МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА СОЦІАЛЬНОЇ МЕРЕЖІ ЗАСОБАМИ REACT ТА FIREBASE»**

Виконав: студент 4 курсу, групи 6.1212-пі-2
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
А.Д. Лупашку
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, PhD Столярова А.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

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

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Лупашку Андрію Дмитровичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка соціальної мережі засобами React та Firebase

керівник роботи Столярова Анастасія Валеріївна, PhD
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 02 » грудня 2025 року № 1741-с

2. Строк подання студентом роботи 01.06.2026 р.

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз предметної області та вибір технологічного стеку.
2. Системний аналіз та проєктування архітектури застосунку.
3. Програмна реалізація та тестування системи.

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

6. Консультанти розділів роботи

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

7. Дата видачі завдання 09.12.2025 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	22.12.2025	
2.	Збір вихідних даних.	20.01.2026	
3.	Обробка методичних та теоретичних	12.02.2026	
	джерел.		
4.	Розробка першого та другого розділу.	16.03.2026	
5.	Розробка третього розділу.	18.05.2026	
6.	Оформлення та нормоконтроль	25.05.2026	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	17.06.2026	

Студент _____ А.Д. Лупашку
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.В. Столярова
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

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

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка соціальної мережі засобами React та Firebase»: 57 с., 17 рис., 3 табл., 15 джерел, 1 додаток.

АВТОРИЗАЦІЯ, БАЗА ДАНИХ, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС, СОЦІАЛЬНА МЕРЕЖА, ФРОНТЕНД, ХМАРНІ ТЕХНОЛОГІЇ, FIREBASE, REACT.

Об’єкт дослідження – процеси проєктування, розроблення та супроводження програмного забезпечення для мобільних систем соціальної комунікації.

Предмет дослідження – методи та програмні засоби кросплатформної розробки та хмарної інтеграції, що застосовуються для створення клієнтської та серверної частин мобільної соціальної мережі.

Мета роботи: зменшення часових витрат на розробку та розгортання інфраструктури мобільної соціальної мережі при одночасному підвищенні швидкості обміну даними в реальному часі шляхом використання BaaS-технологій та кросплатформного підходу.

Методи дослідження – емпіричні, методи системного аналізу та проєктування, методи об’єктно-орієнтованого та функціонального програмування, а також методи тестування програмного забезпечення.

У кваліфікаційній роботі розв’язано актуальне завдання створення сучасної, швидкої та масштабованої соціальної мережі. На основі аналізу існуючих програмних рішень обґрунтовано вибір технологічного стеку, зокрема використання React та платформи Firebase як сервісу Backend-as-a-Service (BaaS), що дозволило оптимізувати процес розробки серверної частини. У ході виконання роботи було спроєктовано архітектуру системи та розроблено оптимальну структуру нереляційної бази даних. Практично реалізовано ключовий функціонал соціальної мережі.

SUMMARY

Bachelor’s qualifying paper “Development of a Social Network Using React and Firebase”: 57 pages, 17 figures, 3 tables, 15 references, 1 supplement.

AUTHORIZATION, CLIENT-SERVER ARCHITECTURE, CLOUD TECHNOLOGIES, DATABASE, FIREBASE, FRONTEND, REACT, SOCIAL NETWORK, USER INTERFACE.

The object of the study is the processes of designing, developing, and maintaining software for mobile social communication systems.

The subject of the study is the methods and software tools for cross-platform development and cloud integration used to create the client and server parts of a mobile social network.

The aim of the study is reducing the time required to develop and deploy the infrastructure of a mobile social network while simultaneously increasing the speed of real-time data exchange by using BaaS technologies and a cross-platform approach.

The methods of research include empirical analysis of existing social media solutions, systems analysis for architectural design, object-oriented and functional programming for software implementation, and functional testing to verify system reliability.

The qualifying paper addresses the relevant task of creating a modern, fast, and scalable social network. Based on the analysis of existing software solutions, the choice of the technological stack was justified, specifically using React and Firebase as a Backend-as-a-Service (BaaS) platform, which optimized the server-side development process. During the work, the system architecture was designed and an optimal structure for a NoSQL database was developed. The key functionality of the social network has been practically implemented.

ЗМІСТ

Завдання на кваліфікаційну роботу……..…..2
Реферат 4
Summary 5
Вступ..... 8
1 Аналіз предметної області та вибір технологічного стеку 10
1.1 Огляд сучасних тенденцій розвитку соціальних мереж 10
1.2 Аналіз існуючих програмних рішень та аналогів..... 11
1.3 Обґрунтування вибору технологічного стеку 13
1.3.1 Платформа Firebase (BaaS)..... 13
1.3.2 Фреймворк React Native 15
1.3.3 Додаткові інструментальні засоби 16
1.4 Висновки до розділу 1 18
2 Системний аналіз та проєктування архітектури застосунку 20
2.1 Специфікація та аналіз вимог до системи 20
2.1.1 Функціональні вимоги..... 20
2.1.2 Нефункціональні вимоги..... 21
2.2 Проєктування діаграм варіантів використання (Use Case)..... 22
2.3 Моделювання структури нереляційної бази даних 25
2.4 Висновки до розділу 2 29
3 Програмна реалізація та тестування 30
3.1 Архітектура клієнтського застосунку, управління станом та інтеграція хмарних сервісів..... 30
3.2 Проєктування UI/UX та програмна реалізація ключових інтерфейсних модулів 36
3.3 Реалізація механізмів гейміфікації та модулів глобальної агрегації контенту 44
3.4 Налагодження та функціональне тестування системи..... 47

3.4.1 Середовище тестування та інструменти налагодження..... 48
3.4.2 Методологія функціонального тестування 48
3.4.3 Тестування безпеки та прав доступу (RBAC)..... 50
3.5 Висновки до розділу 3 51
Висновки 53
Перелік посилань..... 55
Додаток А Повна ER-діаграма бази даних 57

ВСТУП

В епоху цифровізації соціальні мережі стали основою щоденної комунікації. Раніше створення таких платформ вимагало складної серверної інфраструктури, проте сьогодні сучасні фреймворки та хмарні BaaS-сервіси значно оптимізують розробку. Вони дозволяють швидко створювати масштабовані мобільні застосунки. Оскільки потреба у зручних платформах для локальних спільнот постійно зростає, пошук шляхів прискорення їх розробки та зниження витрат є надзвичайно актуальним.

Мета роботи: зменшення часових витрат на розробку та розгортання інфраструктури мобільної соціальної мережі при одночасному підвищенні швидкості обміну даними в реальному часі шляхом використання BaaS-технологій та кросплатформного підходу.

Досягнення поставленої мети передбачає реалізацію наступних **завдань**:

- проаналізувати предметну область;
- дослідити та обґрунтувати вибір технологічного стеку;
- спроєктувати архітектуру системи, користувацький інтерфейс (UI/UX) та логічну структуру бази даних;
- розробити програмний код клієнтської частини та інтегрувати його з хмарними сервісами;
- протестувати розроблену інформаційну систему на предмет коректності роботи, безпеки та відповідності функціональним вимогам.

Об’єкт дослідження – процеси проєктування, розроблення та супроводження програмного забезпечення для мобільних систем соціальної комунікації.

Предмет дослідження – методи та програмні засоби кросплатформної розробки та хмарної інтеграції, що застосовуються для створення клієнтської та серверної частин мобільної соціальної мережі.

Методи дослідження – емпіричні, методи системного аналізу та проєктування, методи об’єктно-орієнтованого та функціонального програмування, а також методи тестування програмного забезпечення.

Для досягнення результату та реалізації клієнтської частини було обрано кросплатформний фреймворк React Native. Для забезпечення бекенд-інфраструктури та зберігання даних – хмарну платформу Firebase. Інформація в ній зберігається у нереляційній базі даних Cloud Firestore у вигляді колекцій та документів. Отриманим результатом є повністю робочий мобільний застосунок. Практичне значення роботи полягає у створенні масштабованої моделі соціальної мережі, що дозволяє суттєво скоротити час виходу продукту на ринок.

Основні положення і результати кваліфікаційної роботи доповідались, обговорювалися та знайшли схвалення на науково-практичній конференції: Сімнадцята Всеукраїнська, двадцять четверта регіональна наукова конференція молодих дослідників «Актуальні проблеми математики та інформатики» (Запоріжжя, 23-24 квітня 2026 р.), за результатами якої опубліковано тези «Архітектура serverless-рішень на базі Firebase у розробці високонавантажених соціальних платформ».

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, переліку посилань та 1 додатка. Загальний обсяг роботи становить 57 сторінок. Додатки викладено на 1 сторінку. У переліку посилань 15 використаних джерел.

Перший розділ присвячений аналізу предметної області та детальному обґрунтуванню вибору інструментальних засобів розробки. У другому розділі наведено процес проєктування інформаційної системи, проілюстровано архітектуру клієнт-серверної взаємодії, діаграми прецедентів, структуру бази даних та макети користувацького інтерфейсу. В третьому розділі описано безпосередню програмну реалізацію основних модулів системи, налаштування зв’язку з базою даних, а також наведено результати тестування.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ТЕХНОЛОГІЧНОГО СТЕКУ

1.1 Огляд сучасних тенденцій розвитку соціальних мереж

Аудиторія соціальних мереж сьогодні стрімко розширюється. Паралельно посилюється конкуренція між глобальними платформами, а користувачі поступово змінюють свої пріоритети: замість універсальних сервісів вони все частіше обирають нішеві екосистеми. Станом на 2025 рік понад 5,66 млрд осіб у світі (а це більш як дві третини населення планети) активно взаємодіють у цифровому просторі. Щодня з'являється близько 700 тисяч нових акаунтів, і така статистика підтверджує незмінний інтерес до віртуальної комунікації [1].

Якщо протягом останнього десятиліття домінувала концепція глобальних платформ з єдиною алгоритмічною стрічкою новин, то сьогодні спостерігається виражений тренд на децентралізацію контенту та перехід користувачів до нішових, тематичних спільнот [2]. Причиною згаданих змін дослідники називають «колапс контексту». Перетин різних соціальних груп (сім'ї, колег, малознайомих людей) в одному інформаційному полі, наприклад, Facebook змушує людину відмовлятися від публікації приватного чи оригінального контенту [3]. Тому експерти з цифрового маркетингу фіксують нову провідну тенденцію 2024-2026 років – закриті мікроспільноти набувають популярності. Замість пасивного гортання змішаної стрічки новин, учасники шукають цілеспрямованої взаємодії серед однодумців, оскільки вузькі екосистеми гарантують вищий рівень залученості [4].

Розвиток мобільних агрегаторів добре ілюструє історія мережі Amino. Тривалий час ресурс дозволяв формувати глибоко кастомізовані середовища зі своїми правилами модерації. Згодом, наприкінці 2025 року, сервіс припинив роботу, а застосунок зник із маркетплейсів [5]. Величезна аудиторія залишилася без улюбленого майданчика. Це не лише підтверджує наявність

стабільного попиту на подібні сервіси, а й звільняє нішу для нових, безпечніших та технологічно досконалих агрегаторів спільнот.

Крім базової комунікації, соціальні платформи суттєво впливають на суспільну думку, допомагають розвивати бізнес та формувати цифрову ідентичність. Понад 80% жителів України за даними 2025 року щодня застосовують їх для читання новин чи професійного спілкування [6]. З огляду на це, розробникам варто фокусуватися не просто на технічному забезпеченні обміну даними. Головним завданням стає проєктування безпечних, структурованих цифрових громад із дієвими механізмами саморегуляції.

