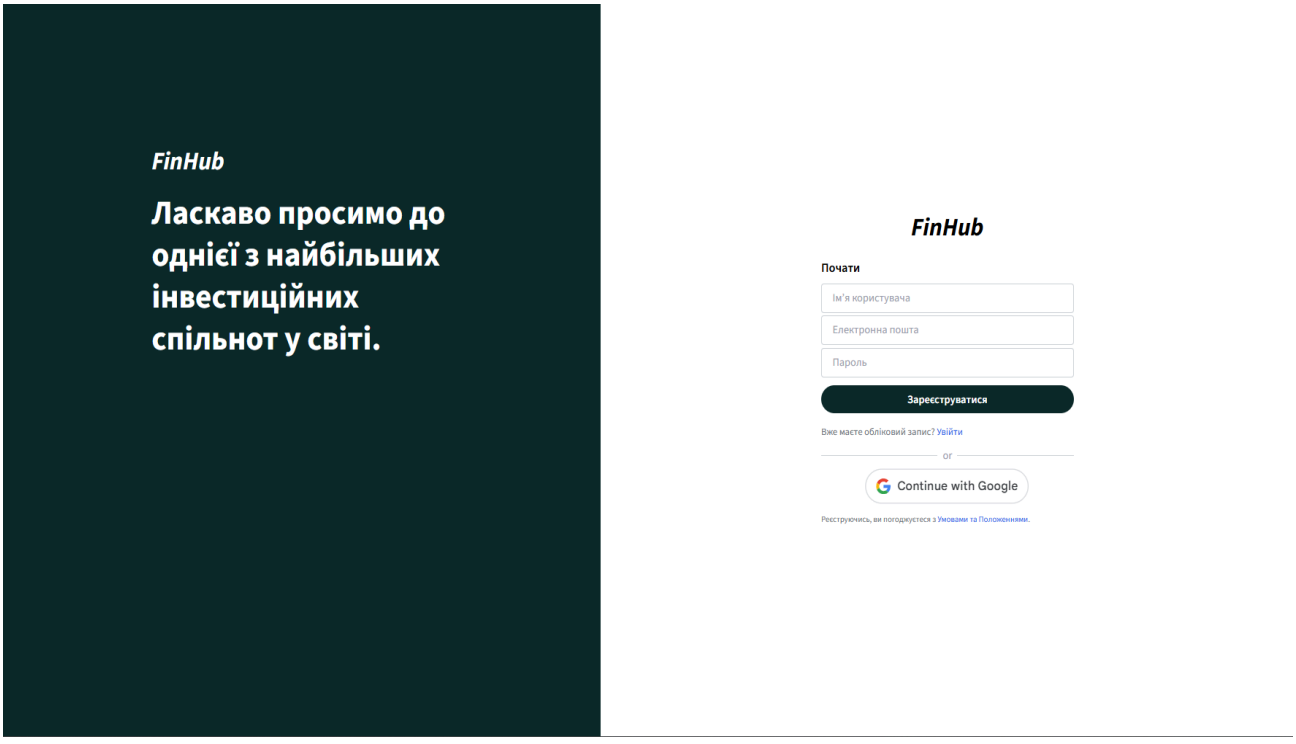

Рис. 2.1. Сторінка для реєстрації користувача.

Джерело: розроблено автором

Він полягає у заповненні користувачем форми з мінімально необхідними даними, серед яких електронна пошта, ім’я користувача та пароль. Після успішної

валідації цих даних система створює обліковий запис, зберігає інформацію у базі даних та генерує JWT-токен для подальшої авторизації користувача [27]. Цей токен забезпечує безпечний доступ до захищених ресурсів системи. Внаслідок реєстрації користувач отримує автоматичний доступ до персонального кабінету, що відкриває подальші можливості взаємодії.

Подальшим важливим сценарієм є створення контенту — додавання публікацій (постів) або створення тематичних спільнот — хабів. Для створення поста користувач повинен бути авторизованим. Він вводить текстовий вміст, а також може прикріпити зображення. Перед відправленням на сервер система перевіряє коректність введених даних (відсутність порожніх полів), після чого клієнтська частина відправляє дані відповідно до REST-архітектурного підходу, що забезпечує масштабованість і незалежність клієнтського інтерфейсу від серверної логіки [29]. Опубліковані пости стають доступними для перегляду іншими користувачами у загальній стрічці.

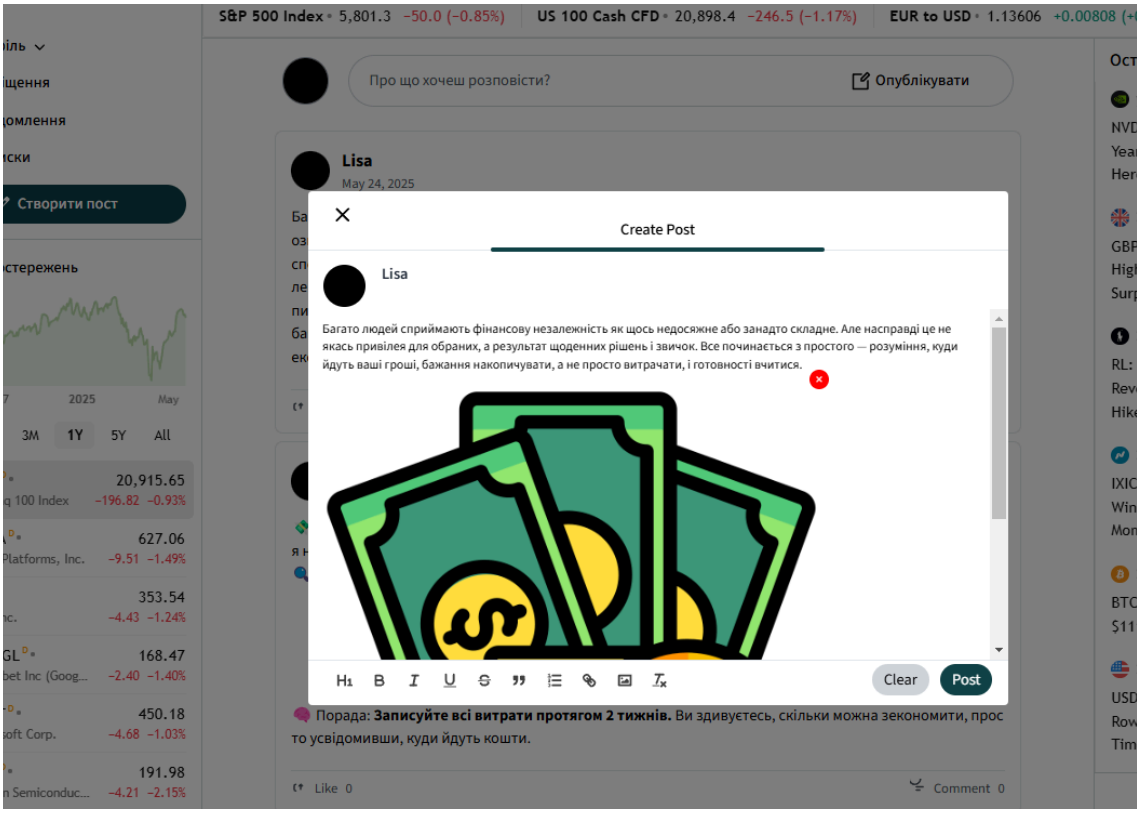

Рис. 2.2. Створення поста.

Джерело: розроблено автором

Користувачі мають можливість переглядати пости інших учасників, залишати коментарі, ставити вподобання, а також відстежувати активність певних авторів чи хабів. Перехід до профілів інших користувачів дозволяє ознайомитися з їхньою активністю: публікаціями, вподобаннями та участю у хабах.

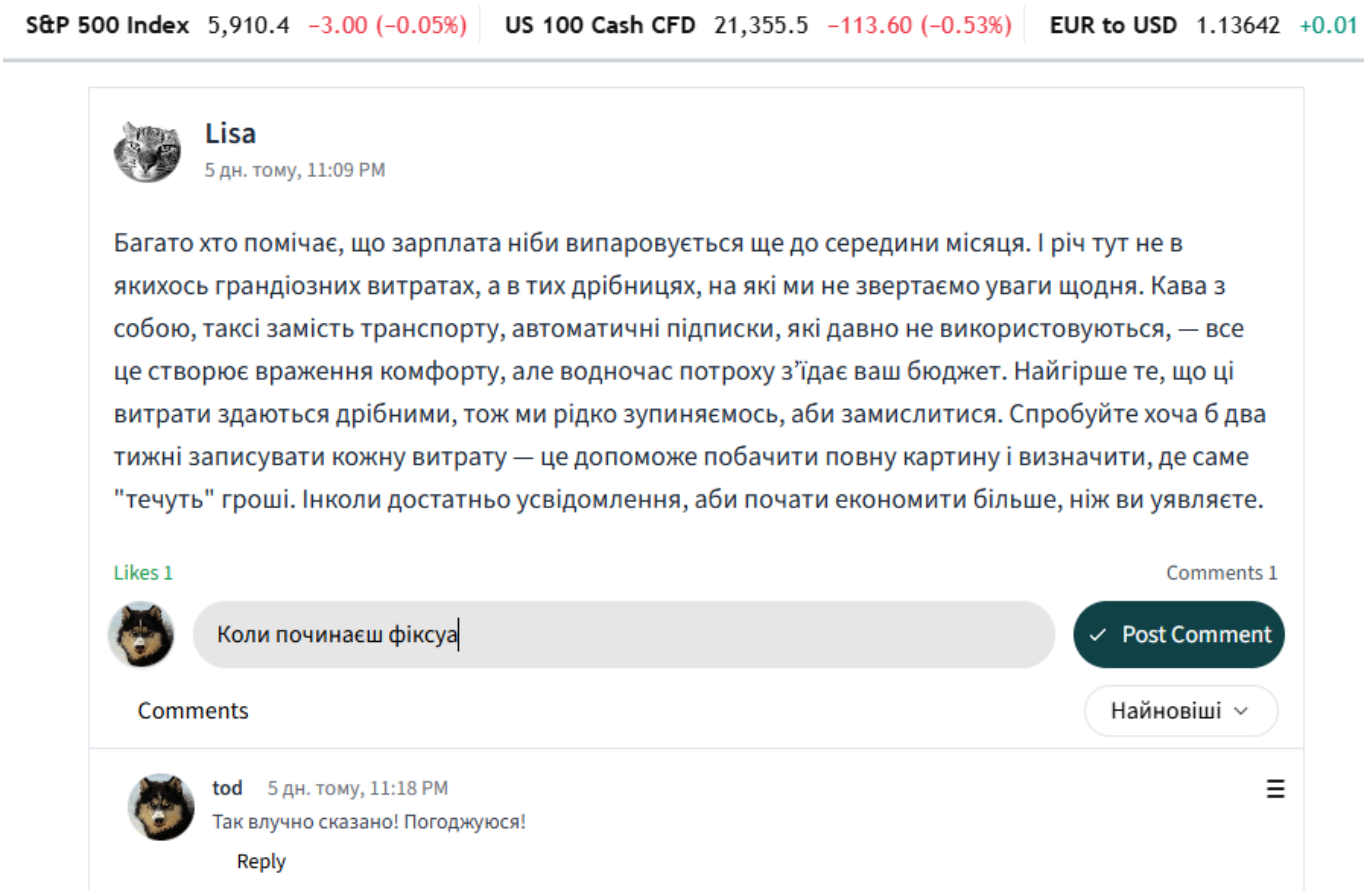

Рис. 2.3. Написання коментаря.

Джерело: розроблено автором

Особливою функціональністю системи є механізм вступу до хабів — тематичних спільнот, які об'єднують користувачів за інтересами в сфері інвестицій, фінансів, економіки тощо. Для вступу користувач може подати відповідний запит, який набуває чинності лише після схвалення адміністратором хабу. Всередині хабу користувачі отримують доступ до публікацій, сповіщень та спеціальних розділів, які недоступні для сторонніх.

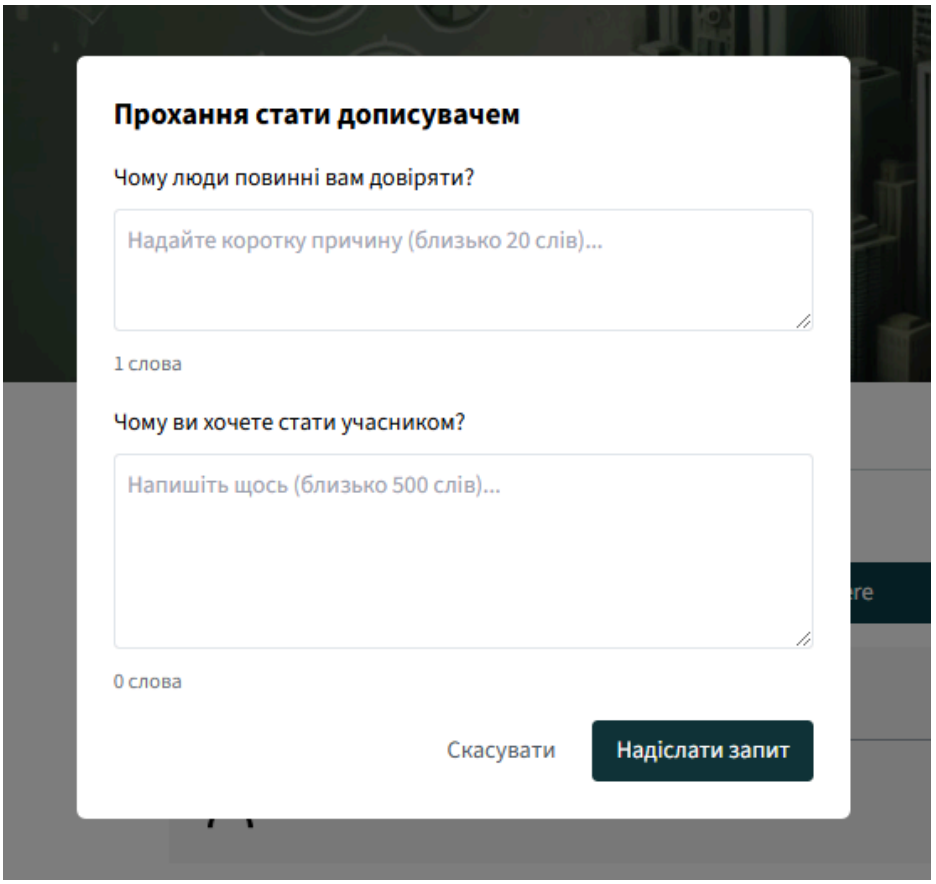

Рис. 2.4. Форма для приєднання до хабу.

Джерело: розроблено автором

Ще одним важливим і поширеним сценарієм взаємодії користувачів у системі є обмін приватними повідомленнями. Реалізація функціоналу особистого листування дозволяє користувачам спілкуватися між собою у зручному форматі діалогів. Зокрема, система надає можливість ініціювати нові розмови, надсилати та отримувати відповіді, а також переглядати повну історію попереднього листування. Кожне повідомлення, що надсилається, обробляється через серверний API, забезпечуючи надійність передачі даних між клієнтською і серверною частинами застосунку. Усі повідомлення зберігаються в базі даних із чітким відображенням, хто саме був ініціатором повідомлення та хто його отримав — для цього використовується прив’язка до унікальних ідентифікаторів обох сторін комунікації [28]. Такий підхід дозволяє забезпечити структуроване зберігання інформації та дає змогу легко відтворювати логіку діалогу при необхідності.

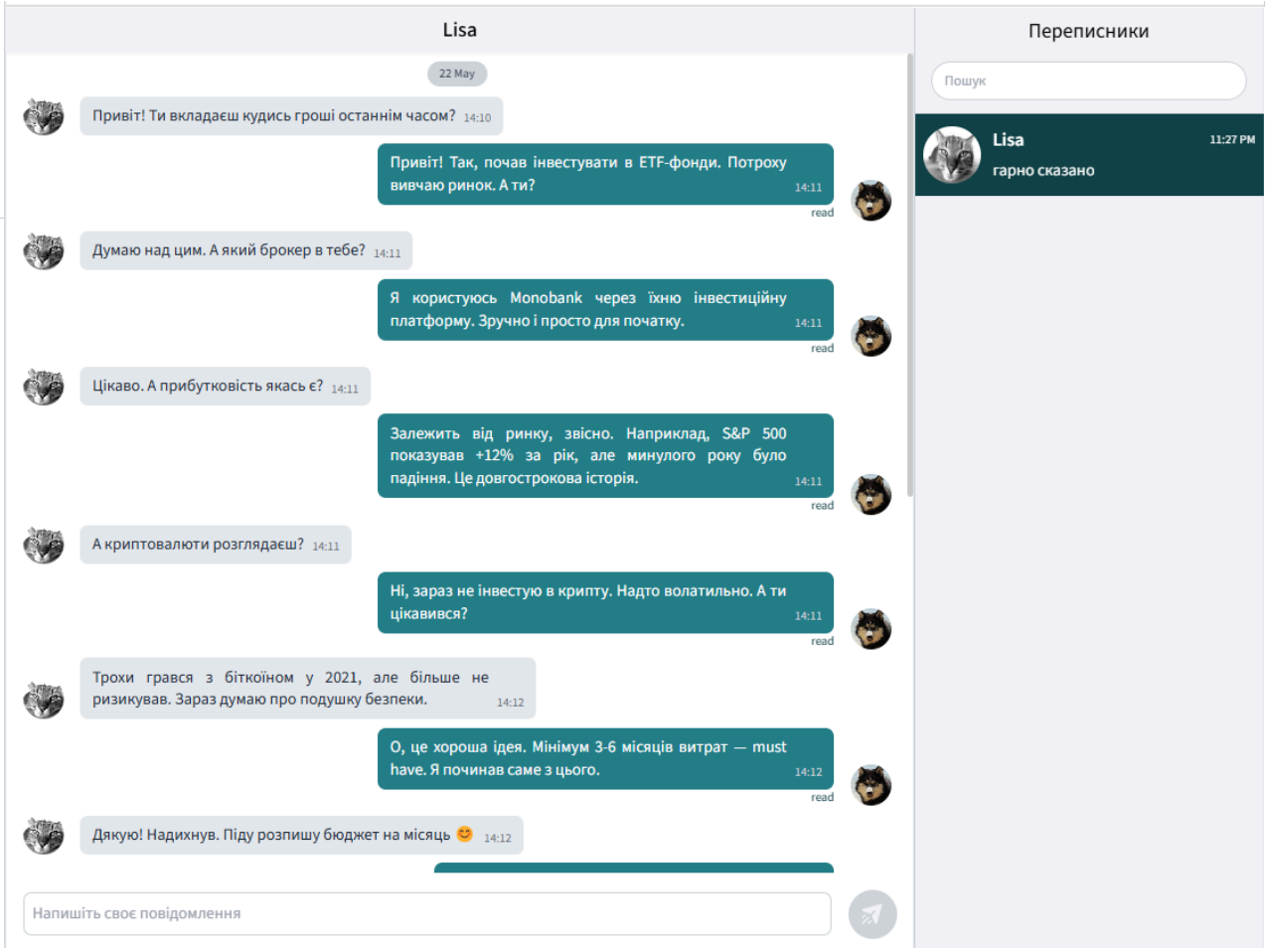

Рис. 2.5. Чат листування між користувачами.

Джерело: розроблено автором

У системі реалізовано механізм сповіщень, який відіграє ключову роль у забезпеченні зручного користувацького досвіду. Усі значущі події, що відбуваються під час взаємодії користувача з платформою або в результаті дій інших користувачів, фіксуються й обробляються через спеціальну систему сповіщень. Серед прикладів таких подій — отримання вподобання (лайку) на публікацію, надходження запрошення або підтвердження вступу до хабу, а також початок стеження за профілем іншого користувача.

Кожне сповіщення містить короткий опис події та інтерактивне посилання на об’єкт, до якого вона стосується, що дозволяє миттєво перейти до відповідного елементу інтерфейсу або сторінки. Крім того, сповіщення мають статус прочитання, що забезпечує можливість розмежування між новими та вже переглянутими подіями, підвищуючи ефективність орієнтації в інформаційному потоці.

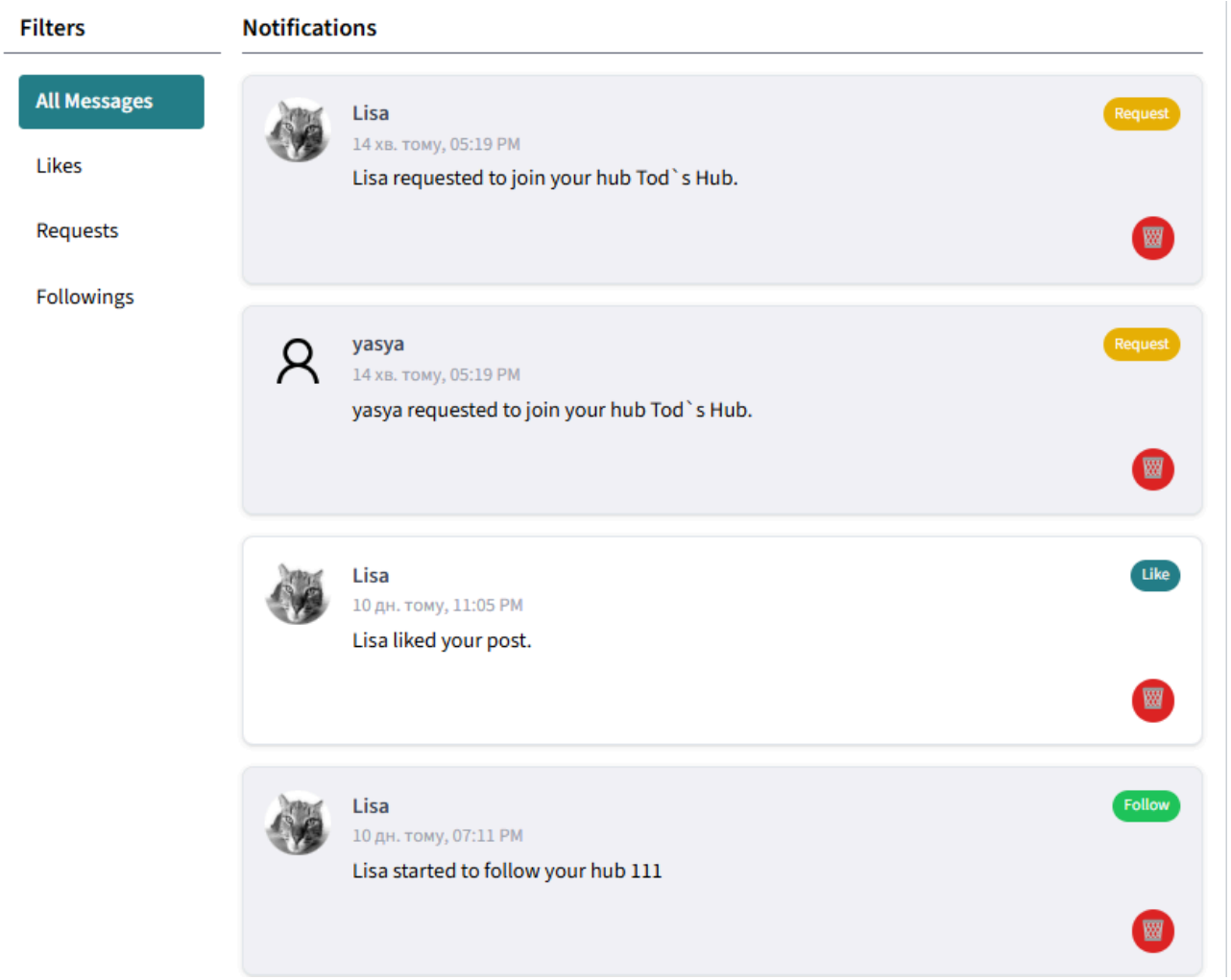

Рис. 2.6. Панель сповіщень.

Джерело: розроблено автором

Система передбачає можливість персоналізації та керування власним профілем для кожного авторизованого користувача. Зокрема, він може редагувати свій профіль, змінюючи аватар (зображення профілю), оновлюючи опис, що його характеризує, а також редагуючи своє відображуване ім’я, яке бачать інші учасники платформи. Крім цього, користувачу надається право самостійно керувати своїм контентом — наприклад, за потреби можна видалити створений раніше пост або вийти з певного хабу, в якому він більше не хоче брати участь.

Усі дії, що мають наслідки для внутрішнього стану системи або змінюють дані користувача, дозволені виключно тим, хто пройшов автентифікацію, тобто авторизованим користувачам. У той же час для незареєстрованих відвідувачів доступ обмежується лише переглядом загальнодоступного контенту без можливості взаємодії чи внесення будь-яких змін у систему.