1.2 Аналіз існуючих програмних рішень та аналогів

Для формування обґрунтованих вимог до архітектури та функціоналу соціальної мережі Minisocial було проведено порівняльний аналіз існуючих програмних рішень, які базуються на концепції агрегації спільнот. На сучасному ринку цифрових продуктів основними представниками цієї ніші є платформи Reddit, Discord, а також історичний прецедент – мобільна мережа Amino.

Reddit – одна з найбільших та найстаріших платформ-агрегаторів, яка побудована навколо системи тематичних форумів (сабреддітів). Платформа має потужну систему асинхронного споживання контенту (пости, статті, медіа) та добре розвинену ієрархію модерації всередині спільнот [7]. Ключовим недоліком Reddit у контексті сучасних вимог є глобальна система репутації «карма». Соціальний капітал користувача прив’язаний до його профілю загалом, а не до конкретної спільноти, що порушує принцип ізольованої цифрової ідентичності. Крім того, інтерфейс застосунку часто піддається критиці з боку фахівців із UX-дизайну за перевантаженість текстовою інформацією та недостатній рівень візуальної персоналізації спільнот.

Discord – платформа, що початково створювалася для геймерів, але еволюціонувала у глобальний інструмент для управління спільнотами. Продукт пропонує ідеальну ізоляцію контенту – кожен сервер є незалежною екосистемою з власними правилами, гнучкою системою ролей та управлінням доступами. До недоліків можна навести те, що архітектура Discord орієнтована переважно на синхронну комунікацію (чати в реальному часі та голосові канали). Платформі бракує повноцінної алгоритмічної стрічки для розгорнутих публікацій, що робить її незручною для авторів візуального та текстового контенту, які потребують тривалого утримання уваги аудиторії.

Amino – найближчий концептуальний аналог розроблюваної системи. Мережа функціонувала як конструктор мікроспільнот (Amino Communities). Платформа мала унікальну систему глибокої гейміфікації. Користувач мав окремий профіль для кожної спільноти, ізольовану репутацію, таблиці лідерів та власні рівні (від 1 до 20) у кожному конкретному фендомі. Незважаючи на успішну концепцію, платформа припинила своє існування у 2025 році. Головними причинами стали застарілий технологічний стек, який призводив до критичних затримок у роботі мобільного застосунку (проблеми з оптимізацією пам’яті), а також агресивна стратегія монетизації, яка зруйнувала користувацький досвід.

Для наочності результати порівняльного аналізу систем-аналогів та розроблюваного застосунку наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз

Функціональні можливості	**Reddit**	**Discord**	**Amino**	**Minisocial (проєкт)**
Формат споживання контенту	Переважно асинхронний (пости)	Переважно синхронний (чати)	Змішаний (пости + чати)	Змішаний (пости + чати)
Ізоляція профілю та репутації	Ні (глобальна карма)	Частково (локальні ролі)	Так (локальні рівні)	Так (локальні рівні)
Гнучка гейміфікація спільнот	Низька	Середня	Висока	Висока

Продовження таблиці 1.1

Функціональні можливості	**Reddit**	**Discord**	**Amino**	**Minisocial (проєкт)**
Архітектура інтерфейсу (UI)	Текстово-орієнтована	Чат-орієнтована	Візуально-орієнтована	Візуально-орієнтована
Швидкодія та сучасний стек	Середня	Висока	Низька (застаріла)	Висока (React Native + Firebase)

Проведений аналіз доводить, що на ринку існує незадоволений попит на платформу, яка поєднає зручність асинхронних публікацій, ізоляцію соціальних ролей та чати із глибокою гейміфікацією та деталізованим інтерфейсом. Враховуючи технічні помилки попередників, розроблювана система Minisocial має бути реалізована на сучасному стеку технологій для забезпечення високої реактивності інтерфейсу, надійності збереження даних та плавного користувацького досвіду без перевантаження пристрою.

1.3 Обґрунтування вибору технологічного стеку

1.3.1 Платформа Firebase (BaaS)

Сучасні соціальні мережі потребують надійної та масштабованої серверної інфраструктури. Зазвичай розробка бекенду вимагає самостійного розгортання сервера, створення REST API або GraphQL-інтерфейсів, налаштування баз даних та вебсокетів. Створення двостороннього зв’язку з нуля – об’ємне завдання. На етапі стартапу або під час написання real time застосунків подібна рутина забирає критично багато ресурсів і годин на адміністрування (DevOps). З огляду на згадані труднощі, базою для серверної частини Minisocial стала хмарна екосистема Firebase від компанії Google. Зазначений продукт втілює архітектурну концепцію Backend-as-a-Service. Завдяки делегуванню інфраструктурних завдань провайдеру, розробники

отримують можливість сфокусуватися виключно на бізнес-логіці та вдосконаленні клієнтського інтерфейсу [8]. Комплексний набір інструментів платформи покриває всі ключові потреби майбутньої мережі.

Firebase Authentication. Захист даних користувачів – базова вимога до соціальних сервісів. Створення власного модуля авторизації означає написання механізмів хешування паролів та управління сесіями або JWT-токенами з нуля. Firebase Auth натомість містить перевірені на вразливості методи автентифікації (пошта/пароль, провайдери Google, Apple). Готове рішення багаторазово знижує ризик витоку даних [9].

Cloud Firestore. Масштабована нереляційна база даних використовує документо-орієнтовану модель. Вона суттєво відрізняється від класичних SQL-баз на кшталт PostgreSQL із жорсткими таблицями. Документна структура ідеально підходить архітектурі Minisocial. Оскільки головна особливість проєкту полягає в ізольованих спільнотах, Firestore дає змогу формувати вкладені підколекції (наприклад, communities -> {community_id} -> posts -> {post_id}). Інформація кожної групи логічно розділяється без важких JOIN-запитів, здатних уповільнити систему [10].

Синхронізація в режимі реального часу. Minisocial містить чати, стрічку новин, систему сповіщень. Cloud Firestore автоматично оновлює інформацію на екранах відразу після змін у базі (працює механізм слухачів – listeners). Розробнику більше не доводиться вручну налаштовувати WebSockets чи писати логіку постійного опитування сервера (long polling) [9].

Вибір екосистеми Firebase як серверного ядра цілком обґрунтований технічно та економічно. Впровадження BaaS позбавляє потреби утримувати власні фізичні або віртуальні сервери. Документо-орієнтована база гарантує гнучкість під час створення мікроспільнот. Вбудована синхронізація в реальному часі забезпечує плавний користувацький досвід у чатах та стрічці новин.

1.3.2 Фреймворк React Native

Формат мобільного агрегатора вимагає вкрай обережного підходу до створення клієнтської частини застосунку. Сьогодні розробники зазвичай обирають між двома шляхами. Перший – нативний, коли під iOS та Android пишуть дві окремі кодові бази. Другий шлях зводиться до кросплатформності, тобто створення єдиного спільного коду для обох операційних систем. Запускати цифровий продукт на ринок потрібно швидко (мінімізуючи Time-to-Market), а ресурси на старті завжди обмежені. Саме тому клієнтську частину Minisocial вирішено будувати на кросплатформному фреймворку React Native.

Розроблений компанією Meta продукт з відкритим кодом дає змогу писати мобільні програми мовою JavaScript чи TypeScript, використовуючи добре знайому бібліотеку React. Старі гібридні технології (на кшталт Cordova або Ionic) просто малювали інтерфейс усередині вбудованого браузера. Натомість React Native працює інакше. Він транслює JS-код у справжнісінькі нативні UI-компоненти смартфона. Як наслідок, анімації та загальна продуктивність майже не поступаються класичним нативним застосункам [11].

Інтерфейс соціальної мережі завжди містить безліч деталей: екрани профілю, стрічки новин, списки груп та чати. Завдяки React Native кожен такий елемент (аватар, кнопку чи допис) можна винести в окремий незалежний компонент. Згодом ці фрагменти коду легко перевикористовувати по всьому застосунку. Так зберігається візуальна консистентність, а час написання коду суттєво скорочується.

Ізоляція контенту всередині спільнот змушує проєктувати складні маршрути. Користувачеві потрібно переходити зі списку у конкретну групу, а звідти – падати в гілку коментарів. Бібліотека React Navigation чудово закриває цю потребу. Вона ефективно керує стеками екранів, нижніми панелями та глибокими посиланнями, не гублячи при цьому стан попередніх сторінок.

Оскільки бекендом слугує саме платформа від Google, важливо мати стабільний міст між клієнтом та базою. Офіційна бібліотека React Native

Firebase надає прямий доступ до нативних SDK під iOS та Android. Відповідно, застосунок отримує надійну роботу push-сповіщень і реактивне оновлення чатів чи лайків без перезавантажень екрана [12].

Агрегатору вкрай необхідно правильно тримати в пам’яті закешовані пости, поточний рівень користувача або дані профілю. До екосистеми React без проблем підключаються сучасні менеджери стану на кшталт Zustand чи Redux Toolkit, що гарантує передбачувану поведінку інформації під час масштабування продукту. Фреймворк дозволяє миттєво бачити зміни коду на екрані смартфона чи емулятора без повної перекомпіляції проєкту. Ця функція відчутно пришвидшує етап UI/UX дизайну.

Звісно, на ринку існує потужна альтернатива – Flutter від Google. Проте перевагу віддано саме React Native через його мовну базу. Використання JavaScript/TypeScript відкриває доступ до гігантської екосистеми готових npm-пакетів. Знайти розв’язання нестандартної архітектурної задачі тут набагато простіше, ніж у випадку зі специфічною мовою Dart, яку використовує Flutter.

Проведений аналіз предметної області підтверджує: ринок дійсно має запит на сучасний мобільний агрегатор спільнот. Комбінація React Native та Firebase виглядає найбільш оптимальним технологічним фундаментом для такого проєкту. Швидкий кросплатформний клієнт у парі з надійною хмарною інфраструктурою дозволяють успішно обійти слабкі місця наявних аналогів. Завдяки цьому стеку стає можливим реалізувати ізольовані групи з глибокою гейміфікацією, причому без залучення великого штату фахівців для підтримки серверів.

1.3.3 Додаткові інструментальні засоби

Базової зв’язки React Native та Firebase часто виявляється недостатньо для стабільної роботи застосунку. Щоб продукт працював швидко, а процес написання коду не гальмувався, технологічний стек довелося розширити

вузькопрофільними інструментами. Кожен доданий сервіс вирішує конкретну архітектурну проблему Minisocial.

Класичний підхід до розробки через React Native CLI має суттєвий недолік: інженери витрачають невиправдано багато часу на конфігурацію нативних модулів в Android Studio чи Xcode. Вирішити проблему допомагає платформа Expo, яка огортає базовий фреймворк додатковими корисними інструментами [13]. Старт проєкту з нею відбувається в рази швидше. Тестувати код на реальному смартфоні дозволяє зручний клієнт Expo Go, а публікація фінальних збірок проходить без типових технічних перепон. Окремої уваги заслуговує сучасна маршрутизація Expo Router. Вона спирається на файлову систему (file-based routing). Для вкладеної архітектури Minisocial, де користувач постійно переміщується між спільнотами, чатами та профілем, цей підхід виявився найефективнішим.