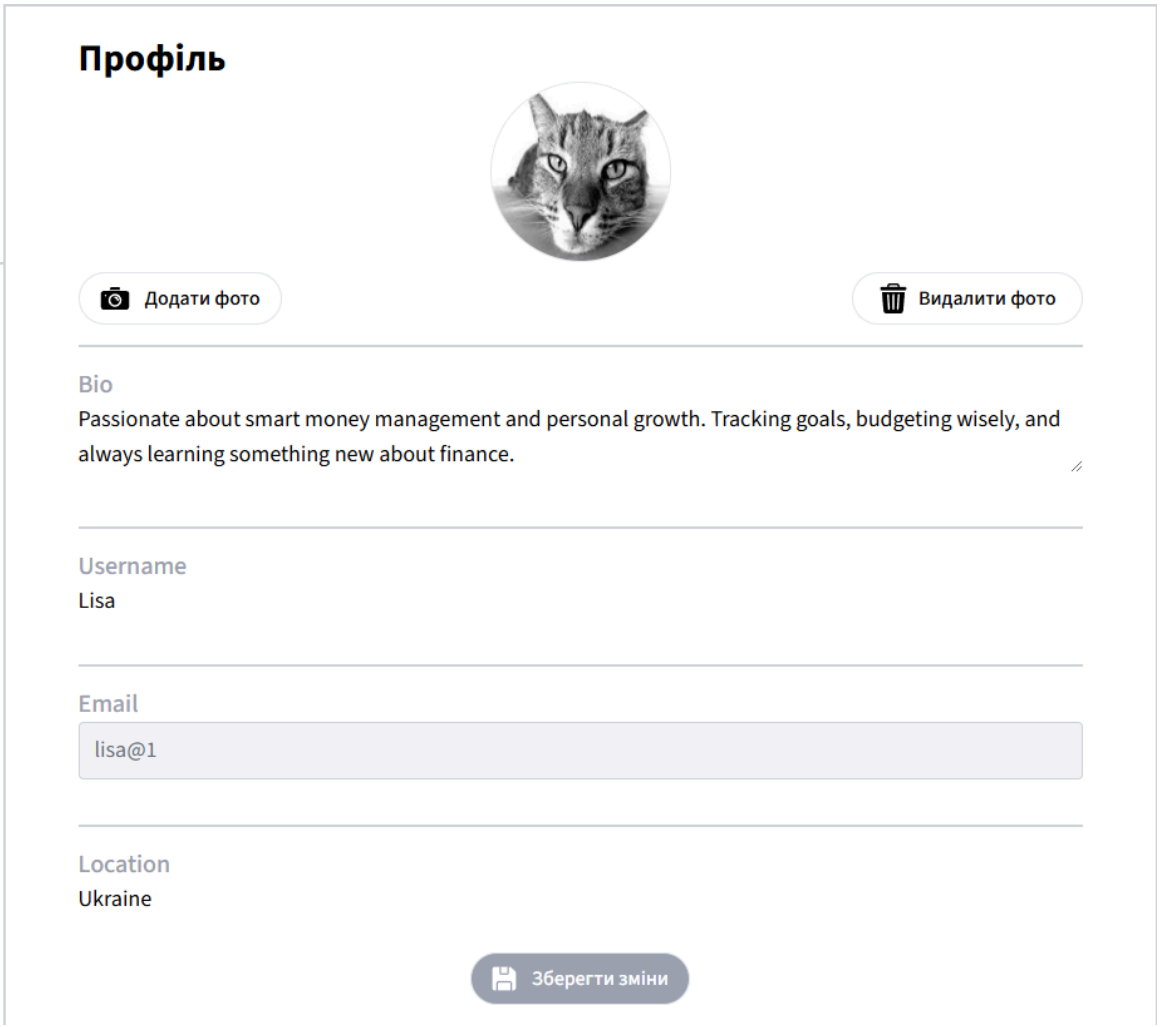

Рис. 2.6. Панель редагування профілю користувача.

Джерело: розроблено автором

Структуроване представлення описаних користувацьких сценаріїв, із зазначенням їх призначення, вимог до авторизації та очікуваних результатів, наведено у Додатку Б.

2.5. Відповідність архітектури рівням моделі OSI

Модель OSI (Open Systems Interconnection) є загальновизнаною теоретичною основою для опису взаємодії мережевих систем. Вона складається з семи рівнів, кожен з яких виконує окрему функцію у процесі передачі даних. У контексті розробленої системи (вебзастосунку на Angular, .NET Web API, PostgreSQL та Azure Blob Storage) відповідність архітектурних компонентів цим рівням можна охарактеризувати наступним чином.

Фізичний та канальний рівні (Physical, Data Link). Ці рівні реалізуються інфраструктурою хмарного середовища — у нашому випадку це платформа Microsoft Azure, яка забезпечує віртуальні машини, мережеві адаптери, комутатори та фізичні канали передачі даних. Azure відповідає за маршрутизацію, балансування навантаження, а також фізичну надійність зберігання (диски, оптоволокно тощо) [30]. У нашому проєкті зберігання зображень і ключів доступу здійснюється в Azure Blob Storage, який працює поверх цих рівнів.

Мережевий рівень (Network).На цьому рівні функціонує IP-маршрутизація та забезпечується адресація. З боку Azure це реалізовано через віртуальні мережі (VNet), NAT, міжмережеві екрани, а також автоматизоване керування IP-адресами. У випадку деплойменту бекенду в Docker-контейнерах — мережеві драйвери Docker Bridge чи Overlay також оперують на цьому рівні [31].

Транспортний рівень (Transport). Використання протоколу TCP є основою для HTTP-комунікації між Angular-клієнтом і .NET API. Це забезпечує гарантовану доставку, контроль потоку та повторну передачу при втраті пакетів. Усі запити до RESTful API та responses між клієнтом і сервером проходять через цей рівень .

Сеансовий рівень (Session). Рівень сесії у вебзастосунках зазвичай реалізується через механізми аутентифікації, керування сесією та встановлення з’єднань. У нашій системі аутентифікація реалізована через JWT-токени, які передаються клієнтом у заголовках HTTP-запитів. Відповідність сеансового рівня забезпечується middleware-системою ASP.NET (наприклад, AuthenticationHandler), яка керує автентифікацією, сесіями та авторизацією [32].

Рівень представлення (Presentation). Цей рівень відповідає за форматування, кодування та шифрування даних. У нашій системі це реалізується шляхом:

- серіалізації/десеріалізації JSON-повідомлень (через System.Text.Json на бекенді та HttpClient/RxJS map() на фронтенді)
- SSL/TLS-шифруванням при передачі даних (HTTPS-з'єднання)
- підтримкою MIME-типів для обробки зображень (у контенті постів) [33].

Прикладний рівень (Application). На цьому рівні функціонують Angular-клієнт, RESTful API, Web UI, повідомлення, стрічка постів, підписки тощо. Angular виконує

роль представлення інтерфейсу, а .NET Web API — роль бізнес-логіки та доступу до бази даних PostgreSQL. Цей рівень відповідає за повну взаємодію користувача з системою через браузер та API-запити .

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОГРАМНО-ТЕХНІЧНОГО ЗАБЕЗПЕЧЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1. Реалізація функціональних компонентів системи

3.1.1. Реєстрація та автентифікація користувачів

У проєкті реалізовано механізм автентифікації на основі JWT, що забезпечує безпечну, масштабовану та REST-сумісну модель управління сесією користувача. Такий підхід відповідає сучасним вимогам до безпеки у вебзастосунках і дозволяє централізовано контролювати доступ до API через передачу токенів у HTTP-запитах.

Архітектура реалізації автентифікації інтегрована у серверну частину,. Основним компонентом виступає сервіс JwtService, відповідальний за генерацію JWT токена на основі даних користувача [34]. Для формування токена використовується бібліотека System.IdentityModel.Tokens.Jwt, а також механізм SigningCredentials з симетричним ключем, отриманим з конфігурації. У тілі токена вбудовуються claims, зокрема ідентифікатор користувача, email, а також ролі, що витягуються через UserManager<User> з Microsoft Identity.

Лістинг коду 3.1. Claims під час генерації токена.

```
var claims = new List<Claim>
{
    new(ClaimTypes.NameIdentifier, user.Id.ToString()),
    new(ClaimTypes.Email, user.Email)
};
var roles = await userManager.GetRolesAsync(user);
claims.AddRange(roles.Select(role => new Claim(ClaimTypes.Role, role)));
```

Цей токен формується лише після підтвердження достовірності облікових даних через відповідні команди MediatR (наприклад, GetUserByCredentialsQuery), що реалізують шаблон CQRS. У запиті використовується репозиторій IUserRepository для взаємодії з базою користувачів, де перевіряється наявність користувача за email та відповідність пароля.

У разі успішної аутентифікації токен генерується окремою командою MediatR (GenerateTokenCommand), після чого API повертає DTO користувача (GetUserDto) разом із JWT. Клієнтський застосунок Angular зберігає цей токен у localStorage. Для доступу до захищених маршрутів клієнт додає токен у заголовок Authorization, а на стороні API доступ обмежується через атрибути [Authorize].

На клієнтському рівні токен, отриманий після успішної автентифікації, зберігається у localStorage та вручну додається до заголовків під час здійснення запитів до захищених маршрутів API. Поточна реалізація забезпечує коректну перевірку автентифікації на стороні сервера, а навігація у застосунку обмежується логікою, що базується на наявності токена та ролі користувача, що зчитується з його даних. Завдяки цьому користувач отримує доступ лише до дозволених функцій, а додаток поводиться стабільно та безпечно.

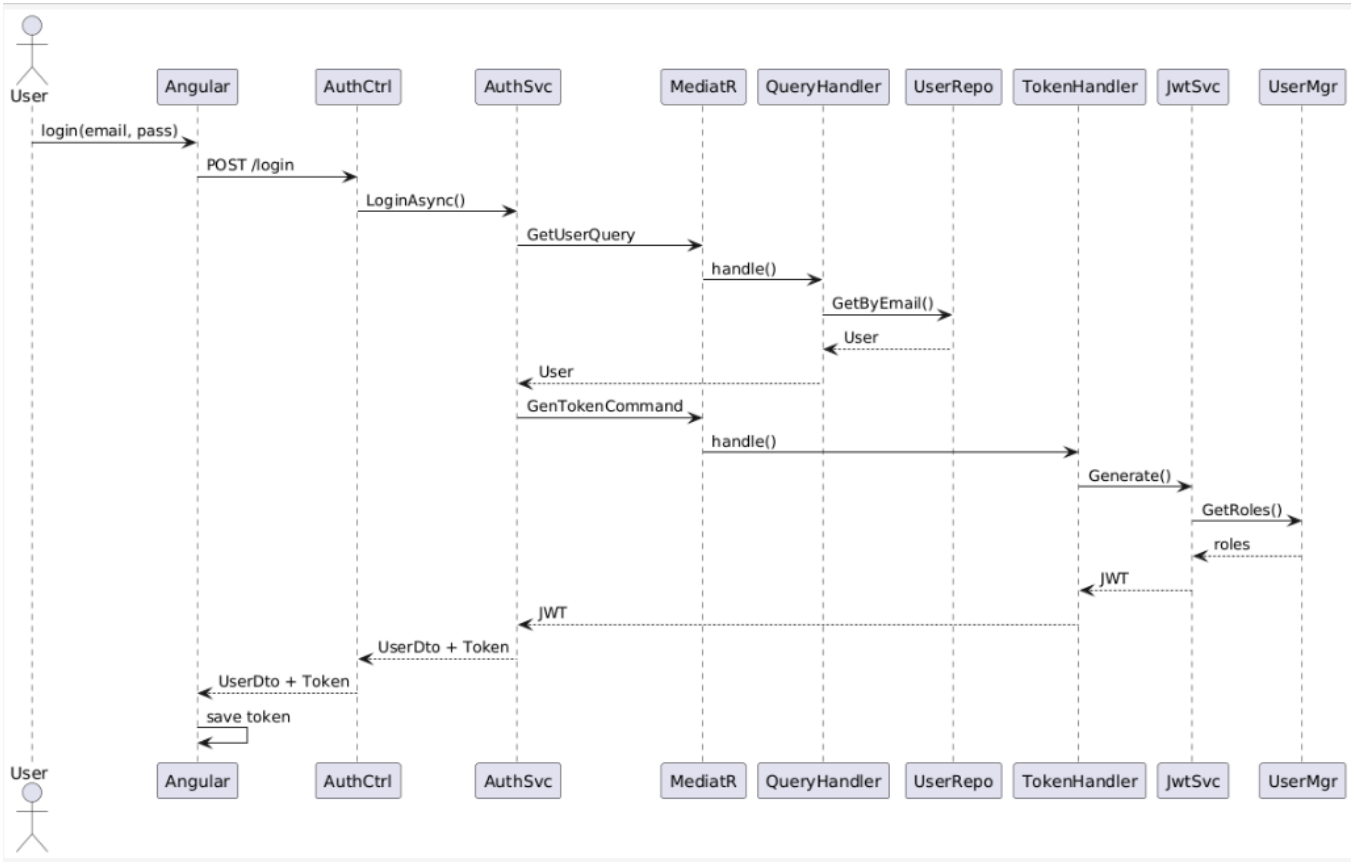

Рис. 3.1. Послідовність дій під час автентифікації користувача

Джерело: розроблено автором

Процес реєстрації реалізовано окремо, з відповідним збереженням користувача в базі даних. У разі некоректних облікових даних система надає зворотний зв’язок

про помилки — наприклад, неіснуючий email або неправильний пароль — через стандартні механізми обробки виключень і передачу повідомлень до клієнта у форматі JSON.

Цей підхід до реалізації автентифікації забезпечує [35]:

- повну незалежність від серверної сесії;
- зменшення навантаження на сервер завдяки відсутності зберігання стану;
- гнучке масштабування, оскільки авторизація здійснюється через валідацію токена без потреби доступу до централізованої сесії;
- можливість інтеграції з іншими сервісами або API на основі JWT, що є загальновизнаним стандартом.

3.1.2. Функціонал профілю користувача

Функціонал профілю користувача у системі охоплює два основні аспекти: перегляд та редагування. Він забезпечує користувачам можливість керування особистими даними, а також ознайомлення з профілями інших користувачів у межах дозволених прав доступу.