Соціальна мережа безперервно оновлює дані в пам’яті: підвантажує публікації, тримає активну групу або зберігає інформацію профілю. Контролювати цей потік повинен надійний глобальний менеджер. Спільнота React зазвичай радить інтегрувати Redux, але цей інструмент має високий поріг входження і змушує писати великі масиви надлишкового коду (boilerplate) [14]. Замість нього вирішено використати більш сучасну бібліотеку Zustand. Інструмент працює на базі хуків, відрізняється мінімалізмом і діє дуже швидко. Бізнес-логіка тут легко відокремлюється від візуалу, а сам застосунок не страждає від зайвих перемальовувань екрана.

Учасники агрегатора постійно обмінюються візуальними даними, від обкладинок профілів до фотографій у дописах. Стандартний Firebase Storage вміє ці файли тільки зберігати. Якщо мобільний клієнт почне завантажувати оригінальні нестиснені картинки, виникне перевитрата трафіку, а інтерфейс стрімко втратить швидкість відгуку [15]. Щоб запобігти цьому, до системи підключили хмарний сервіс Cloudinary. Він здатен динамічно трансформувати графіку звичайними URL-параметрами. На сервер потрапляє оригінал, а на екран користувача віддається вже обрізана, оптимізована копія у сучасному форматі WebP. Як наслідок, загальна швидкодія застосунку відчутно зростає.

Інтеграція описаних сервісів остаточно формує надійний фундамент для клієнтської частини. Платформа Expo бере на себе комфортну маршрутизацію та збірку. Zustand закриває питання легковагового управління станом. Своєю чергою, Cloudinary рятує інтерфейс від зависань під час обробки важких зображень. У комплексі такий набір технологій дозволяє розробити дійсно швидкий та конкурентоспроможний програмний продукт.

1.4 Висновки до розділу 1

Детальне дослідження предметної області виявило стрімку зміну пріоритетів сучасної аудиторії. Замість глобальних платформ із єдиною алгоритмічною стрічкою новин користувачі дедалі частіше обирають закриті нішеві мікроспільноти. Оцінюючи провідні ринкові аналоги, вдалося підтвердити наявність незадоволеного попиту на універсальний мобільний агрегатор. Комерційно успішний продукт повинен гармонійно поєднувати асинхронні публікації, мультимедійні чати, ізольовані соціальні профілі та розвинену систему гейміфікації. Задля успішної реалізації окреслених завдань було сформовано оптимальний технологічний стек. Функції серверної інфраструктури виконуватиме хмарна екосистема Firebase (за моделлю Backend-as-a-Service). Відмова від самостійного розгортання бекенду суттєво пришвидшує загальний процес розробки. Водночас інтеграція нереляційної документо-орієнтованої бази даних Cloud Firestore гарантує безперебійне збереження інформації всередині ізольованих груп із миттєвою синхронізацією контенту.

Основою клієнтського застосунку виступив кросплатформний фреймворк React Native разом з інструментарієм Expo. Комбінування згаданих технологій зменшує час виходу продукту на ринок, дозволяючи паралельно генерувати продуктивні збірки під різні платформи. Уникаючи надмірного споживання оперативної пам’яті, глобальним станом програми ефективно

керує мінімалістичний менеджер Zustand. Паралельно проблему перевантаження мережі важким мультимедіа вирішує підключений сервіс оптимізації зображень Cloudinary.

У комплексі затверджений набір інструментів створює надійний фундамент майбутньої платформи, готової до високих навантажень та горизонтального масштабування. Завершивши огляд предметної області та остаточно визначивши технологічну базу, логічним продовженням роботи стає безпосереднє проєктування архітектури застосунку. Наступний етап дослідження включатиме формалізацію функціональних вимог, побудову UML-діаграм прецедентів, детальне моделювання структури бази даних.

2 СИСТЕМНИЙ АНАЛІЗ ТА ПРОЄКТУВАННЯ АРХІТЕКТУРИ ЗАСТОСУНКУ

2.1 Специфікація та аналіз вимог до системи

Проєктування архітектури застосунку логічно розпочати з визначення базових вимог до системи. Вони ґрунтуються на результатах аналізу предметної області та враховують недоліки існуючих аналогів, які були розглянуті в попередньому розділі. Сформований перелік дозволяє окреслити межі мінімально життєздатного продукту. Загалом, функціональні вимоги спрямовані на вирішення ключової задачі – створення ізольованих спільнот із вбудованою гейміфікацією, тоді як нефункціональні гарантують стабільність застосунку та його готовність до подальших навантажень.

2.1.1 Функціональні вимоги

Функціональні вимоги визначають, що саме має робити застосунок та як користувач буде з ним взаємодіяти. Основний функціонал розділено на кілька логічних блоків.

Управління обліковими записами:

– реєстрація та авторизація користувачів (через email/пароль або соціальні провайдери) за допомогою Firebase Authentication;
– можливість відновлення доступу до облікового запису;

Навігація та пошук:

– перегляд глобального списку доступних мікроспільнот;
– пошук спільнот за назвою або тематикою;
– створення власної спільноти;

Ізольована взаємодія у спільнотах:

– вступ користувача до обраної спільноти та створення унікального

локального профілю (нікнейм, біографія, аватар), який діє лише в її межах;

– читання, створення, редагування та видалення публікацій (дописів) усередині конкретної спільноти;
– взаємодія з контентом: система вподобань (лайків) та багаторівневих коментарів;
– участь у текстових чатах спільноти в режимі реального часу;

Гейміфікація та ролі:

– система нарахування балів репутації за активність (публікації, коментарі, отримані лайки);
– відображення поточного «рівня» користувача в його локальному профілі;
– розподіл ролей у спільноті (Учасник, Модератор, Лідер/Адміністратор) із відповідними правами доступу до управління контентом;

Управління медіаконтентом: завантаження зображень для аватарів, обкладинок спільнот та публікацій.

2.1.2 Нефункціональні вимоги

На відміну від функціональних вимог, цей перелік визначає якісні характеристики системи – тобто те, як саме вона повинна працювати, щоб користуватися нею було комфортно.

Продуктивність та оптимізація: застосунок має забезпечувати плавне відображення інтерфейсу (цільовий показник – 60 кадрів на секунду) завдяки використанню фреймворку React Native, а завантаження та обробка медіафайлів має відбуватися з мінімальною затримкою, для чого система повинна використовувати динамічні трансформації зображень Cloudinary (стиснення, зміна розміру) для економії мобільного трафіку користувачів.

Надійність та управління станом: для запобігання втраті даних та забезпечення швидкого відгуку інтерфейсу система повинна ефективно керувати глобальним станом (дані профілю, закешовані публікації) за допомогою бібліотеки Zustand, уникаючи зайвого рендерингу компонентів.

Масштабованість: архітектура бази даних Cloud Firestore повинна бути спроєктована таким чином, щоб система могла підтримувати збільшення кількості користувачів та спільнот без деградації швидкості виконання запитів (читання/запис).

Кросплатформеність: завдяки використанню інструментарію Expo клієнтський код має компілюватися та коректно працювати на основних мобільних платформах, маючи ідентичну бізнес-логіку та візуальну консистентність.

Безпека: доступ до бази даних (читання та запис) має бути суворо регламентований за допомогою правил безпеки Firebase (Firestore Security Rules), щоб унеможливити несанкціоновану зміну чужого контенту або штучне завищення репутації.

2.2 Проєктування діаграм варіантів використання (Use Case)

Концептуальне проєктування інформаційної системи передбачає визначення меж системи, її основних користувачів та сценаріїв їхньої взаємодії з програмним продуктом.

Для формалізації цих вимог використовується метод об’єктно-орієнтованого моделювання за допомогою діаграм варіантів використання мови UML.

Оскільки ключовою архітектурною особливістю платформи Minisocial є ізоляція контенту та використання системи контролю доступу на основі ролей (Role-Based Access Control, RBAC), визначення ієрархії акторів є дуже важливим етапом проєктування.

У розробленій системі виділено п’ять основних типів користувачів (акторів), які мають різний рівень доступу до функціоналу:

– *неавторизований користувач*: користувач, який щойно завантажив застосунок, його єдиним прецедентом є проходження процедури реєстрації та створення глобального облікового запису (Global Account) за допомогою провайдера авторизації Google;
– *учасник спільноти*: базовий авторизований користувач, який має глобальний профіль та створив локальні профілі у вибраних спільнотах;
– *модератор спільноти*: користувач із розширеними правами всередині конкретної мікроспільноти, призначений для контролю за дотриманням правил;
– *лідер спільноти*: власник мікроспільноти, який володіє повним набором прав для управління її внутрішньою екосистемою;
– *глобальний адміністратор*: розробник або власник платформи, який управляє глобальними налаштуваннями застосунку та не обмежений рамками окремих спільнот.

Розподіл прецедентів між акторами демонструє чітку ієрархію: кожен наступний рівень доступу (від «Учасника» до «Лідера») успадковує всі базові можливості попереднього, додаючи специфічні функції модерації. Графічне відображення взаємодії базового користувача та модераторів з основною системою наведено на рисунку 2.1.

Модуль глобальної взаємодії:

– перегляд глобальної стрічки: доступно всім авторизованим користувачам для перегляду закріплених публікацій зі спільнот, у яких вони перебувають;
– пошук та фільтрація спільнот: взаємодія з каталогом за назвою або глобальними категоріями;
– управління глобальним профілем: редагування загального нікнейму, аватара та налаштувань сповіщень.