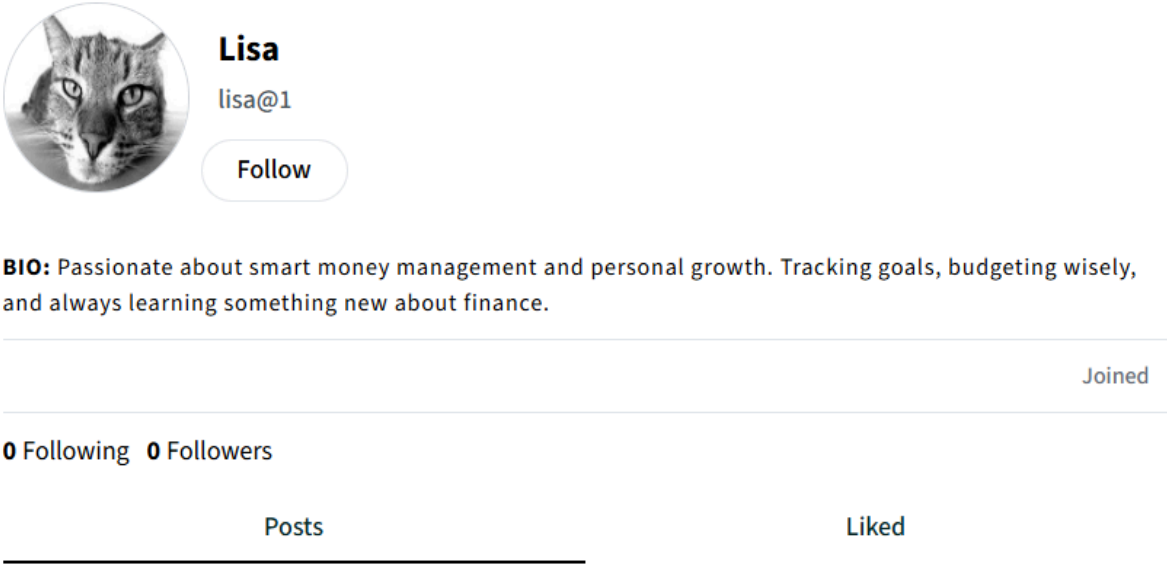

Рис. 3.2 Перегляд профілю іншого користувача

Джерело: розроблено автором

Перегляд профілю реалізований через окремий API-запит до ресурсу */user/by-username/{username}*, який повертає об'єкт типу *GetUserDto*. Цей DTO містить базові ідентифікаційні та профільні поля: унікальний ідентифікатор, електронну адресу, ім'я користувача, роль, біографію, країну, дату створення облікового запису, час останньої активності та URL зображення профілю. Крім того, на клієнтському інтерфейсі додатково відображається кількість підписників, кількість користувачів, на яких підписаний поточний профіль, а також перелік написаних і вподобаних постів. Таке збагачене уявлення дозволяє створити більш цілісний соціальний образ користувача.

Редагування профілю доступне лише автентифікованому власнику облікового запису. Для цього у клієнтській частині реалізовано окрему форму, яка дозволяє змінювати ім'я користувача, країну проживання та біографію. Усі введені значення проходять валідацію перед надсиланням, зокрема перевірку на порожнечу, допустиму довжину та формат символів. Зміни передаються на сервер через HTTP PUT-запит до */user*, із вказанням JWT у заголовку авторизації, що гарантує безпеку операції. Серверна обробка оновлень реалізована у вигляді відповідного обробника, який взаємодіє з UserManager і зберігає зміни до бази даних PostgreSQL через застосування патерна CQRS.

Лістинг коду 3.2. Метод для оновлення користувача на frontend

```
updateMember(user: User) {
        return this.http.put<User>(`${this.baseUrl}/user`, user, {
            headers: new HttpHeaders().set('Authorization', `Bearer
${this.authService.currentUser()?.token}`)
        });
    }
```

Щодо візуалізації аватара користувача, зображення завантажуються за допомогою форми, що працює з об'єктом FormData, і передаються на сервер через окремий ендпоінт */user/add-photo*. Файли зберігаються у хмарному сховищі Azure Blob Storage, URL-адреса збереженого зображення передається в базу даних, а потім додається до профілю користувача, що дозволяє клієнту відображати його як

частину інтерфейсу профілю. Також передбачено можливість видалення фото через запит до */user/delete-photo*.

Лістинг коду 3.3 Метод для видалення фото користувача на backend

```
public async Task DeletePhotoAsync(string imageUrl)
{
   var blobServiceClient = new BlobServiceClient(connectionString);
   var blobContainerClient =
blobServiceClient.GetBlobContainerClient(containerName);

   var blobName = GetBlobNameFromUrl(imageUrl);
   var blobClient = blobContainerClient.GetBlobClient(blobName);
   await blobClient.DeleteIfExistsAsync();
}
```

Користувачі можуть переглядати профілі інших зареєстрованих осіб, однак редагування доступне лише для власного акаунта, що контролюється як на рівні UI, так і через перевірку прав доступу на сервері. В цілому реалізація функціоналу профілю відповідає сучасним вимогам до зручності користування, гнучкості налаштувань та безпеки персональних даних.

3.1.3. Стрічка публікацій, створення та коментування постів

Програма дозволяє користувачам створювати контент, взаємодіяти з ним та переглядати пости інших. Центральною сутністю є модель Post, яка містить такі основні поля: унікальний ідентифікатор, текстовий контент, дату створення, а також зв’язки з автором, зображеннями та коментарями. Автор поста представлений через посилання на користувача, що дозволяє однозначно ідентифікувати творця контенту. Зображення, що додаються до поста, зберігаються у вигляді URL, а коментарі пов’язані з постом через відповідну колекцію, що підтримує вкладеність. Вкладені коментарі реалізовані через поле *ParentId*, яке вказує на батьківський коментар, дозволяючи будувати ієрархію відповідей.

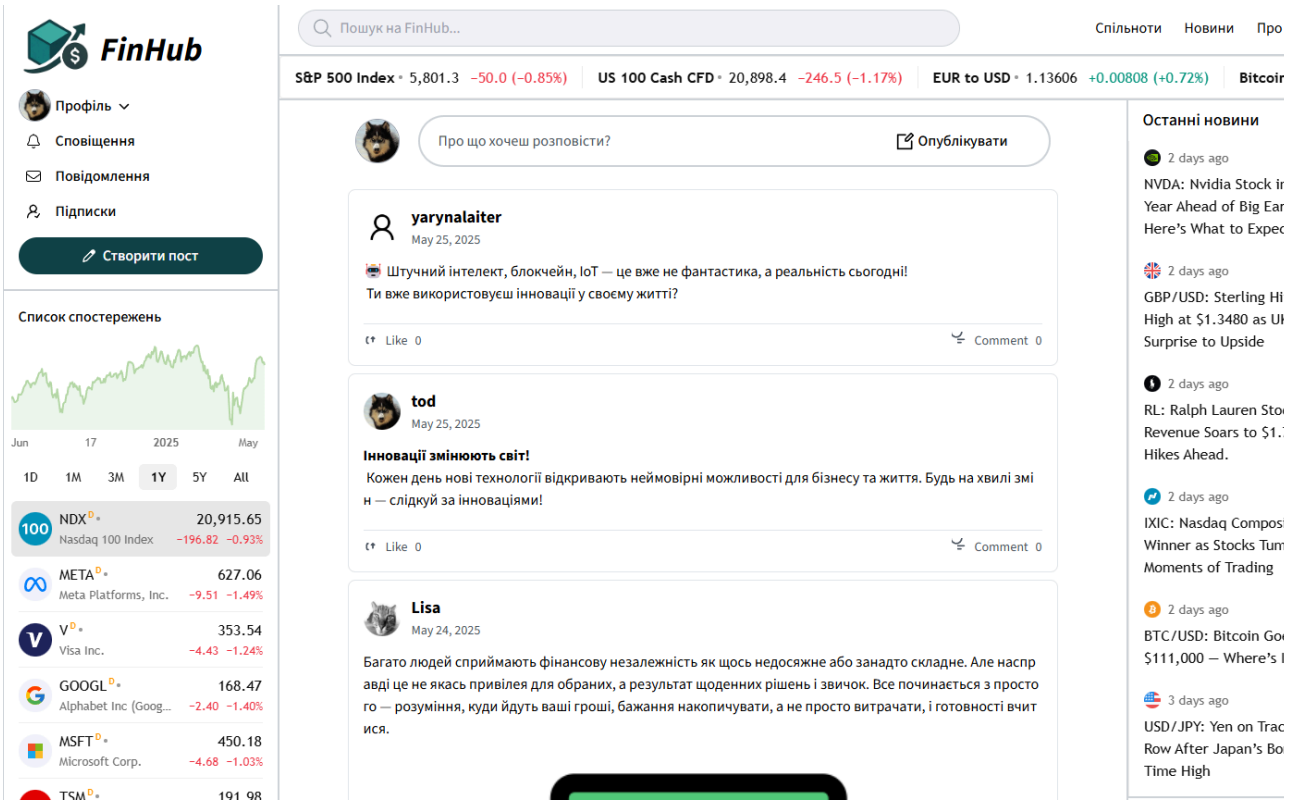

Рис. 3.3. Головна стрічка постів

Джерело: розроблено автором

Формування стрічки публікацій у системі реалізоване шляхом поділу на кілька логічних категорій, кожна з яких відповідає певним інформаційним потребам користувача. Основна стрічка, яка відкривається за замовчуванням, містить усі публікації, що створені зареєстрованими користувачами. В межах цієї стрічки передбачено механізм виділення найпопулярніших постів — таких, що набрали найбільше вподобань або коментарів за певний період. Ці популярні пости мають більшу ймовірність бути показаними у верхній частині, що забезпечує підвищену видимість найактуальнішого контенту. Основна стрічка є відкритою для всіх користувачів незалежно від того, чи мають вони підписки.

Окрім загальної стрічки, реалізована також вкладка "Підписки", яка містить пости виключно від тих користувачів, на яких підписаний активний користувач. Це дозволяє формувати персоналізовану стрічку новин, орієнтовану на інтереси конкретного користувача, зменшуючи кількість несуттєвої інформації. Такий підхід сприяє більш зручному споживанню контенту та підвищує залученість до взаємодії з публікаціями.

Лістинг коду 3.4. Метод для отримання постів з реалізованою пагінацією

```
getPosts(pageNumber: number, pageSize: number):
Observable<HttpResponse<Post>> {
    const params = new HttpParams()
      .set('pageNumber', pageNumber)
      .set('pageSize', pageSize);

    return this.http
      .get<Post>(`${this.baseUrl}/post`, {
        observe: 'response', params, headers: { 'Cache-Control': 'no-cache'
},
        transferCache: { includeHeaders: ['Pagination'] }
      }).pipe(
        tap(response => {
          const paginationHeader = response.headers.get('Pagination');
          this.paginatedResult.set({
            items: response.body ? (Array.isArray(response.body) ?
response.body : [response.body]) : [],
            pagination: JSON.parse(paginationHeader!),
          });
        }
        )
      )
  }
```

Для підвищення залученості користувачів у постах реалізована система лайків. Користувачі можуть ставити вподобайки, що зберігаються і враховуються при формуванні популярних постів. Коментарі, які супроводжують пости, також мають інформацію про дату створення, а для їх виведення застосовується пагінація з можливістю сортування за датою.

Створення постів відбувається через інтерфейс з використанням редактора Quill Editor, який забезпечує зручне форматування тексту. Контент, отриманий із редактора, додатково обробляється клієнтською логікою для очищення від небажаних тегів та стилів, що допомагає підтримувати уніфікований вигляд і безпеку. Зображення в постах додаються окремо: вони конвертуються з формату *base64* у файлові об’єкти і надсилаються на сервер у складі FormData разом з іншими даними поста. На сервері приймається запит із формою, у якій окремо

обробляються текстовий контент і файли зображень. Всі зображення зберігаються у хмарному сховищі Azure, а посилання на них зберігаються у моделі поста.

Лістинг коду 3.5. Конвертація base64-рядків зображень у файли

```
submitPost(images: string[]) {
  const formData = new FormData();
  formData.append('UserEmail', this.currentUser()?.user.email!);
  formData.append('Content', this.content);

  images.forEach((base64Image, index) => {
    const file = this.base64ToFile(base64Image, `image${index}.png`);
    formData.append('Images', file);
  });

  this.postService.createPost(formData).subscribe(() => {
    this.closeModal();
  });
}
```

На сервері для створення поста реалізовано API, яке приймає форму з текстом і файлами, створює новий запис у базі даних і повертає створений об’єкт поста. Запит захищений через JWT, що гарантує автентифікацію користувача.

Коментування постів організовано через окремі API-ендпоінти, які забезпечують роботу з коментарями на сервері. Модель коментаря включає поля для ідентифікатора поста, автора, тексту, а також опційне поле ParentId, що дає можливість створювати вкладені відповіді і будувати багаторівневу структуру дискусій.

Користувач може додавати нові коментарі або відповідати на вже існуючі, а також видаляти лише власні коментарі, що гарантує контроль над власним контентом. Запити на отримання коментарів підтримують пагінацію та фільтри, що дозволяє отримувати обмежену кількість записів за раз для оптимізації продуктивності.

На клієнті взаємодія з коментарями реалізована через CommentService, який відповідає за відправку запитів на створення, отримання та видалення коментарів. Така організація роботи забезпечує зручність і ефективність обробки коментарів.