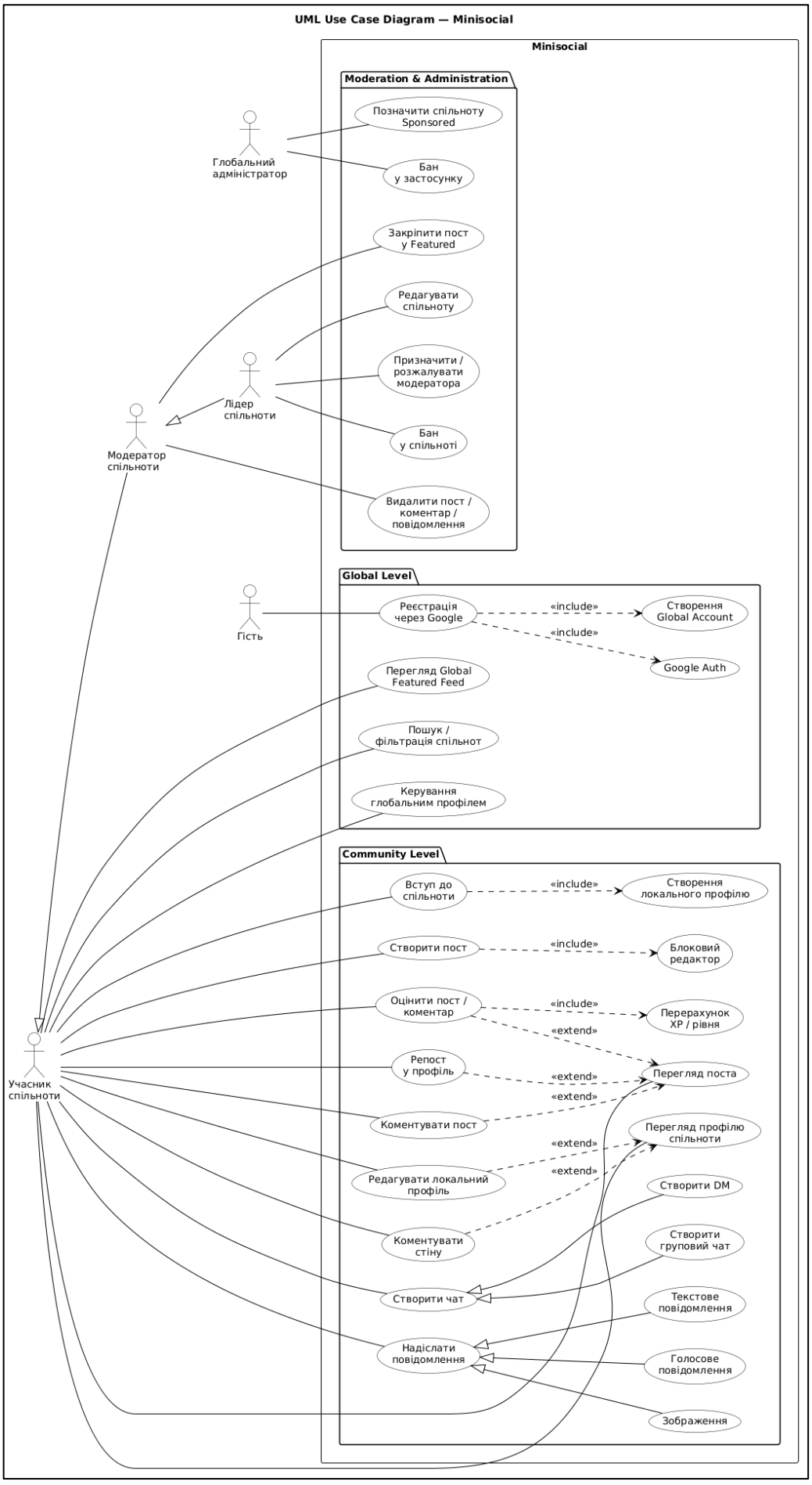


Рисунок 2.1 – Діаграма варіантів використання

Модуль локальної взаємодії у спільнотах:

– вступ до спільноти: створення унікального локального профілю для взаємодії всередині ізольованої групи;
– створення контенту: використання блокового редактора для створення публікацій із чергуванням текстових та медіа-блоків;
– соціальна взаємодія: оцінювання публікацій та коментарів (що ініціює перерахунок очок досвіду – XP та рівня користувача за спеціальною формулою), залишення коментарів на стінах інших користувачів;
– комунікація: створення особистих (DM) та групових чатів, надсилання текстових повідомлень, зображень та голосових повідомлень.

Модуль управління та модерації:

– управління контентом («Модератор» / «Лідер»): видалення публікацій і повідомлень звичайних учасників, закріплення публікацій у вкладці Featured;
– управління екосистемою («Лідер»): редагування опису та обкладинки спільноти, призначення та розжалування модераторів спільноти.
– глобальне управління («Глобальний адміністратор»): призначення статусу «Спонсорська спільнота» для відображення на головному екрані застосунку.

2.3 Моделювання структури нереляційної бази даних

Для забезпечення зберігання та обробки даних розроблюваної соціальної мережі було обрано нереляційну документо-орієнтовану базу даних Cloud Firestore. На відміну від класичних реляційних баз даних, які використовують жорстко структуровані таблиці, зв’язані зовнішніми ключами, Firestore зберігає дані у вигляді колекцій та документів формату

JSON. Це ідеально відповідає архітектурі ізольованих спільнот, забезпечуючи високу швидкість читання даних та горизонтальну масштабованість. Концептуальну модель сутностей зображено на рисунку 2.2.

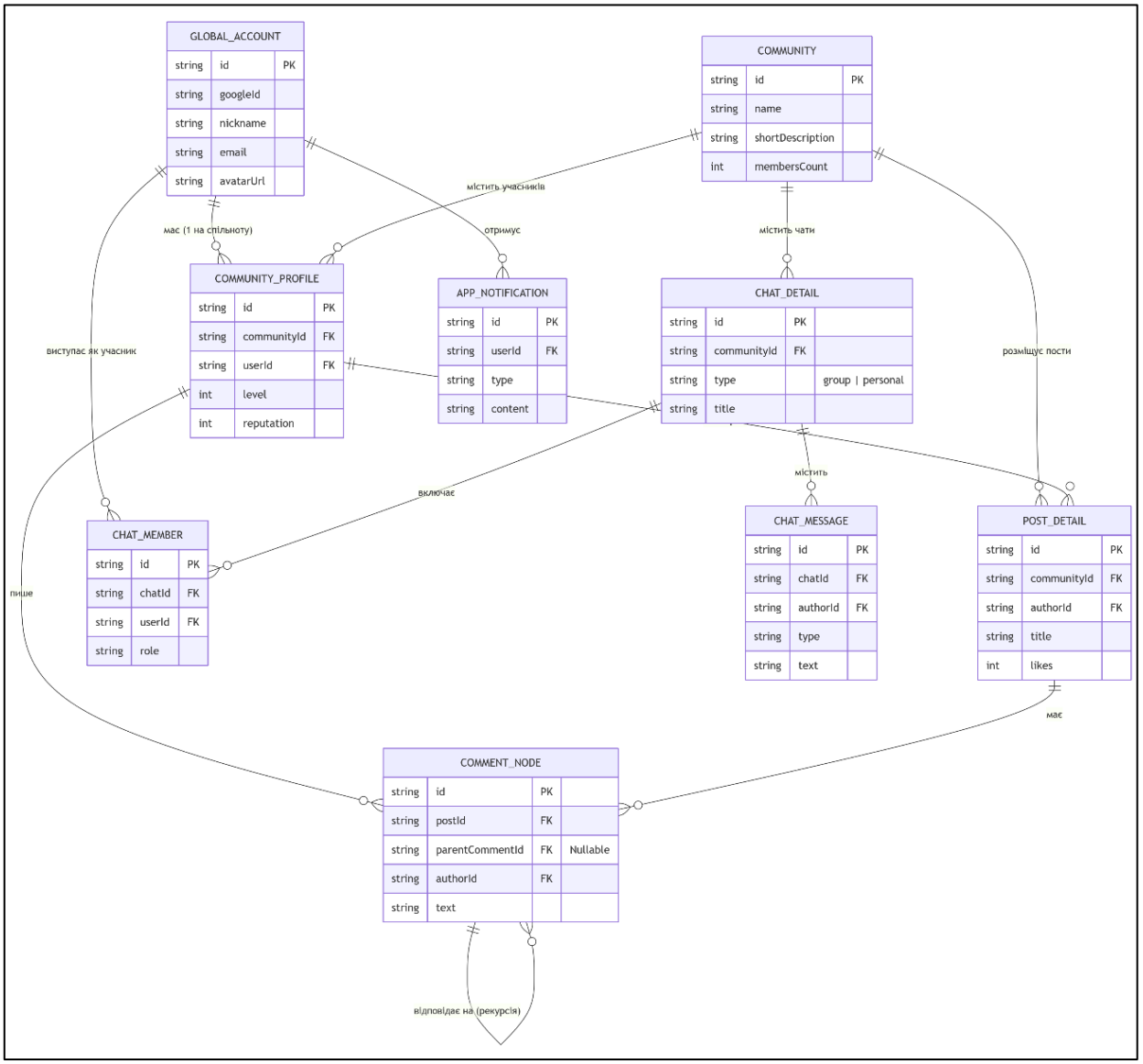


Рисунок 2.2 – Концептуальна модель зв’язків між сутностями системи

На етапі логічного проєктування було розроблено детальну схему бази даних, що відображає сутності системи, їхні атрибути (поля) та типи даних. Розглянемо основні колекції, з яких складається логічна структура бази даних.

Колекція GLOBAL_ACCOUNT зберігає базову інформацію про користувача, яка є незмінною для всіх спільнот і створюється під час первинної реєстрації (OAuth через Google):

– id (string) – унікальний ідентифікатор документа (Primary Key);
– googleId, email (string) – ідентифікаційні дані провайдера;
– nickname, avatarUrl (string) – глобальні публічні дані користувача;
– notificationsEnabled (boolean) – глобальні налаштування застосунку.

Колекція COMMUNITY відповідає за зберігання метаданих кожної окремої мікроспільноти:

– id (string), name, shortDescription, fullDescription (string) – основна інформація;
– coverUrl, icon (string) – посилання на медіафайли у хмарному сховищі Cloudinary;
– tags (string[]) – масив тегів для глобального пошуку;
– members (int) – лічильник кількості учасників (для оптимізації запитів);
– isSponsored (boolean) – прапорець, що встановлюється глобальним адміністратором.

Колекція COMMUNITY_PROFILE реалізує концепцію «ізоляції контенту». Кожен документ є унікальним профілем користувача всередині конкретної спільноти:

– id (string) – PK;
– communityId, userId (string) – посилання на відповідну спільноту та глобальний акаунт;
– nickname, avatarUrl (string) – локальні візуальні дані;
– level, reputation, postsCount (int) – локальна статистика та гейміфікація (XP система);
– about (json: array_aboutBlocks) – складний об’єкт, що містить блоки з текстом та зображеннями для детального опису профілю;
– isOnline (boolean) – статус користувача для відображення у списках чатів.

Колекція POST_DETAIL зберігає дописи користувачів. Оскільки пости створюються через спеціальний блоковий редактор, їхній контент має складну структуру:

– id, communityId, authorProfileId (string);
– content (json) – масив блоків (текстових та графічних), що дозволяє гнучко рендерити пост на клієнті;
– likes (int) – лічильник вподобань;
– isFeatured (boolean) – прапорець для закріплення посту лідерами/модераторами на головній сторінці спільноти.

Підсистема обміну повідомленнями нормалізована на три пов’язані колекції для забезпечення ефективного читання історії в реальному часі:

– CHAT_DETAIL: зберігає тип чату (group або personal), назву, обкладинку та закріплені повідомлення;
– CHAT_MEMBER: пов’язує локальний профіль із конкретним чатом, зберігаючи його роль (role: CommunityRole);
– CHAT_MESSAGE: зберігає безпосередньо повідомлення (type: text | audio | image | system). Поля text, image та audioDuration є опціональними залежно від типу повідомлення.

Колекція COMMENT_NODE реалізує древоподібну структуру обговорень під публікаціями або на стіні користувача:

– id, postId (string);
– parentCommentId (string, Nullable) – ідентифікатор батьківського коментаря, що дозволяє вибудовувати нескінченну вкладеність відповідей.

Детальна діаграма бази даних із зазначенням типів полів та логічних зв’язків між колекціями наведена у Додатку А.

Спроєктована структура бази даних Cloud Firestore повністю відповідає вимогам специфікації.

Використання денормалізованих даних є стандартною та необхідною практикою для NoSQL рішень. Це дозволяє уникнути складних операцій об’єднання на клієнті та забезпечити максимальну швидкість відображення стрічки новин і чатів.

2.4 Висновки до розділу 2

Розпочавши з детального системного аналізу, у межах другого розділу вдалося успішно спроєктувати архітектуру майбутнього мобільного застосунку Minisocial. Спираючись на результати попереднього дослідження предметної області, сформовано вичерпний перелік функціональних та нефункціональних вимог, які чітко окреслили межі запланованого продукту. Особливу увагу під час моделювання приділено забезпеченню повної ізольованості мікроспільнот, досягненню високої швидкодії клієнтської частини та загальній масштабованості системи.

Формалізувати сценарії взаємодії користувачів із програмним продуктом дозволила побудова UML-діаграм варіантів використання. Завдяки розробці детальної ієрархії акторів з'явилася можливість закласти надійний фундамент для впровадження гнучкого контролю доступу на основі ролей. Шляхом реалізації зазначеного розмежування було досягнуто безпечного розділення глобального простору платформи та локального вмісту всередині кожної окремої групи.

Надзвичайно важливим етапом роботи виявилося моделювання структури бази даних. Орієнтуючись на специфіку затвердженого технологічного стеку, спроєктовано денормалізовану схему для документо-орієнтованої СУБД Cloud Firestore. Повністю закриває архітектурну потребу в ізоляції контенту логічний розподіл інформаційних масивів на окремі колекції, паралельно гарантуючи максимальну швидкість рендерингу даних.

Надійну основу для переходу до безпосереднього написання програмного коду заклало успішне завершення концептуального та логічного проєктування. Логічним продовженням розвідки у межах третього розділу стане опис практичної реалізації клієнтської частини застосунку, верстки інтерфейсу користувача, налаштування зв'язку з хмарними сервісами Firebase та проведення комплексного функціонального тестування готової системи.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Архітектура клієнтського застосунку, управління станом та інтеграція хмарних сервісів

Створення клієнтської частини соціальної мережі вимагало впровадження актуальних інженерних патернів, адже мобільні платформи вкрай чутливі до оптимізації коду. Щоб уникнути структурного хаосу під час подальшого масштабування продукту, архітектуру застосунку логічно розділили на три незалежні шари. Йдеться про рівень маршрутизації (суміщений із користувацьким інтерфейсом), рівень управління станом та, найголовніше, рівень предметної області (Domain Layer). Фундаментом для всієї розробки стала мова TypeScript: її сувора типізація допомагає виловлювати більшість критичних помилок ще до етапу компіляції програми.

Навігацію між екранами забезпечує бібліотека Expo Router, яка пропонує сучасну файлову маршрутизацію. Роль своєрідного диспетчера бере на себе кореневий макет app/_layout.tsx. Зазначений компонент діє як глобальний ініціалізатор і водночас охороняє приватні маршрути (Auth Guard).

На практиці алгоритм працює цілком прямолінійно: під час старту застосунок миттєво перевіряє статус авторизації. Неавторизований гість не зможе потрапити до внутрішніх розділів, оскільки система примусово перенаправить його на сторінку входу. Паралельно згаданий макет безперервно відстежує активність пристрою, щоб вчасно оновлювати онлайн-статус юзера у хмарній базі даних. Фрагмент коду із логікою захисту маршрутів ілюструє рисунок 3.1.

Специфіка соціальної мережі передбачає безперервний потік динамічної інформації: стрічки новин постійно оновлюються, приходять свіжі повідомлення, змінюються списки спільнот. Контролювати ці процеси довірили мінімалістичній бібліотеці Zustand.

```
React.useEffect(() => {
  if (bootStatus !== ‘ready’) return;
  const inAuthScreen = segments[0] === ‘auth’;
  // Перенаправлення неавторизованого користувача
  if (!isAuthed && !inAuthScreen) {
    router.replace(‘/auth’);
    return;
  }
  // Перенаправлення авторизованого користувача на головний екран
  if (isAuthed && inAuthScreen) {
    router.replace(‘/(tabs)’);
  }}, [bootStatus, isAuthed, router, segments]);
```

Рисунок 3.1

Замість одного масивного монолітного сховища застосовано модульну архітектуру. Кожен логічний блок виокремлюється у власний «слайс». Завдяки подібному розмежуванню елементи інтерфейсу зчитують виключно потрібні їм фрагменти інформації і не перемальовуються без нагальної потреби. Цикл взаємодії відбувається так: інтерфейс звертається до методу слайсу, той надсилає запит до репозиторію (реальної бази Firebase або локальної заглушки), після чого успішно оновлює глобальний стан. Практичну реалізацію механізму приєднання до групи можна побачити на рисунку 3.2.

```
export function createCommunitiesSlice(initialSnapshot: AppBootstrapSnapshot):
AppSliceCreator<CommunitiesSlice> {
  return (_set, get) => ({
    communities: initialSnapshot.communities,
    async joinCommunity(communityId) {
      const runtime = await getRepositoryRuntime();
      await runtime.repositories.communities.joinCommunity(communityId);
      await get().refreshFromRepositories();} // Оновлення глобального стану
  });}
```

Рисунок 3.2

З інженерного погляду, найважливішим кроком стало винесення «чистої» бізнес-логіки в ізольований предметний шар (відповідні модулі domain/usecases.ts та domain/models.ts).

Компоненти інтерфейсу повинні займатися виключно відмальовуванням пікселів на екрані, а не складними математичними обчисленнями. Саме тому всередину предметного шару перемістили ключові алгоритми платформи. Серед них:

– процедура розрахунку досвіду (функція calculateXp), яка трансформує кількість отриманих вподобань у відповідний ранг і колір бейджа;
– сувора система контролю доступу (RBAC), де функція getCommunityPermissions розрізняє повноваження звичайних учасників, модераторів та лідерів;
– механізми глибокого парсингу контенту, включно з вилученням тексту з блокових структур чи ієрархічним сортуванням користувачів.

Принцип перевірки адміністративних прав доступу детально проілюстровано на рисунку 3.3.

```
export function getCommunityPermissions(role: CommunityRole, isOwner: boolean) {
  if (role === ‘leader’ || isOwner) {
    return {
      canFeature: true,
      canDeleteAnyPost: true,
      canDeleteAnyMessage: true,
      canManageModerators: true,
      canBanMembers: true,
    };
  }
  // ... логіка для інших ролей
}
```

Рисунок 3.3

Додатково варто зазначити важливу роль контрактів та строгих інтерфейсів, якими покрита вся кодова база. Класичний CommunityRepository жорстко диктує перелік методів для роботи зі спільнотами. Через це систему абсолютно не хвилює, звідки саме надходять дані – підключити іншу базу чи змінити провайдера в майбутньому не складе жодних труднощів.

Сформована архітектура чудово витримує навантаження і легко масштабується. Бібліотека Expo Router бере на себе стабільну маршрутизацію, Zustand ефективно розвантажує оперативну пам’ять пристрою, а виділення бізнес-логіки в незалежний Domain Layer дозволяє безболісно розширювати функціонал соціальної мережі та писати для неї надійні автоматизовані тести.

Інтеграція хмарних сервісів та патерн «Репозиторій». Забезпечення гнучкості, стійкості до мережевих збоїв та високої продуктивності стає першочерговим завданням під час оперування даними у сучасній мобільній розробці. Задля побудови надійної серверної частини платформи Minisocial реалізовано концепцію Backend-as-a-Service ресурсами хмари Firebase, паралельно інтегрувавши спеціалізований сервіс Cloudinary з метою оптимізації обробки мультимедійного контенту. Захистити клієнтську кодову базу від жорсткої прив’язки до певного сховища дозволяє архітектурний патерн «Репозиторій», покладений в основу проєктування рівня доступу до даних. Завдяки взаємодії винятково через абстрактні інтерфейси (контракти) з’являється можливість безболісно підміняти джерела інформації, не порушуючи функціонування базової бізнес-логіки застосунку.

У межах згаданого архітектурного шару головна роль належить модулю середовища виконання (runtime.ts), відповідальному за адаптивне перемикання сховищ (RuntimeDataSource). Одразу в момент запуску програми починається спроба ініціалізації нативних плагінів Firebase. За умов відсутності валідної конфігурації або у разі виникнення мережевих помилок відбувається автоматичне переспрямування запитів на локальні тимчасові масиви (Mock Data). Завдяки сформованому алгоритму гарантується безперебійне функціонування інтерфейсу користувача, що дозволяє

повноцінно тестувати візуальні елементи навіть за повної відсутності інтернет-з’єднання. Схематично логіку ініціалізації та гнучкого перемикання інформаційних джерел відображає фрагмент коду, наведений на рисунку 3.4.

```
let dataSource = resolveRequestedDataSource();
if (dataSource === ‘firebase’) {
  try {
    getFirebaseAppInstance();
    getFirebaseAuthInstance();
    getFirebaseFirestoreInstance();
  } catch (error) {
    // Автоматичний перехід на mock-дані у разі помилки конфігурації
    dataSource = ‘mock’;
    fallbackReason = error instanceof Error
      ? `Firebase initialization failed... ${error.message}`
      : ‘Firebase initialization failed.’;
  }
}
```

Рисунок 3.4 – Логіка ініціалізації джерела даних

Додатково в середовищі виконання реалізовано патерн «Декоратор» (зокрема, функцію wrapBundle), покликаний автоматично перехоплювати запити, спрямовані на генерацію дописів чи оновлення акаунтів. Після автоматичного вивантаження локальних медіафайлів у хмарне сховище система транслює отримані віддалені URL-адреси безпосередньо до головної бази даних. У межах файлу firebaseBundle.ts зосереджено базовий набір репозиторіїв, призначених для оперування реальними хмарними структурами. До складу розроблених скриптів увійшла імплементація механізмів автентифікації (AuthRepository), взаємодії всередині мікроспільнот, а також інструменти створення постів, гілок обговорення та чатів. Через нереляційну специфіку сховища Cloud Firestore виникає необхідність синхронного оновлення денормалізованих полів у межах кількох суміжних колекцій

(зокрема, підрахунку кількості читачів під час оформлення підписки або коригування лічильника лайків). Зберегти цілісність інформаційних масивів допомагає активне впровадження інструментів пакетного запису (writeBatch) та атомарних транзакцій (runTransaction). Принцип забезпечення транзакційної цілісності під час оцінювання матеріалу користувачами демонструє фрагмент коду, наведений на рисунку 3.5.

```
await runTransaction(database, async (transaction) => {
  const likeSnap = await transaction.get(likeRef);
  const postSnap = await transaction.get(postRef);
  if (!postSnap.exists()) return;
  // Якщо лайк вже існує – видаляємо його та зменшуємо лічильник
  if (likeSnap.exists()) {
    transaction.delete(likeRef);
    transaction.update(postRef, { likes: increment(-1), updatedAt: serverTimestamp() });
  } else {
    // Якщо лайка немає – створюємо документ та збільшуємо лічильник
    transaction.set(likeRef, { userId: user.uid, createdAt: serverTimestamp() });
    transaction.update(postRef, { likes: increment(1), updatedAt: serverTimestamp() });
  }
});
```

Рисунок 3.5 – Реалізація транзакційного лічильника вподобань

У контексті обробки графічних об’єктів (аватарів, фонових обкладинок та медіа-вкладень) було реалізовано інтеграцію із сервісом Cloudinary. Шляхом наслідування контракту ImageUploadService, клас cloudinaryImageUploadService.ts здійснює попередню підготовку файлів та ініціює відправку захищених POST-запитів через об’єкт FormData (див. рис. 3.6). За умов відсутності зв’язку із сервісом або у випадках, коли картинка вже має валідне віддалене посилання, логіка перемикається на резервну заглушку passthroughImageUploadService. Завдяки згаданому рішенню потрібні ідентифікатори генеруються без повторного вивантаження файлів у мережу, що значно заощаджує мобільний трафік користувачів.

```
export const cloudinaryImageUploadService: ImageUploadService = {
  async prepareImageAsset({ uri, purpose }): Promise<UploadedImageAsset> {
    // ... логіка валідації
    const endpoint =
`https://api.cloudinary.com/v1_1/${appEnv.cloudinary.cloudName}/image/upload`;
    const response = await fetch(endpoint, {
      method: 'POST',
      body: buildUploadFormData(trimmedUri, purpose), // Формування FormData
    });
    if (!response.ok) {
      throw new Error(`Cloudinary upload failed (${response.status}).`);
    }
    const payload = await response.json() as CloudinaryUploadResponse;
    return { url: payload.secure_url ?? payload.url, publicId: payload.public_id };
  },
};
```

Рисунок 3.6

Комбінація архітектурного патерну «Репозиторій» з адаптивним модулем виконання (runtime.ts) забезпечила максимальну надійність усього застосунку. Завдяки потенціалу Firebase вдалося налагодити безперебійну автентифікацію та транзакційну стабільність сховища, тоді як інтеграція Cloudinary дозволила оптимізувати оперування важкими файлами, знявши зайве навантаження з основної інфраструктури системи.