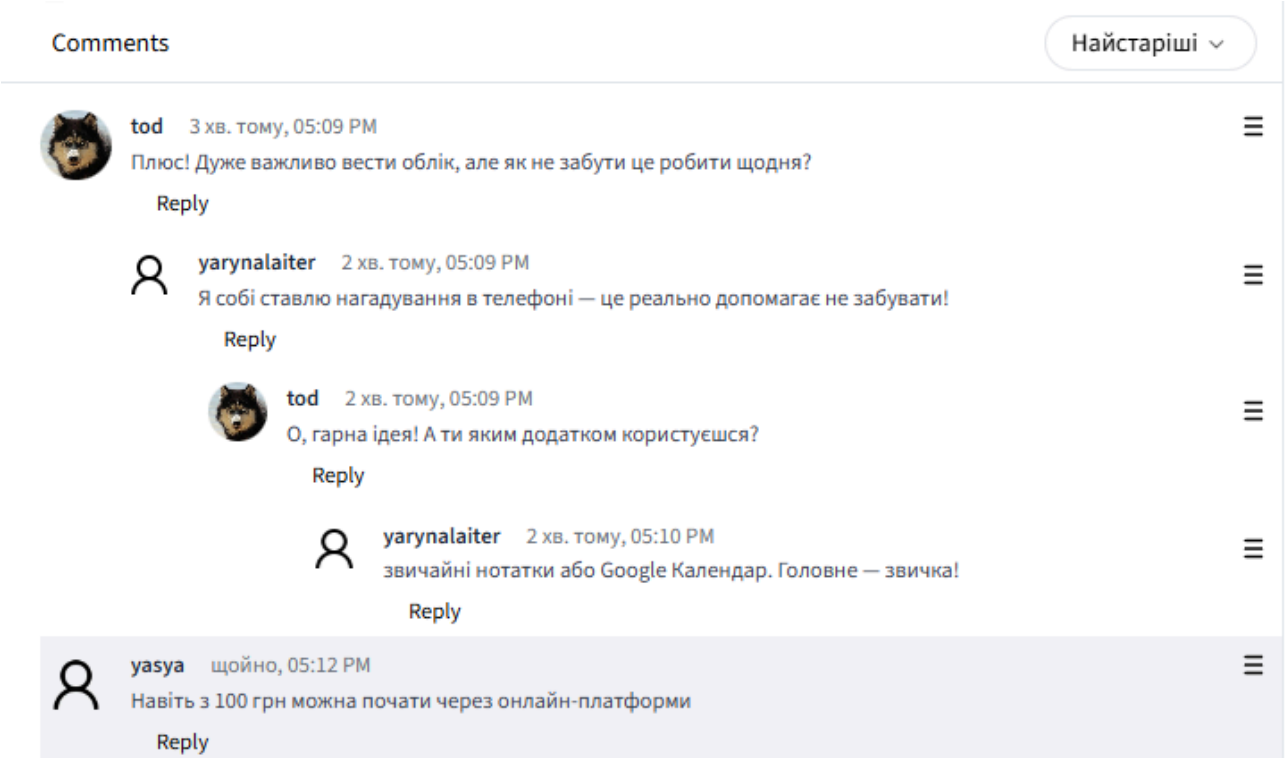

Рис. 3.4 Логіка вкладених коментарів на платформі

Джерело: розроблено автором

Пости та коментарі містять інформацію про дату створення, яка відображається користувачам у форматі, зручному для читання. Для цього у фронтенді використовується спеціальний Angular pipe, який форматує дату у вигляді, наприклад, «*Травень 9, 2025*». Це підвищує зрозумілість і естетичність інтерфейсу.

Форма створення поста відкривається у модальному вікні, що не відволікає від перегляду стрічки. Всі операції відбуваються з урахуванням сучасних практик безпеки та зручності користування.

3.1.4. Механізм роботи з хабами та модерація

Механізм роботи з хабами в системі передбачає комплексну модель взаємодії користувачів з тематичними спільнотами, які об'єднують учасників за інтересами. Кожен хаб має свого власника або адміністратора, який володіє повним набором прав для управління спільнотою, а також учасників, які створюють основний контент.

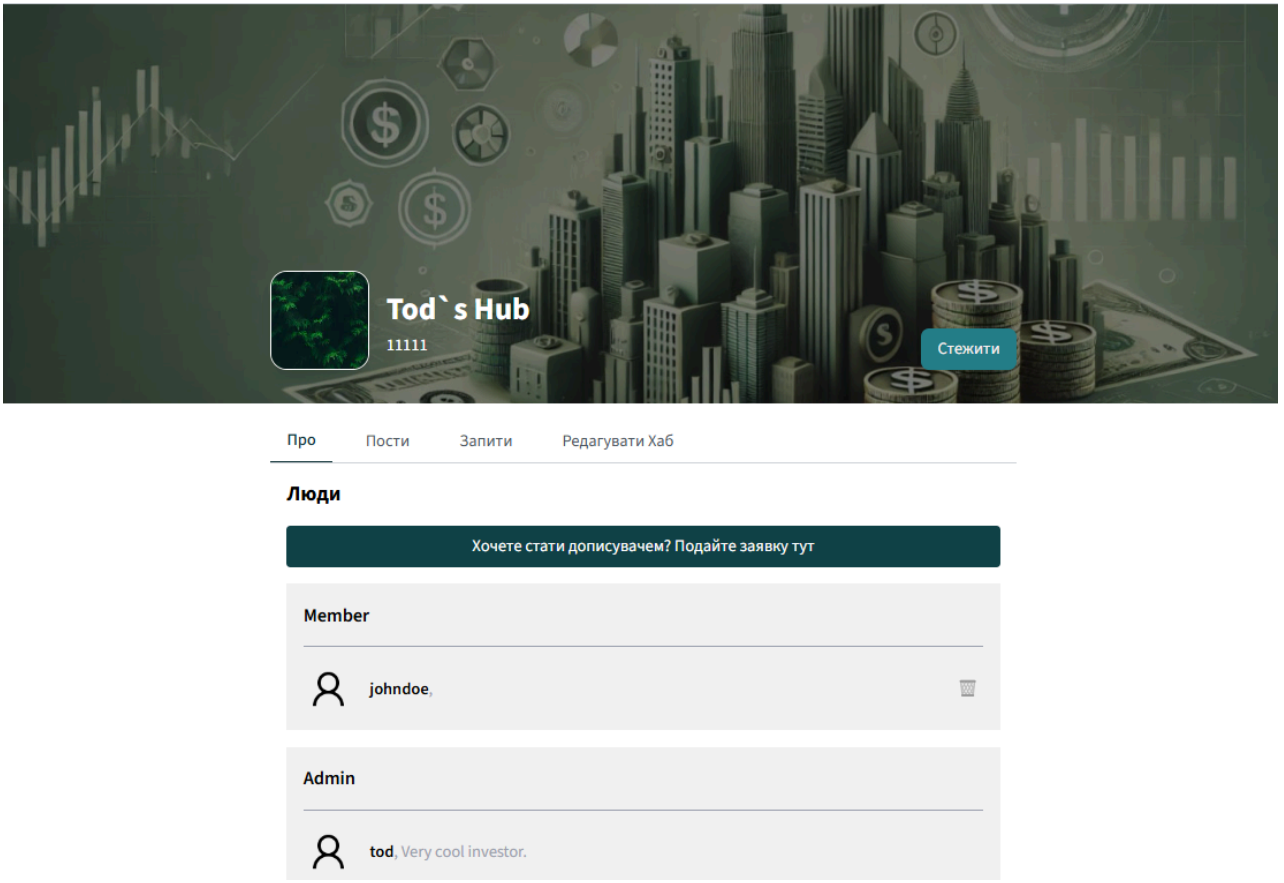

Рис. 3.5 Вкладка учасників хабу

Джерело: розроблено автором

Користувачі можуть подавати запити на вступ до хабу через спеціальну форму та кнопку «Приєднатися», що ініціює створення запису у базі даних із статусом «Очікує підтвердження». Цей запит містить ідентифікатор користувача, хабу та час подачі. Запити зберігаються в окремій таблиці HubJoinRequest, яка відображає поточний стан заявки — «Очікує», «Прийнято» або «Відхилено». Таке розділення дозволяє легко керувати процесом та фільтрувати заявки.

Адміністратор хабу мають доступ до інтерфейсу управління заявками, де вони можуть переглядати всі нові запити на вступ, оцінювати їх та приймати рішення про дозвіл або відмову. Після підтвердження заявка переходить у статус «Прийнято», і користувач автоматично додається до списку учасників хабу — таблиці HubMember, яка зберігає інформацію про дату вступу та роль учасника (наприклад, звичайний учасник, адмін). Якщо запит відхилено, користувачу надсилається відповідне повідомлення.

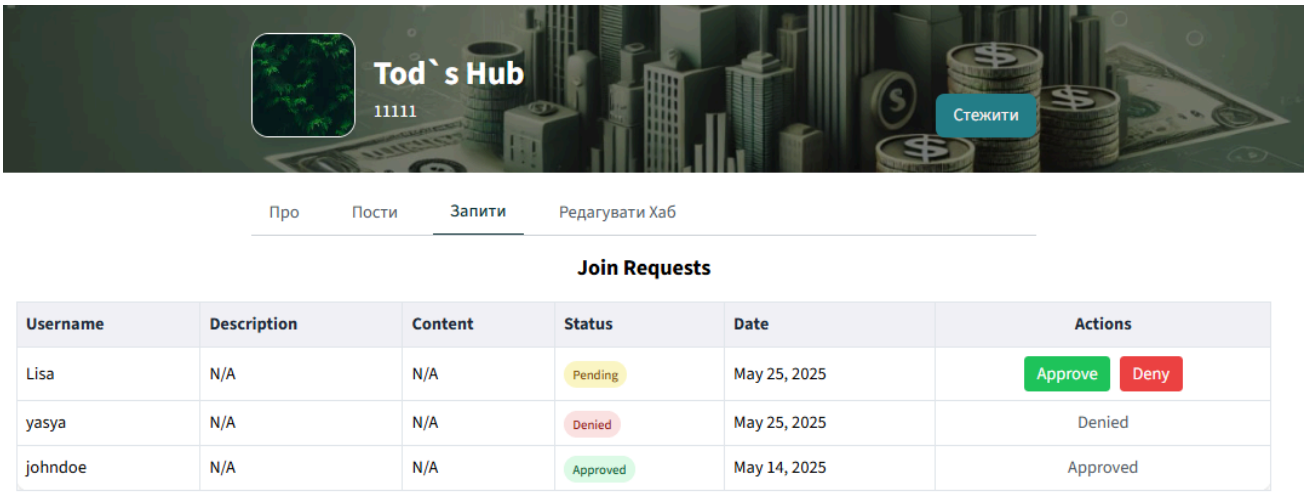

Рис. 3.6 Вкладка прийняття заявок адміном

Джерело: розроблено автором

Адміністратор хабу має широкі повноваження для управління учасниками спільноти. Він може видаляти користувачів зі спільноти у випадках, коли ті створюють неприйнятний контент або порушують встановлені правила поведінки. Процес видалення здійснюється за допомогою спеціальної команди, яка знищує зв’язок між користувачем і хабом у таблиці HubMember. Важливо, що при цьому сам користувач не видаляється з системи загалом — він лише вилучається зі складу конкретної спільноти, зберігаючи свій обліковий запис та доступ до інших функцій платформи.

Окрім керування учасниками, адміністратор має можливість редагувати інформацію самого хабу. Це включає зміну основного та заднього фото, а також редагування назви і опису хабу. Завдяки цьому хаб завжди може залишатися актуальним та відповідати потребам і тематичній спрямованості своєї спільноти.

Усі ці операції захищені надійними механізмами авторизації і контролю доступу. Лише користувачі, які мають відповідні права адміністратора, можуть виконувати подібні дії. API-ендпоінти, що відповідають за ці функції, обов’язково перевіряють дійсність JWT-токена та роль користувача, що забезпечує високий рівень безпеки і гарантує цілісність даних. Такий підхід дозволяє уникнути несанкціонованих змін і забезпечує надійний захист внутрішніх ресурсів платформи.

Лістинг коду 3.6. Метод на клієнті для підтвердження запиту на вступ до хабу:

```
 approveRequest(notificationId: string) {
        const headers = new HttpHeaders()
            .set('Authorization', `Bearer ${this.authService.currentUser()?.token}`);

        return this.http.put(`${this.baseUrl}/hub/approve-request/${notificationId}`,
{}, { headers });
    }
```

3.1.5. Реалізація обміну повідомленнями між користувачами

Реалізація системи обміну повідомленнями між користувачами передбачає не лише базовий функціонал чату, але й ряд додаткових можливостей, які роблять спілкування зручним, інформативним і швидким. Інтерфейс чату складається з двох основних частин: списку діалогів з усіма користувачами, з якими є історія листування, та області активного чату, де безпосередньо відображаються повідомлення.

З правого боку від основного вікна чату розміщується список усіх користувачів, з якими в поточного користувача вже був обмін повідомленнями. Цей список оновлюється на основі даних з бекенду, який повертає масив об’єктів типу ChatUserDto. Кожен елемент цього списку відображає ім’я користувача, його аватар, останнє повідомлення у цьому чаті та кількість непрочитаних повідомлень, якщо такі є. Реалізовано також клієнтський пошук по цьому списку, що дає змогу швидко знайти потрібного співрозмовника серед уже наявних діалогів.