3.2 Проєктування UI/UX та програмна реалізація ключових інтерфейсних модулів

Базовим інструментом для створення клієнтської частини соціальної мережі став фреймворк React Native. Однак стандартних засобів виявилося замало задля втілення специфічних візуальних задумів (зокрема, ефектів

Glassmorphism та Neumorphism). Архітектуру довелося розширити спеціалізованими бібліотеками react-native-reanimated разом із expo-blur, завдяки яким анімації відтворюються максимально плавно. Під час розробки інтерфейсу виникала потреба проєктувати унікальні віджети, паралельно долаючи платформозалежні обмеження. Одною з найпомітніших перепон стала непередбачувана поведінка клавіатури всередині операційної системи Android.

Концепція Neumorphism базується на імітації об'ємних деталей – опуклих або втиснутих поверхонь. У процесі тестування з'ясувалася нездатність базових методів React Native однаково якісно генерувати тіні для різних платформ. Універсальний компонент NeumorphBox було створено саме для обходу згаданих технічних бар'єрів. Модуль автоматично аналізує середовище виконання та активує найбільш доречний алгоритм рендерингу. Наприклад, екрани пристроїв Apple бездоганно обробляють подвійні тіні через параметри shadowColor і shadowOffset. Системи Android навпаки жорстко обмежують підтримку кольорових тіней, вимагаючи альтернативного підходу: комбінації властивості elevation зі спрямованим тонуванням границь. Розроблений віджет підтримує два фізичні стани (embossed та inset), миттєво змінюючи форму під час торкання користувача. Детальну програмну логіку компонента наведено на рисунку 3.7.

Окремої уваги потребувала маршрутизація всередині ізольованих спільнот. Спеціальний компонент CommunityTabBar успішно взяв на себе роль локального навігатора. Додавання елемента BlurView дозволило зімітувати естетичне матове скло (стиль Glassmorphism). Водночас за плавне переміщення індикатора активної вкладки відповідає пружинна анімація (метод withSpring), імпортована з пакета react-native-reanimated.

Формування стрічки новин відійшло від формату монолітних текстових полів. Замість них публікація збирається з масиву типізованих блоків (графіки або тексту). Безпосереднім рендерингом займається модуль ContentBlocks. Отримавши на вхід колекцію об'єктів формату ContentBlock, функція

ітеративно розгортає правильну послідовність компонентів Text чи Image. Запропонована структура суттєво підвищує візуальну гнучкість, надійно оберігаючи верстку від збоїв. Додатково з’являється простір для легкого масштабування, адже згодом систему можна розширити відеопрогравачами або опитуваннями.

```
if (isAndroid) {
  content = (
    <View style={[
      !isInset && {
        elevation: 4,
        borderWidth: 1.5,
        borderTopColor: ‘rgba(255,255,255,0.7)’,
        borderBottomColor: ‘rgba(174,174,192,0.25)’,
      },
      isInset && { borderWidth: 1, borderColor: ‘rgba(174,174,192,0.15)’ }
    ]}>
      {children}
    </View>
  );
} else {
  // Реалізація через подвійні тіні (shadowColor) для iOS
}
```

Рисунок 3.7

Інженерна складність екрана повідомлень дуже потужна. Тут одночасно відбувається рендеринг історії діалогів, моніторинг статусів прочитання та динамічний розрахунок рівня співрозмовника. Можливість створювати голосові замітки з’явилася після інтеграції бібліотеки expo-audio. Прагнучи покращити досвід користування, додано сучасний жест «Slide to cancel» за допомогою обробника подій PanResponder. Відповідний алгоритм безперервно зчитує координати пальця вздовж осі X під час утримання кнопки запису. Зміщення вліво понад мінус 160 пікселів миттєво скасовує дію, супроводжуючи процес відчутною вібрацією пристрою (рис. 3.8).

```
const panResponder = useRef(
  PanResponder.create({
    onPanResponderMove: (e, gestureState) => {
      if (gestureState.dx < 0) {
        const dragX = Math.max(gestureState.dx, -160);
        panX.setValue(dragX);
        if (dragX <= -130 && !isCanceledRef.current) {
          isCanceledRef.current = true;
          Vibration.vibrate(100); // Відгук при досягненні порогу скасування
        }
      }
    },
    onPanResponderRelease: async () => {
      setIsRecording(false);
      await stopRecording(isCanceledRef.current); // Збереження або скасування
    }
  })
).current;
```

Рисунок 3.8

Відтворенням аудіофайлів керує ізольований компонент AudioBubble. Всередині нього інкапсульована логіка плеєра useAudioPlayer, тоді як хуки useSharedValue разом із useAnimatedStyle гарантують високу частоту кадрів під час оновлення смуги прогресу. Основна перевага обраного підходу в делегуванні обчислень нативному потоку UI, зберігаючи ресурси JavaScript-ядра.

Під час тестування чатів виник баг, характерний винятково для операційної системи Android. Ідеться про збій режиму adjustResize, коли після згортання програми віртуальна клавіатура починає перекривати поле вводу. Усунути недолік допомогла кастомна система слухачів, де події клавіатури Keyboard.addListener перетинаються з моніторингом статусу застосунку AppState.addEventListener. Завдяки написаному скрипту система вміє вираховувати різницю висот екрана, миттєво додаючи необхідний відступ знизу. Користувач завжди бачить зону введення тексту незалежно від поведінки системних вікон.

Підсумовуючи, глибоке опрацювання клієнтських модулів сформувало надійний та естетичний програмний продукт. Безперервну візуальну

консистентність на різних пристроях підтримують авторські віджети. Водночас відмова від стандартних текстових полів на користь блокового редактора та складні механіки всередині мультимедійних чатів підтверджують готовність платформи до значних навантажень.

Створений програмний базис та розроблені кастомні компоненти дозволили перейти до фінального етапу – збирання та візуального оформлення користувацького інтерфейсу (UI/UX) застосунку Minisocial.

Розробка зручного та зрозумілого інтерфейсу – один із найважливіших етапів створення соціальної мережі, адже від цього залежить, чи залишиться користувач у застосунку. Візуальне оформлення Minisocial спирається на сучасні підходи: стилістику неоморфізму з використанням м’яких тіней та елементи матового скла. Таке поєднання додає інтерфейсу об’єму і не відвертає увагу від головного – самого контенту.

Оскільки архітектура застосунку має два рівні (глобальний і локальний), навігацію довелося логічно розділити. Головний екран працює як глобальний агрегатор (див. рис. 3.9).

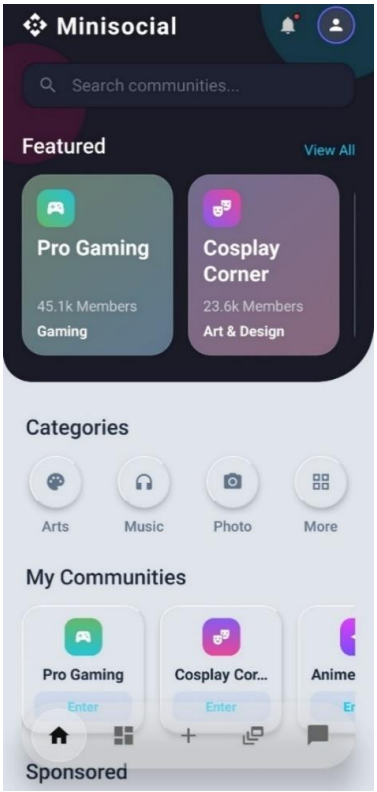
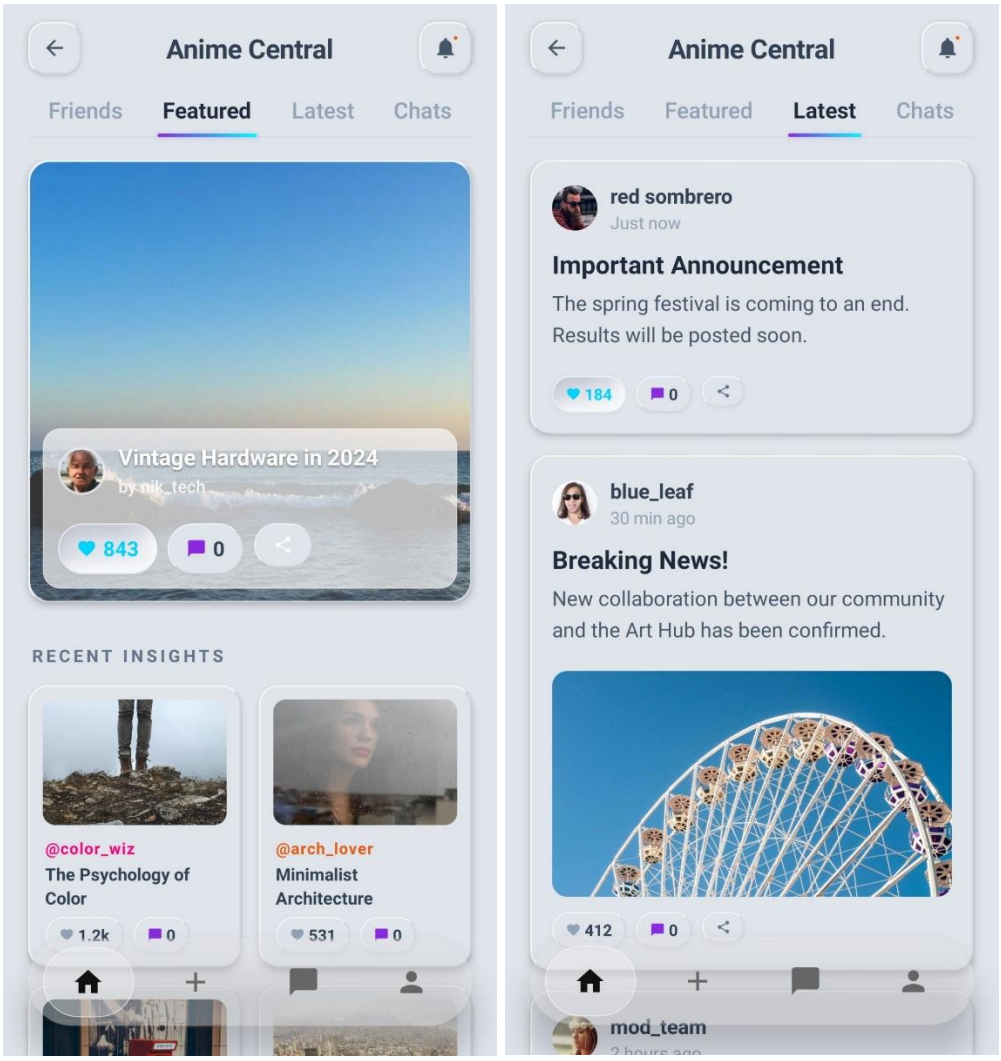


Рисунок 3.9 – Головний екран застосунку

Зверху тут розміщено рядок пошуку для швидкого знаходження потрібних груп. Нижче реалізовані горизонтальні каруселі з рекомендованими спільнотами та категоріями, а окремим блоком виведено список тих груп, до яких користувач вже приєднався. Нижня панель навігації на цьому етапі дає доступ до загальних розділів платформи: головної сторінки, каталогу, створення нової спільноти, загальної стрічки постів та списку чатів.