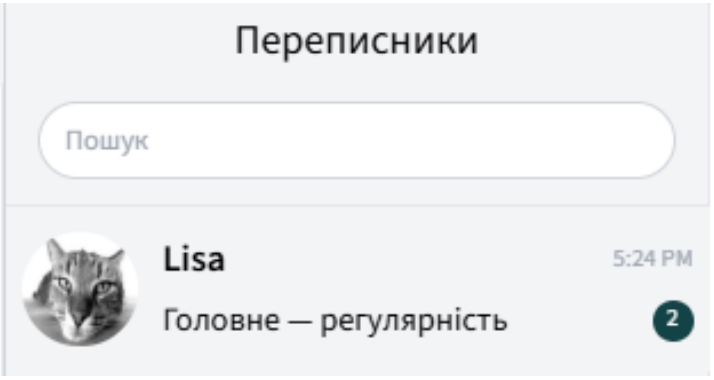

Рис. 3.7. Бічна панель з списом чатів

Джерело: розроблено автором

Коли користувач натискає на контакт у цьому списку, відкривається діалог, у якому відображається історія листування, згрупована за датою надсилання повідомлень. Кожне повідомлення містить текст, дату й час, а також вказівку, чи було воно прочитано. Це дозволяє обом сторонам чіткого бачити, чи доставлено повідомлення, і коли його було прочитано.

Завантаження повідомлень здійснюється з пагінацією, яка активується при прокручуванні вгору — таким чином, не потрібно завантажувати всю історію одразу, а лише частину, що відображається на екрані. Це забезпечує високу продуктивність навіть у разі тривалих розмов. Пагінація реалізована на боці сервера, а клієнтський інтерфейс надсилає запити з параметрами сторінки.

Надсилання повідомлення реалізується через метод sendMessage, який надсилає POST-запит із вмістом повідомлення та іменем одержувача:

Лістинг коду 3.7. Метод для відправлення повідомлень

```
sendMessage(username: string, content: string) {
    return this.http.post<Message>(`${this.baseUrl}/message`, {
recipientUsername: username, content }, {
        headers: new HttpHeaders().set('Authorization', `Bearer
${this.authService.currentUser()?.token}`)
    });
}
```

Після надсилання повідомлення воно зберігається в базі даних, що гарантує його надійне збереження та доступність для подальшого опрацювання. Одночасно у користувача-одержувача автоматично оновлюється лічильник непрочитаних повідомлень, що відображає кількість нових повідомлень, які він ще не переглянув. Самі повідомлення з’являються у вхідних або відповідному діалозі під час наступного звернення клієнта до сервера або при оновленні сторінки чи діалогу.

Кожне повідомлення має свій статус прочитання, який дозволяє відслідковувати, чи було повідомлення відкрито отримувачем. Коли користувач відкриває діалог, усі вхідні непрочитані повідомлення цього діалогу автоматично позначаються як прочитані. Для цього виконується виклик спеціального методу markMessagesAsRead, який надсилає POST-запит до API і оновлює статус

повідомлень на сервері. Такий підхід дозволяє підтримувати актуальний стан повідомлень і забезпечує точну індикацію для користувача про прочитані та непрочитані повідомлення.

Лістинг коду 3.8. Метод для познаення повідомлень як прочитаних

```
markMessagesAsRead(senderUsername: string): Observable<void> {
    return this.http.post<void>(
        `${this.baseUrl}/message/mark-as-read/${senderUsername}`,
        {}, {
            headers: new HttpHeaders().set('Authorization', `Bearer
${this.authService.currentUser()?.token}`)
        }
    );
}
```

Ця логіка дозволяє точно контролювати стан повідомлень і повідомляти користувача про нові вхідні. У самому списку контактів зліва поруч з кожним користувачем відображається кількість непрочитаних повідомлень, а в чаті ці повідомлення виділяються візуально.

Повідомлення також можна видаляти — для цього є метод deleteMessage, який викликає відповідний API-запит, і повідомлення зникає як із інтерфейсу, так і з бази даних:

Лістинг коду 3.9. Метод для видалення повідомлення

```
deleteMessage(id: string) {
    return this.http.delete(`${this.baseUrl}/message/${id}`, {
        headers: new HttpHeaders().set('Authorization', `Bearer
${this.authService.currentUser()?.token}`)
    });
}
```

Таким чином, система обміну повідомленнями у проєкті реалізована як повноцінний чат з реальним часом, контролем прочитаності, збереженням історії, зручним інтерфейсом пошуку і ефективною пагінацією для роботи з великими обсягами даних. Це створює зручне середовище для комунікації всередині соціальної мережі.

3.1.6. Система сповіщень про події у системі

У системі реалізовано повноцінний механізм сповіщень, який дозволяє інформувати користувачів про важливі події, пов’язані з їхньою активністю або взаємодією з іншими учасниками платформи. Основна мета цієї системи — забезпечити оперативну реакцію на зміни у середовищі користувача та підвищити залученість у процеси, що відбуваються в межах соціальної мережі.

Сповіщення створюються автоматично у відповідь на певні дії користувачів або події в системі. До прикладу, якщо хтось залишив лайк, надіслав запит на вступ до хабу чи прийняв такого запиту, почав стежити за користувачем створюється відповідне повідомлення. Кожне сповіщення містить інформацію про тип події, її джерело (тобто ініціатора), а також мітку часу.

Типи сповіщень охоплюють як особисті взаємодії (наприклад, отримання повідомлення або підписку), так і дії, пов’язані з хабами, включно з запитами на вступ, прийняттям, видаленням з хабу або призначенням модератора. Всі ці події на рівні доменної логіки призводять до створення нової сутності Notification, яка зберігається у базі даних. Кожне сповіщення містить поля, що вказують на отримувача, ініціатора, тип, опис події, а також прапорець прочитання.

При кожному відкритті панелі сповіщень відбувається запит на сервер для отримання нових повідомлень. Відображення організоване у вигляді окремої сторінки, де користувач може бачити список подій з коротким описом, датою і статусом прочитання. Також з правого боку він має можливість фільтрації повідомлень залежно від типу події, до якої ці сповіщення належать (лайки, стеження, хаби). Непрочитані сповіщення відмічаються окремим стилем і підраховуються для відображення у загальному інтерфейсі — у вигляді іконки.

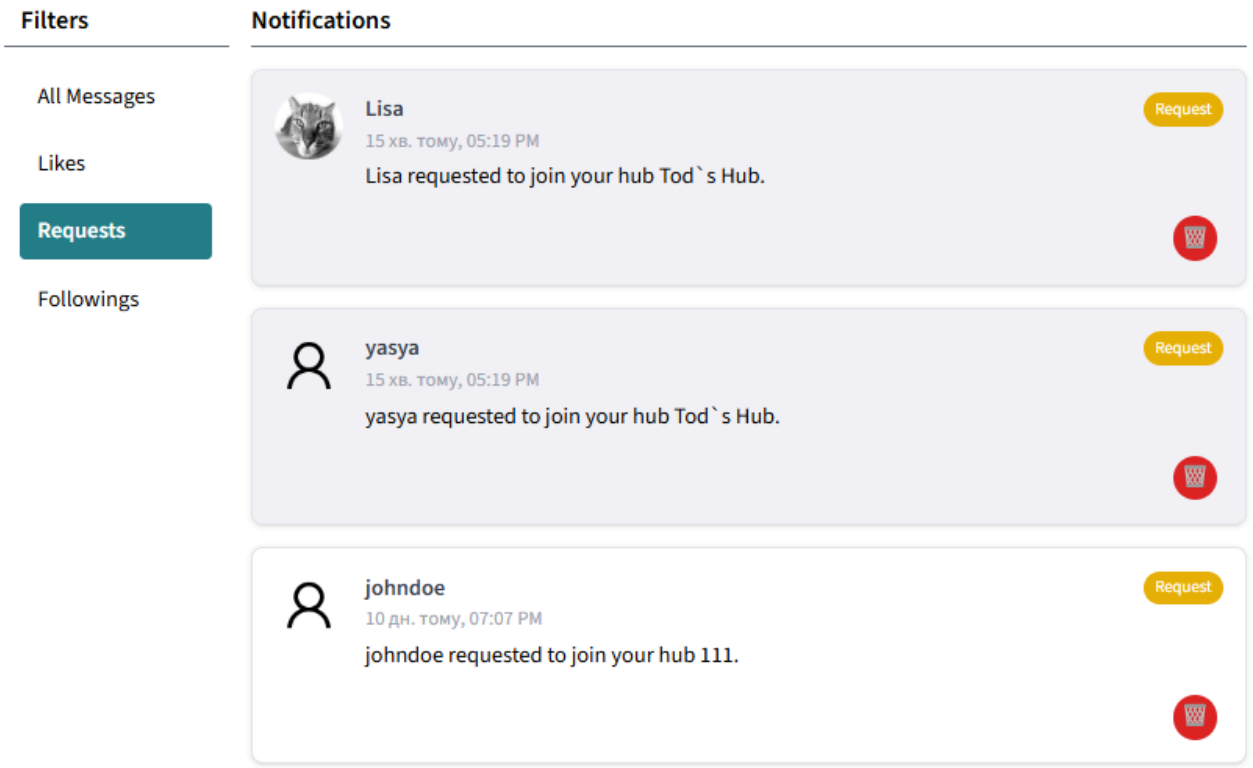

Рис. 3.8. Панель фільтрованих повіщень

Джерело: розроблено автором

Окрім пасивного відображення, користувач може клікнути на сповіщення, що перенаправляє його безпосередньо до відповідної сутності або сторінки: наприклад, до публікації, хабу або профілю іншого користувача. Після цього сповіщення автоматично позначається як прочитане.

З технічного боку система сповіщень реалізована шляхом взаємодії кількох ключових компонентів, що відповідають принципам чистої архітектури. Основну роль відіграє контролер NotificationController, який забезпечує набір захищених маршрутів (ендпоінтів) для роботи зі сповіщеннями. Ці маршрути дозволяють клієнту отримувати список сповіщень, позначати їх як прочитані, а також видаляти непотрібні записи.

Для обробки бізнес-логіки використовується сервісний шар, який опрацьовує запити контролера, а також шаблон посередника MediatR. Цей патерн дозволяє централізовано управляти командами та запитами, розділяючи обробку запитів і покращуючи підтримуваність коду.

Кожен запит до контролера проходить автентифікацію за допомогою JWT-токена, що гарантує, що лише авторизовані користувачі можуть виконувати операції зі сповіщеннями. Ідентифікація користувача здійснюється на основі електронної пошти, витягнутої з токена, що дозволяє коректно зв'язати запити зі сповіщеннями конкретного користувача і забезпечує безпеку та цілісність даних у системі.

Лістинг коду 3.10. Метод на серверній частині для отримання сповіщень

```
[HttpGet]
[Authorize]
public async Task<ActionResult<IEnumerable<GetNotificationDto>>>
GetNotificationsForUser()
{
   var email = User.GetEmail();
   var currentUser = await mediator.Send(new GetByEmailUserQuery(email));
   if (currentUser == null)
       return Unauthorized();

   var notifications = await
notificationService.GetAllNotificationsForUser(currentUser.Id);
   return Ok(notifications);
}
```

Реалізована система забезпечує інтерактивність, реагування на події та дозволяє користувачеві залишатися в курсі всіх важливих змін у межах платформи без необхідності перевіряти кожну дію вручну. Це критичний функціональний елемент соціального застосунку, що підвищує зручність користування і взаємодію між учасниками.

3.2. Технічна логіка взаємодії компонентів

Технічна логіка взаємодії між клієнтом, сервером, API та базою даних у системі побудована відповідно до принципів чистої архітектури з чітким розділенням відповідальностей між компонентами, що забезпечує модульність, гнучкість і зручність підтримки. У центрі цієї взаємодії лежить клієнтський

застосунок, створений за допомогою Angular, та серверна частина на платформі .NET, яка побудована з урахуванням патернів CQRS (Command Query Responsibility Segregation) і MediatR.

Уся взаємодія починається з користувача, який ініціює певну дію — наприклад, надсилання повідомлення, підписку на іншого користувача або участь у хабі. Клієнтська частина надсилає HTTP-запит до REST API, передаючи необхідні параметри та токен автентифікації у заголовках. API-контролери, розташовані в шарі API, приймають запит і не містять бізнес-логіки безпосередньо — замість цього вони передають команди або запити в MediatR.

Запити (Queries) використовуються для отримання даних і спрямовуються в обробники, які читають інформацію через сервіси або напряму з репозиторіїв, без зміни стану системи. Команди (Commands), навпаки, відповідають за зміну стану — створення, оновлення чи видалення сутностей. Усі команди також передаються через MediatR до відповідних обробників на рівні Application, де міститься бізнес-логіка. Тут за потреби відбувається валідація, обробка умов і підготовка до запису в базу даних.

Для роботи з даними використовується шар Infrastructure, який реалізує репозиторії на основі Entity Framework Core. Вони звертаються до бази PostgreSQL і виконують як стандартні CRUD-операції, так і складніші запити з фільтрацією, пагінацією та агрегацією.

Автоматичне перетворення між доменними сутностями та DTO виконує AutoMapper, що усуває дублювання коду. Безпечну ідентифікацію користувачів і контроль доступу до захищених API забезпечують JWT токени.

Лістинг коду 3.11. User мапінг профіль

```
public UserProfileMappingProfile()
{
   CreateMap<CreateUserDto, User>()
       .ForMember(dest => dest.UserName, opt => opt.MapFrom(src =>
src.Username))
       .ForMember(dest => dest.Email, opt => opt.MapFrom(src => src.Email))
       .ForMember(dest => dest.PasswordHash, opt => opt.MapFrom(src =>
PasswordHasher.HashPassword(src.Password)));}
```

Таким чином, усі компоненти — клієнт, API, бізнес-логіка, інфраструктура й база даних — взаємодіють через чітко визначені контракти, забезпечуючи як розділення запитів і команд (через CQRS), так і централізовану обробку через MediatR. Це дозволяє масштабувати систему, додавати нові функції без порушення існуючих модулів та спрощує тестування кожного окремого шару.

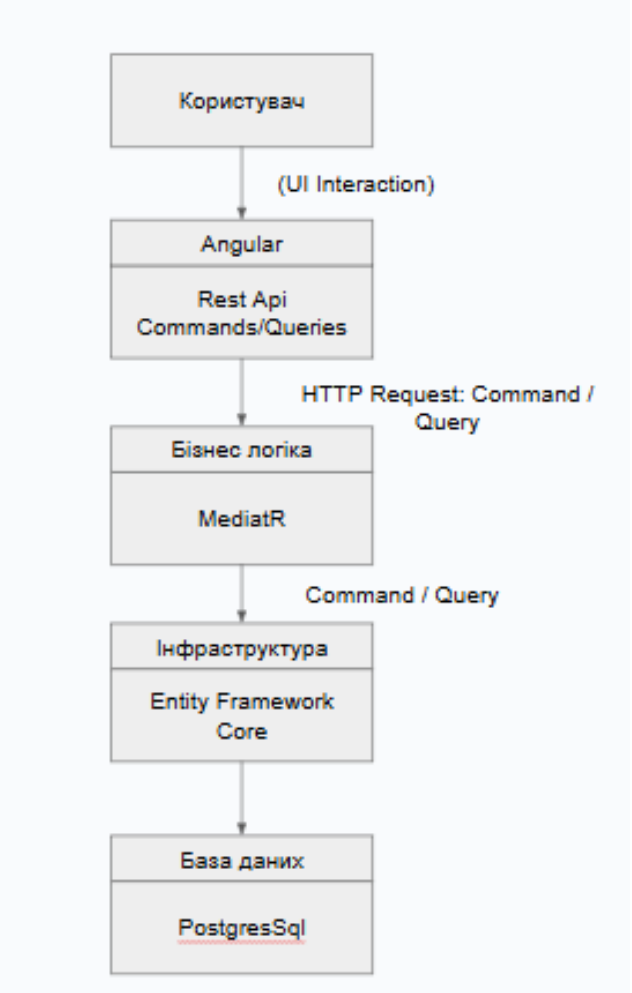

Рис. 3.9. Схема взаємозв’язків між компонентами

Джерело: розроблено автором

3.3. Обробка виняткових ситуацій

Обробка виняткових ситуацій у системі реалізована на всіх рівнях — від клієнта до серверної частини — з урахуванням безпеки, зручності для користувача та підтримуваності коду. Особливу увагу приділено помилкам, що можуть виникати

під час автентифікації, авторизації, повторної реєстрації, надсилання запитів до захищених маршрутів або порушення бізнес-логіки.

На стороні клієнта всі HTTP-запити супроводжуються механізмами перехоплення та обробки помилок. Angular-сервіси використовують RxJS-оператор catchError, що дозволяє реагувати на помилки вже на етапі запиту. Якщо, наприклад, при автентифікації API повертає статус 401 або 403, інтерфейс одразу сповіщає користувача відповідним повідомленням про недійсні облікові дані або відсутність прав доступу. Подібним чином обробляється ситуація, коли користувач намагається повторно зареєструватися з уже використаною адресою електронної пошти — API повертає статус 400 з описом проблеми, який відображається у формі введення.

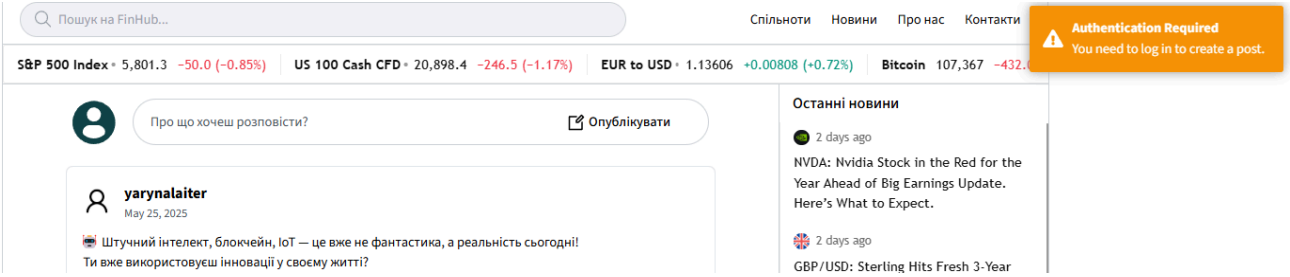

Рис. 3.10 Сповіщення про те, що потрібно залогінитися

Джерело: розроблено автором

На серверній стороні всі публічні точки входу до API захищені за допомогою атрибута [Authorize], що автоматично відкидає запити без коректного JWT-токена. Крім того, авторизаційні атрибути застосовуються для контролю доступу до окремих дій, наприклад, редагування чи видалення об'єктів, що не належать користувачу. Виявлення спроб несанкціонованого доступу призводить до повернення статусу 403 з описом причини.

Помилки, що виникають на рівні обробки команд. Якщо, наприклад, користувач намагається надіслати запит на хаб, до якого вже приєднаний, або коментує неіснуючий пост, викидається виняток, який перехоплюється централізовано.

Додатково всі критичні ситуації, включаючи помилки авторизації, неправильні запити та винятки, що виникають унаслідок логічних конфліктів, реєструються у логах, що дозволяє проводити аудит і спрощує відлагодження. Цей підхід дозволяє

зберігати цілісність системи, інформувати користувача про помилки у зручній формі та забезпечувати безпеку і стабільність під час експлуатації.

3.4. Контейнеризація та розгортання системи

У сучасній розробці вебзастосунків дедалі важливішою стає можливість забезпечити ізольоване, відтворюване та незалежне від середовища виконання програмного забезпечення. Для цього в межах даного проєкту була застосована технологія контейнеризації за допомогою платформи Docker. Контейнеризація дозволяє об’єднати програму разом з усіма необхідними бібліотеками, залежностями та конфігураціями у самодостатній модуль — контейнер, який гарантовано буде працювати однаково в будь-якому середовищі: локальному, тестовому чи продакшн.

У процесі контейнеризації було прийнято рішення охопити лише серверну частину системи та базу даних. Клієнтська частина (Angular) не контейнеризується, оскільки застосунок орієнтовано на розгортання бекенду як REST API з подальшим окремим хостингом фронтенду у статичному середовищі (наприклад, через GitHub Pages або Azure Static Web Apps).

В основі серверної архітектури лежить два головних компоненти: застосунок FinanceHub.API, реалізований на платформі .NET 8, а також система керування базами даних PostgreSQL. Для об'єднання цих компонентів та спрощення управління їх конфігурацією використано Docker Compose. Конфігурація описується у файлі docker-compose.yml, де кожен із сервісів має власні параметри запуску, змінні середовища, мережеві налаштування та точки монтування томів.

Розроблений файл docker-compose.yml описує сервіс financehub, який відповідає за запуск .NET API, та сервіс db, який піднімає екземпляр PostgreSQL. Обидва контейнери об’єднані в окрему логічну мережу типу bridge, що дає змогу реалізувати безпечну взаємодію між ними без зовнішньої маршрутизації. API-сервер доступний ззовні на порту 8080, який проброшено на той самий порт на хості, тоді як база даних доступна на порту 5432. Для забезпечення збереження даних при

перезапуску системи або контейнерів використано том postgres-data, що зберігає файли бази даних поза межами життєвого циклу контейнера.

У таблиці 3.1 наведені характеристики контейнерів, що використовуються у проєкті.

Таблиця 3.1

Характеристики контейнерів

Компонент	**Образ / Базовий образ**	**Порт (внутр./зовн.)**	**Дані/Томи**	**Змінні середовища**
FinanceHub API	mcr.microsoft.com/dotnet/aspnet:8.0	8080/8080	~/.azure (Azure креденшіали)	PORT, AZURE_TENANT_ID, ін.
PostgreSQL	postgres:latest	5432/5432	postgres-data	POSTGRES_DB, POSTGRES_USER, ін.

Окрема увага приділяється змінним середовища. У конфігурації сервісу бази даних вони визначають ім’я бази, користувача та пароль. Для бекенду ж передаються параметри, пов’язані з Azure-автентифікацією, зокрема AZURE_TENANT_ID, AZURE_CLIENT_ID, AZURE_CLIENT_SECRET. Їхні значення не вбудовуються безпосередньо у Dockerfile або Compose-файл, а підставляються динамічно з .env-файлу, що відповідає принципам безпечного розгортання і забезпечує розділення конфігурації від коду.

Щодо структури збірки .NET-застосунку, варто зазначити, що використано багатостадійний Dockerfile, який оптимізує процес формування фінального образу. На першому етапі застосовується образ із повним SDK для відновлення залежностей та компіляції проєкту. На наступному етапі відбувається публікація артефактів у режимі Release. Фінальний контейнер створюється на основі легшого рантайм-образу (aspnet), куди копіюються тільки зібрані файли. Такий підхід дозволяє зменшити розмір образу, мінімізувати поверхню атаки та відокремити стадії розробки й виконання.

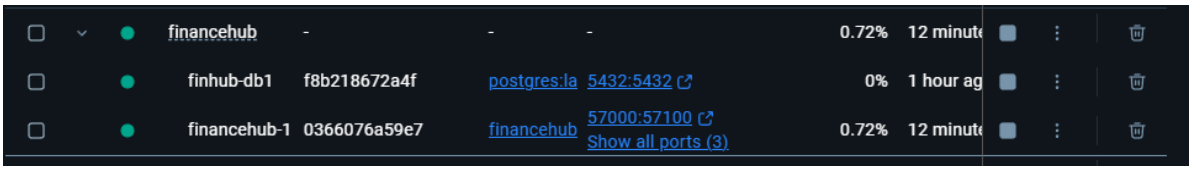

Рис. 3.11. Запущені докер контейнери

Джерело: розроблено автором

Розгортання системи відбувається локально або в хмарі за допомогою простої команди docker compose up -d --build. Завдяки контейнеризації, процес підняття середовища зводиться до автоматичного створення образів, запуску контейнерів, створення мереж та монтування томів. Додаткові конфігурації, як-от авторизація в Azure або інтеграція з CI/CD-системами (GitHub Actions), можуть бути реалізовані через зовнішні секрети або змінні середовища в пайплайнах.

Таким чином, впровадження контейнеризації дало змогу досягти високої мобільності, повторюваності середовища, простоти супроводу та масштабування. Особливо це важливо в контексті потенційного розгортання у продакшн, де стабільність та передбачуваність оточення мають критичне значення. Крім того, модульна структура конфігурації дозволяє гнучко адаптувати систему до майбутніх змін, таких як додавання кеш-сервісу, reverse-proxy, балансування навантаження чи інші мікросервіси.

ВИСНОВКИ

У межах дипломної роботи було спроєктовано та реалізовано повнофункціональну інформаційну систему у вигляді тематичної соціальної мережі, орієнтованої на фінансову тематику, що поєднує в собі сучасні підходи до архітектури клієнт-серверної взаємодії, безпечної автентифікації, інтерактивного обміну контентом та реального спілкування між користувачами.

На етапі аналізу було досліджено актуальні рішення у сфері соціальних платформ, зосереджено увагу на їхніх перевагах і недоліках, визначено специфіку тематичних мереж, орієнтованих на фінанси, та обґрунтовано потребу в створенні платформи, яка дозволяє не лише обмінюватися інформацією, а й формувати спільноти за інтересами (хаби), взаємодіяти в реальному часі, отримувати сповіщення та мати зручний механізм модерації.

Системне проєктування передбачало вибір технологічного стеку, що включає Angular як клієнтський фреймворк, .NET Web API для серверної логіки, PostgreSQL як СУБД, а також Azure Storage для зберігання зображень. Архітектура була побудована за принципами чистої архітектури з розділенням на шари (API, Application, Domain, Infrastructure), що забезпечило гнучкість, масштабованість і зрозумілу логіку взаємодії між компонентами. В системі реалізовано шаблон CQRS із використанням бібліотеки MediatR, що дозволило розділити команди й запити, полегшити супровід і тестування. З клієнтської сторони структура побудована модульно з виокремленням core, shared та feature-модулів, що відповідають за конкретну бізнес-логіку.

Під час реалізації розроблено ключові функціональні компоненти: безпечну реєстрацію та автентифікацію на основі JWT-токенів і Google OAuth, редагування профілю користувача, створення постів із текстом і зображеннями, коментування з підтримкою вкладеності, стрічку публікацій (загальну та на основі підписок), гнучку систему підписки, механізм створення та модерації хабів. Також розроблено повноцінний чат із групуванням повідомлень за датами, маркуванням прочитаних

повідомлень, лістом контактів і пошуком, а також систему сповіщень, що відстежує важливі події (нові підписки, вподобання, запити на вступ до хабу).