Коли ж користувач переходить у конкретну спільноту, інтерфейс повністю змінюється, щоб підкреслити ізольованість цього середовища (див. рис. 3.10). Нижня панель навігації підлаштовується під локальний рівень: з’являються кнопки стрічки новин, створення публікації, місцевих чатів та профілю учасника саме в цій спільноті.



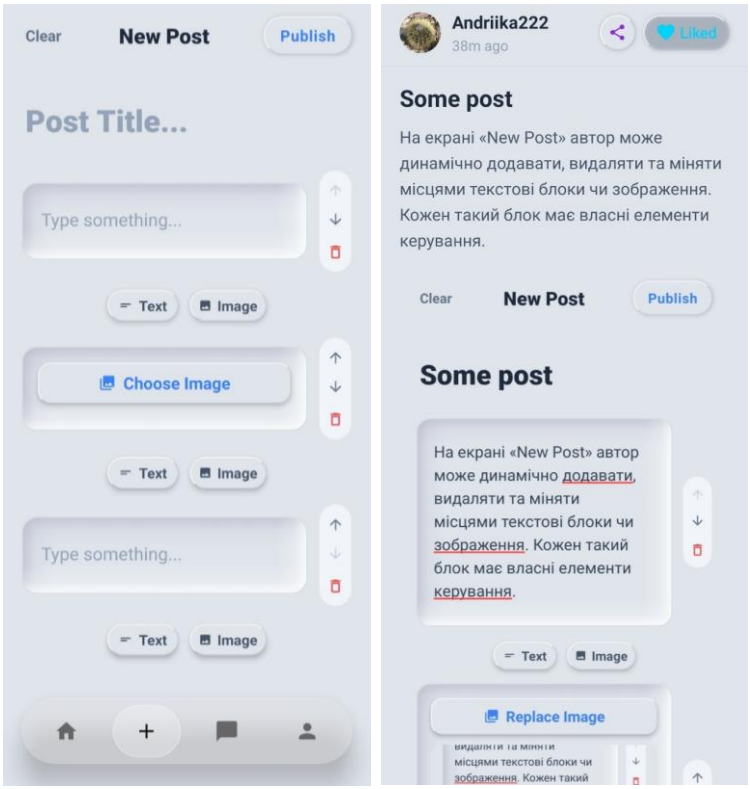
а) Вкладка Featured б) Вкладка Latest

Рисунок 3.10 – Інтерфейс локальної спільноти

Верхня частина екрана містить систему вкладок для зручного перемикання між різними типами контенту:

– Friends – публікації від користувачів, на яких оформлено підписку;
– Featured – головні, закріплені модераторами публікації (відображаються у вигляді великих карток-банерів та сітки);
– Latest – хронологічна стрічка всіх публікацій спільноти у вигляді класичної прокрутки (див. рис. 3.10);
– Chats – список доступних розмов.

Щоб зробити створення дописів максимально гнучким, замість одного звичного текстового поля було розроблено блоковий редактор контенту (див. рис. 3.11). На екрані «New Post» автор може динамічно додавати, видаляти та міняти місцями текстові блоки чи зображення. Кожен такий блок має власні елементи керування. Такий підхід стимулює користувачів писати якісні лонгріди та візуально насичені статті, що дуже важливо для вузькотематичних мікроспільнот.



а) Створення поста б) Деталі поста

Рисунок 3.11 – Інтерфейс поста

Інтерфейс локального чату розроблявся з акцентом на комунікацію та елементи гейміфікації (див. рис. 3.12). Крім стандартних текстових хмаринок та картинок, тут передбачено:

– блок закріпленого повідомлення у верхній частині екрана;
– відображення поточного локального рівня користувача безпосередньо на його аватарі, що візуалізує його статус та репутацію у даній спільноті;
– багатофункціональне поле вводу з можливістю додавання вкладень та запису голосових повідомлень.

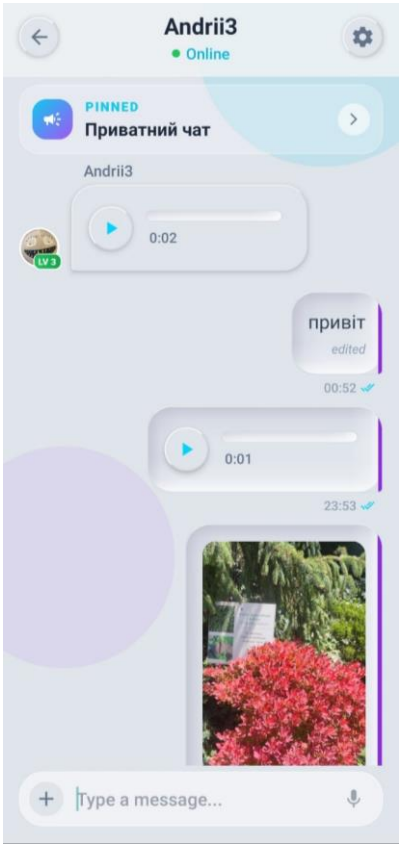


Рисунок 3.12 – Інтерфейс чату

Загалом спроєктований UI успішно вирішує головну проблему – візуально розділяє глобальну частину платформи та ізольовані внутрішні спільноти. Завдяки продуманій навігації та блоковому редактору, застосунок вийшов інтуїтивно зрозумілим і цілком відповідає очікуванням сучасної аудиторії.

3.3 Реалізація механізмів гейміфікації та модулів глобальної агрегації контенту

Гейміфікація допомагає утримувати увагу користувачів і стимулює їх спілкуватися частіше. Платформа використовує класичне накопичення балів досвіду (XP) разом із системою рівнів. Завдяки їм авторитет учасника стають візуально зрозумілими всередині будь-якої спільноти. Окремий лічильник репутації в профілі користувача залежить від кількості постів, які модератори додали в Featured. Швидкість розвитку профілю напряму залежить від реакцій аудиторії на опубліковані матеріали. Щоб збалансувати ігрову механіку, кожна корисна дія отримала власну вагу. Отриманий лайк під публікацією додає автору 0,6 XP. Якщо ж інші учасники оцінили коментар, нараховується 0,4 XP. Загальний показник досвіду розраховуємо за формулою:

$$currentXP = (postLikes \times 0{,}6) + (commentLikes \times 0{,}4).$$

Проєкт використовує нелінійну модель прогресії. Вона підтримує інтерес до платформи протягом тривалого часу. Спочатку рівні зростають швидко, але згодом вимагають значно більше активності. Максимальна межа становить 100 рівнів (або 100 000 одиниць досвіду). Поточний статус обчислюється за допомогою функції кубічного кореня, яка гарантує плавне ускладнення:

$$currentLevel = \min\left(100, \left\lfloor 100 \times \sqrt[3]{\frac{currentXP}{100000}} \right\rfloor\right).$$

Перехід на нову сходинку відкриває унікальний ранг і кольоровий бейдж. Система автоматично додає його до аватара в профілі (див. рис. 3.13) та повідомленнях. Інші користувачі одразу бачать рівень співрозмовника. Детальний розподіл рангів наведено в таблиці 3.1.

Таблиця 3.1 – Система рангів та рівнів користувачів

Рівні	Назва рангу	Колір бейджа
1–10	Novice	Зелений
11–20	Amateur	Синій
21–50	Experienced	Пурпурний
51–60	Expert	Фіолетовий
61–70	Guru	Білий
71–80	Bonvivant	Жовтий
81–90	Oracle	Помаранчевий
91–95	Master	Червоний
96–99	Adept	Чорний
100	Legend	Золотий

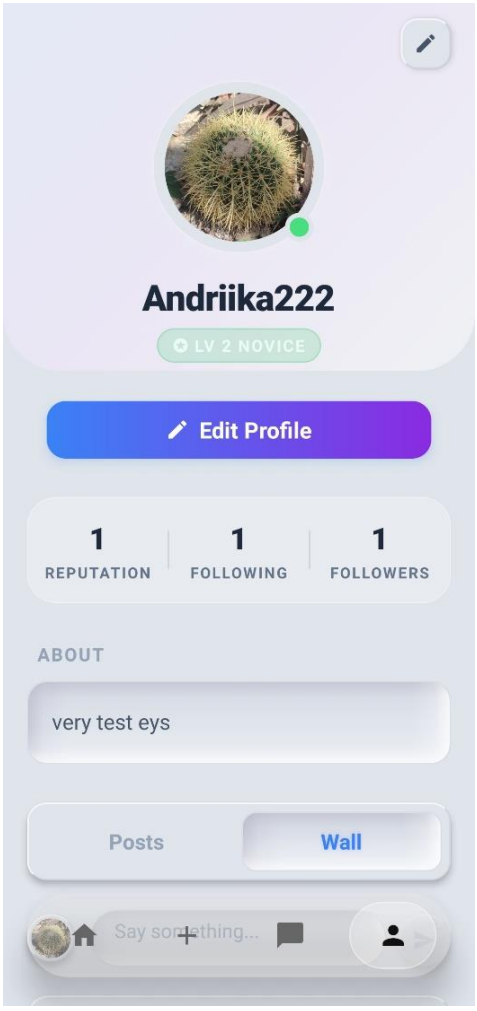


Рисунок 3.13 – Профіль користувача

Запропонована механіка мотивує створювати якісні дописи замість пасивного гортання стрічки. Отримати найвищий статус «Легенда» об’єктивно складно через нелінійну криву досвіду. Саме тому досягнення топу стає престижною ціллю, яка суттєво підвищує загальну залученість аудиторії.

Формування єдиного інформаційного простору поверх ізольованих мікроспільнот вимагає створення спеціалізованих екранів агрегації. Основне призначення згаданих модулів полягає у наданні миттєвого доступу до найважливіших публікацій та активних діалогів без потреби ручного перемикання між групами.

Програмна архітектура глобальної стрічки новин спирається на алгоритм вибірки закріплених матеріалів з усіх спільнот, куди встиг приєднатися користувач. Постачання масиву даних забезпечує менеджер стану Zustand за допомогою реактивного селектора selectGlobalFeaturedFeed. Візуальне відображення кожної публікації делеговано кастомному компоненту NeumorphBox, налаштованому на генерацію об’ємної картки з ефектом опуклості. Взаємодіяти з контентом дозволяє панель інтерактивних кнопок TactilePill. Завдяки розробленим функціям учасник здатен безперешкодно залишати вподобайки, переходити до гілки коментарів або дублювати запис у власний локальний профіль. Спираючись на можливості навігаційної бібліотеки Expo Router, натискання на будь-яку картку автоматично ініціює маршрутизацію всередину відповідної спільноти задля детального перегляду обраного допису.