Технічно система підтримує обробку виняткових ситуацій на всіх рівнях — від перевірки токенів авторизації та реєстрації до перевірки доступу до ресурсів, дублювання запитів чи спроб несанкціонованих дій. Контейнеризація всіх компонентів через Docker забезпечує зручне тестування, транспортування і розгортання в різних середовищах без втрати функціональності.

Таким чином, реалізована система повністю відповідає поставленим завданням. Вона не лише демонструє практичну реалізацію соціальної платформи з сучасною архітектурою, а й підтверджує ефективність обраного підходу до розробки, модульності та масштабування. У перспективі проєкт можна легко розвивати, додаючи нові сервіси, інтегруючи сторонні API або реалізуючи WebSocket для миттєвих оновлень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Wired. *The Second Coming of Java at Twitter*. [Електронний ресурс] – Режим доступу до ресурсу: https://www.wired.com/2013/09/the-second-coming-of-java (дата звернення: 30.04.2025).
2. Reddit GitHub Archive. *Architecture Overview*. [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/reddit-archive/reddit/wiki/Architecture-Overview (дата звернення: 12.02.2025).
3. Wikipedia. *Quora*. [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Quora (дата звернення: 01.05.2025).
4. Itexus. *Twitter Tech Stack – Powering Real-Time Social Networking at Scale*. [Електронний ресурс] – Режим доступу до ресурсу: https://itexus.com/twitter-tech-stack (дата звернення: 12.02.2025).
5. OECD. *Financial literacy levels across countries: G20 comparison*. [Електронний ресурс] – Режим доступу до ресурсу: https://www.oecd.org/financial/education/ (дата звернення: 15.05.2025).
6. Statista. Investment behavior worldwide - statistics & facts. [Електронний ресурс] – Режим доступу до ресурсу: https://www.statista.com/topics/8138/investment-behavior-worldwide/#topicOverview (дата звернення: 11.03.2025).
7. MoneyFIt. *Gen Z and Money*. [Електронний ресурс] – Режим доступу до ресурсу: https://www.morningbrew.com/daily/stories/gen-z-financial-social-media (дата звернення: 28.04.2025).
8. Business Insider. *Finance People to Follow on Twitter*. [Електронний ресурс] – Режим доступу до ресурсу: https://www.businessinsider.com/people-to-follow-on-twitter-from-finance-2017-6 (Дата звернення: 20.02.2025).
9. Statista. *Reddit: number of active users worldwide*. [Електронний ресурс] – Режим доступу до ресурсу: https://www.statista.com/statistics/443332/reddit-global-active-users/ (дата звернення: 14.05.2025).
10. Stocktwits. *Platform features overview*. [Електронний ресурс] – Режим доступу до ресурсу: https://stocktwits.com/ (дата звернення: 18.05.2025).
11. Angular Docs. Why Angular. [Електронний ресурс] – Режим доступу до ресурсу: : https://angular.io/guide/why-angular (дата звернення: 13.05.2025).

12. Angular Docs. Angular CLI Overview. [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/cli (дата звернення: 23.04.2025).
13. Angular Docs. Architecture Overview. [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/guide/architecture (дата звернення: 02.04.2025).
14. Microsoft Docs. ASP.NET Core Web API overview. [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-7.0 (дата звернення: 11.03.2025).
15. Microsoft Docs. Introduction to Clean Architecture. [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures#clean-architecture (дата звернення: 25.04.2025).
16. MediatR GitHub Repository. MediatR: Simple, unambitious mediator implementation in .NET. [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/jbogard/MediatR (дата звернення: 13.03.2025).
17. AutoMapper GitHub Repository. AutoMapper: Convention-based object-object mapper in .NET. [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/AutoMapper/AutoMapper (дата звернення: 16.05.2025).
18. PostgreSQL Global Development Group. PostgreSQL Documentation. [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/docs/current/ (дата звернення: 16.05.2025).
19. Microsoft Docs. Entity Framework Core Overview. [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 14.08.2025).
20. Docker Docs. Overview of Docker. [Електронний ресурс] – Режим доступу до ресурсу: https://docs.docker.com/get-started/overview/ (дата звернення: 15.08.2025).
21. Microsoft Docs. Azure Storage documentation. [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/azure/storage/ (дата звернення: 03.05.2025).
22. AWS. What is RESTful API? [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/what-is/restful-api/ (дата звернення: 07.04.2025).
23. Microsoft Docs. Configure JWT Bearer Authentication in ASP.NET Core. [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/configure-jwt-bearer-authentication?view=aspnetcore-9.0 (дата звернення: 09.04.2025).

24. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley Professional, 2003.
25. Ambler S. The Object Primer: Agile Model-Driven Development with UML 2.0. Cambridge University Press, 2004.
26. Elmasri R., Navathe S. Fundamentals of Database Systems. 7th Edition, Pearson, 2015.
27. Jones M., Bradley J. JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens. IETF RFC 9068, 2022. [Електронний ресурс] – Режим доступу до ресурсу: https://datatracker.ietf.org/doc/html/rfc9068 (дата звернення: 07.05.2025).
28. Richardson L., Ruby S. RESTful Web Services. O'Reilly Media, 2007.
29. Fielding R.T. Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University of California, Irvine, . [Електронний ресурс] – Режим доступу до ресурсу: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (дата звернення: 02.05.2025).
30. Microsoft Azure. What is Azure Blob Storage? [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction (дата звернення: 12.03.2025).
31. Docker Docs. Networking Overview. [Електронний ресурс] – Режим доступу до ресурсу: https://docs.docker.com/network/ (дата звернення: 24.04.2025).
32. Microsoft Docs. Authentication in ASP.NET Core. [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/security/authentication (дата звернення: 01.03.2025).
33. Microsoft Docs. System.Text.Json Namespace. [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/api/system.text.json (дата звернення: 12.05.2025).
34. Microsoft. JSON Web Token Handler for the Microsoft .NET Framework. Microsoft Learn. [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/api/system.identitymodel.tokens.jwt (дата звернення: 07.05.2025).
35. Auth0. Introduction to JSON Web Tokens. Auth0 Learn. [Електронний ресурс] – Режим доступу до ресурсу: https://auth0.com/learn/json-web-tokens (дата звернення: 18.04.2025).

ДОДАТОК А

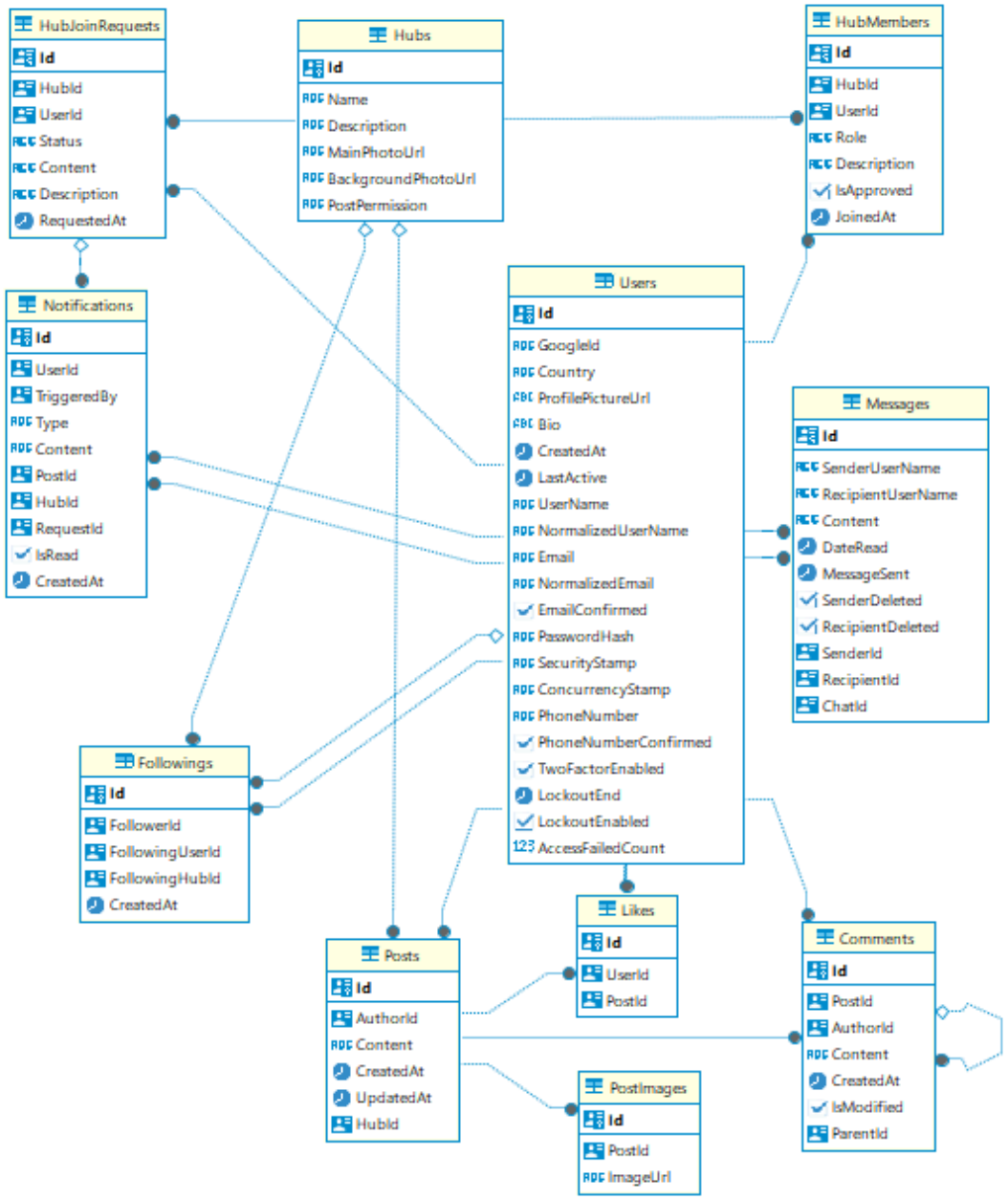

Рис. А.1 Схема таблиць та їх взаємозв’язків основної бази даних платформи (PostgreSQL)

Джерело: розроблено автором

Основні компоненти схеми включають наступне:

- Users — зберігає загальні відомості про користувачів: ім’я користувача, email, країну, біографію, посилання на аватар, дату створення, пароль, гугл айді. Пов’язана з іншими сутностями: Posts, Comments, Likes, Messages, Followings, HubMembers, HubJoinRequests та Notifications.

- Posts — зберігає дописи, створені користувачами у межах хабів: вміст допису, дата створення та оновлення. Пов’язана з Users (як автори), Hubs (місце публікації), а також з Comments, Likes і PostImages.
- Hubs — зберігає інформацію про спільноти (хаби): назву, опис, URL головного та фонового фото, а також параметри дозволів на публікацію. Пов’язана з Posts, HubMembers, HubJoinRequests та Notifications.
- Comments — зберігає коментарі до дописів, включаючи вкладені (через ParentId). Містить автора, текст, дату створення та оновлення. Пов’язана з Users та Posts.
- Followings — реалізує підписки між користувачами, тобто зв'язок багато-до-багатьох у вигляді "користувач підписався на іншого користувача". Містить дату створення підписки. Пов’язана з Users.
- Notifications — зберігає сповіщення, пов’язані з діями користувачів, наприклад, запитами на вступ до хабів або іншими подіями. Містить тип сповіщення, вміст, дату створення, та пов’язана з Users, Hubs і HubJoinRequests.
- Messages — зберігає особисті повідомлення між користувачами: вміст, дату надсилання та читання, ID чату. Пов’язана з Users через імена відправника та отримувача.

База даних підтримує цілісність зв’язків за допомогою зовнішніх ключів, що забезпечує точне відстеження взаємодій між користувачами, хабами та контентом. Це дозволяє ефективно керувати спільнотами, повідомленнями, запитами на вступ, а також динамічно масштабувати систему — наприклад, додаючи нові типи взаємодій або розширюючи функціональність хабів.

ДОДАТОК Б

Таблиця Б.1

Основні користувацьких сценаріїв з позначенням вимог до авторизації та очікуваних результатів

Назва сценарію	**Опис дії**	**Вимоги до авторизації**	**Очікуваний результат**
Реєстрація користувача	Заповнення форми з email, ім’ям, паролем; валідація; створення облікового запису; генерація JWT	Відсутня	Створення облікового запису, автоматичний вхід у персональний кабінет
Створення поста	Введення тексту, додавання зображення, валідація, відправка на сервер	Обов’язкова	Публікація поста, доступна іншим користувачам у стрічці
Перегляд постів	Перегляд публікацій інших користувачів	Необов’язкова	Відображення постів
Коментування та лайки	Додавання коментарів, ставлення вподобань	Обов’язкова	Відображення коментарів і лайків під постами
Управління хабами	Створення хабів, вступ за запитом, адміністрування	Обов’язкова	Доступ до закритих розділів хабу, спільна діяльність
Особисті повідомлення	Ініціація діалогів, обмін повідомленнями, перегляд історії	Обов’язкова	Повноцінне листування між користувачами
Сповіщення	Отримання повідомлень про активність	Обов’язкова	Отримання актуальної інформації про події
Редагування профілю	Зміна аватара, опису, імені	Обов’язкова	Оновлення профільних даних
Видалення поста / вихід з хабу	Управління власним контентом та участю в хабах	Обов’язкова	Видалення поста, вихід із спільнот