Споріднений принцип консолідації інформації закладено в основу глобального месенджера. Простір екрана логічно розмежовано на горизонтальний блок активних друзів та вертикальний реєстр діалогів (див. рис. 3.14). Наповнення обох секцій відбувається паралельно через підписки на селектори selectGlobalOnlineFriends та selectGlobalChats. Задля прискорення роботи з великими списками контактів інтегровано механізм динамічної фільтрації. Введення пошукового запиту провокує миттєве відсіювання нерелевантних елементів шляхом аналізу імен користувачів, заголовків бесід.



Рисунок 3.14 – Список усіх чатів користувача

Графічна презентація списку чатів ретельно враховує структурні відмінності між їхніми типами. Групові обговорення супроводжуються квадратними обкладинками відповідних спільнот із чіткою вказівкою імені відправника останньої репліки. На противагу їм, персональні листування відмальовуються з використанням круглих аватарів співрозмовників.

3.4 Налагодження та функціональне тестування системи

Фінальна стадія розробки платформи Minisocial передбачала проведення комплексного налагодження разом із функціональним тестуванням готового програмного продукту. Ключовим завданням стало виявлення прихованих дефектів коду, перевірка алгоритмів обробки даних, а також підтвердження повної відповідності написаної системи попередньо сформованим вимогам.

3.4.1 Середовище тестування та інструменти налагодження

Оскільки клієнтський застосунок розроблено з використанням кросплатформного фреймворку React Native (версії 0.83) та інструментарію Expo, процес тестування проводився в умовах, максимально наближених до реальної експлуатації на різних мобільних операційних системах.

Для налагодження використовувалися такі інструменти:

– Expo Go: для швидкого запуску проміжних збірок на фізичних пристроях шляхом сканування QR-коду;
– реальна збірка: для аналізу поведінки UI-компонентів через пряме USB-підключення смартфона;
– React Native Debugger та Chrome DevTools: задля глибокого моніторингу дерева стану Zustand, відстеження мережевих запитів до Firebase та пошуку витоків пам’яті під час рендерингу масивних списків.

Завдяки впровадженому механізму Runtime Resolution вдалося організувати повністю ізольоване UI-тестування. Навіть за відсутності інтернет-з’єднання або під час збоїв на серверах Firebase інтерфейс продовжував стабільно працювати, автоматично підвантажуючи локальні Mock-дані з модуля mockRepositoryBundle.

3.4.2 Методологія функціонального тестування

Для перевірки працездатності бізнес-логіки застосунку було використано метод тестування «чорної скриньки». Суть підходу зводиться до аналізу зовнішньої поведінки застосунку без втручання у внутрішню структуру коду, спираючись виключно на вхідні параметри та очікуваний відгук користувацького інтерфейсу. Спеціально розроблений набір сценаріїв охопив найкритичніші модулі платформи: механізми авторизації, ізоляцію

профілів, взаємодію з мультимедіа та алгоритми нарахування репутації. Підсумкові результати перевірки ключових функцій зібрано в таблиці 3.2.

Таблиця 3.2 – Сценарії функціонального тестування застосунку

Назва сценарію	**Кроки для відтворення**	**Очікуваний результат**	**Фактичний результат**
Авторизація через Google	1. Натиснути «Sign in with Google» на екрані /auth. 2. Обрати тестовий обліковий запис.	Система створює GlobalAccount у Firestore, генерує токен та перенаправляє користувача на головний екран /(tabs).	Користувач успішно авторизований, дані профілю завантажено з Google.
Ізоляція локального профілю	1. Вступити до спільноти «Anime Central». 2. Змінити нікнейм у локальному профілі на «Otaku99». 3. Перевірити глобальний профіль та інші спільноти.	Нікнейм «Otaku99» відображається виключно в межах «Anime Central». Глобальний нікнейм залишається незмінним.	Система підтримує незалежні об’єкти CommunityProfile. Ізоляція працює коректно.
Завантаження медіа у публікацію	1. Створити нову публікацію через блоковий редактор. 2. Додати блок типу image та обрати фото розміром 5 МБ з галереї. 3. Натиснути «Publish».	Зображення перехоплюється модулем Cloudinary, завантажується у хмару, стискається. Пост зберігається з отриманим URL-посиланням.	Фото оптимізовано (перетворено у WebP), пост успішно опубліковано у стрічці.
Нарахування XP та прогресія рівнів	1. Створити публікацію у тестовій спільноті. 2. Отримати 100 лайків від інших тестових користувачів.	Згідно з алгоритмом (100 * 0,6), користувач отримує 60 XP. Поточний рівень та ранг перераховуються динамічно.	XP нараховано правильно. Локальний бейдж оновлено.
Запис голосового повідомлення	1. Відкрити чат спільноти. 2. Затиснути іконку «Мікрофон». 3. Говорити протягом 5 сек. 4. Відпустити кнопку.	Модуль expo-audio записує аудіофайл, формує ChatMessage типу audio з параметром audioDuration та надсилає його до БД.	Компонент AudioBubble з’являється у чаті. Відтворення працює з візуалізацією прогресу.

Продовження таблиці 3.2

Назва сценарію	Кроки для відтворення	Очікуваний результат	Фактичний результат
Скасування голосового повідомлення	1. Затиснути іконку «Мікрофон» для старту запису. 2. Виконати свайп (Slide to cancel) вліво до порогу - 130px.	Запис переривається, пристрій вібрує, повідомлення не відправляється до бази даних.	Анімація скасування відпрацьовує, запис видаляється.
Синхронізація «Online» статусу	1. Згорнути застосунок (перевести у background). 2. Перевірити статус користувача з іншого пристрою.	Слухач подій AppState фіксує зміну статусу та оновлює поле isOnline у Firestore на false.	Інші користувачі бачать статус «Offline» у списку online-friends.tsx.

3.4.3 Тестування безпеки та прав доступу (RBAC)

Надзвичайно важливим етапом стала перевірка підсистеми управління правами доступу (Role-Based Access Control), адже архітектура мережі дозволяє лідерам призначати власних модераторів. Задля імітації потенційної загрози було проведено тест на ескалацію привілеїв: звичайний учасник (маючи роль member) намагався надіслати запит на видалення чужого допису та закріплення повідомлення в чаті.

Написана функція getCommunityPermissions впоралася із завданням на клієнтському рівні, повністю приховавши кнопки адміністративних дій від неавторизованих очей. Паралельно серверні правила безпеки миттєво заблокували прямі спроби втручання в базу даних. Натомість акаунти зі статусами leader та global_admin безперешкодно виконали всі передбачені сценарії управління контентом.

Проведений комплекс налагоджувальних заходів остаточно довів повну працездатність створеного продукту Minisocial. Застосування ізольованих середовищ розробки значно пришвидшило виправлення візуальних багів, тоді як бібліотека Zustand гарантувала безперебійну реактивність інтерфейсу,

уникнувши небезпечних витоків пам’яті. Абсолютно всі впроваджені модулі, включаючи гейміфікацію, блоковий редактор контенту та складні мультимедійні чати, демонструють високу стабільність на цільових мобільних платформах.

3.5 Висновки до розділу 3

Третій розділ описує безпосередню програмну реалізацію та комплексне тестування соціальної мережі. Спираючись на попередньо сформовану архітектуру, написано повноцінний мобільний клієнт, паралельно налаштувавши безперебійну синхронізацію програми з хмарною інфраструктурою.

Фундаментом для побудови користувацького інтерфейсу виступив кросплатформний фреймворк React Native разом з інструментарієм Expo. Прагнучи досягти максимальної естетики, графічне оформлення збагатили ефектами Neumorphism та Glassmorphism, додавши візуальним елементам приємного об’єму. Водночас глибоке розділення кодової бази на незалежні шари гарантувало високу модульність продукту, суттєво оптимізувавши швидкодію та відкривши простір для легкого подальшого масштабування.

Стабільне оперування інформаційними потоками забезпечило впровадження архітектурного патерна «Репозиторій». Завдяки розробці адаптивного середовища виконання програма вміє гнучко перемикатися між реальними хмарними базами Firebase та локальними заглушками, дозволяючи продовжувати верстку інтерфейсу навіть без підключення до мережі. Паралельно загрозу перевантаження мобільного трафіку важким мультимедіа усунула інтеграція хмарного сервісу Cloudinary, здатного миттєво трансформувати та стискати графіку. Задля активного стимулювання аудиторії архітектуру застосунку доповнили унікальними механізмами гейміфікації. Шляхом впровадження нелінійної математичної моделі

нарахування балів досвіду та прогресії рівнів аж до сотої позначки вдалося створити систему рейтингів, здатну мотивувати учасників до створення якісних публікацій.

Серед усіх інженерних завдань найбільших зусиль вимагала розробка ключових компонентів застосунку. Зокрема, платформу доповнили кастомним блоковим редактором публікацій, екранами глобальної агрегації контенту та складними мультимедійними чатами із вбудованим розпізнаванням жестів і записом аудіо. Обходячи платформозалежні обмеження операційної системи Android, написано унікальні адаптивні модулі, яскравим прикладом яких є універсальний віджет NeumorphBox.

Підсумком роботи стало проведення комплексного функціонального тестування методом «чорної скриньки» поруч із прискіпливою перевіркою підсистеми прав доступу. Отримані результати беззаперечно доводять працездатність алгоритмів ізоляції профілів, точність нарахування репутації та миттєве серверне блокування будь-яких несанкціонованих дій. Зрештою, висока швидкодія та абсолютна стабільність готового програмного продукту підтверджують успішне виконання всіх завдань, поставлених у межах кваліфікаційної роботи.

ВИСНОВКИ

У кваліфікаційній роботі розв’язано актуальне науково-практичне завдання – створення сучасної, швидкої та масштабованої соціальної мережі Minisocial, яка орієнтована на взаємодію користувачів у межах ізольованих тематичних спільнот. Поставленої мети дослідження – зменшення часових витрат на розробку та розгортання інфраструктури мобільної платформи при одночасному підвищенні швидкості обміну даними в реальному часі – було успішно досягнуто завдяки застосуванню BaaS-технологій та кросплатформного підходу.

У ході аналізу предметної області та існуючих рішень-аналогів, таких як Reddit, Discord та Amino, було виявлено їхні технічні та концептуальні недоліки, що дозволило сформувати вимоги до створюваної системи. Для оптимізації процесу розробки клієнтської частини для операційних систем iOS та Android обґрунтовано вибір кросплатформного фреймворку React Native та інструментарію Expo. Роль бекенд-інфраструктури та зберігання даних у реальному часі делеговано хмарній платформі Firebase, а для ефективного управління станом і медіаконтентом інтегровано Zustand та Cloudinary. Системний аналіз та проєктування архітектури застосунку дозволили побудувати ієрархію акторів на основі ролей за допомогою UML-діаграм, а також змоделювати денормалізовану структуру документо-орієнтованої бази даних Cloud Firestore. Додатково було розроблено кастомний інтерфейс у стилістиці Neumorphism та нелінійну математичну модель гейміфікації з прогресією до сотого рівня для підвищення залученості користувачів.

Програмна реалізація клієнтського застосунку виконана мовою TypeScript із розділенням на три ізольовані шари, що забезпечує високу модульність кодової бази. Завдяки використанню архітектурного патерну «Репозиторій» у поєднанні з адаптивним середовищем виконання runtime, інтерфейс здатний автоматично перемикатися між хмарними даними Firebase

та локальними Mock-даними. У межах розроблених модулів практично втілено унікальні компоненти, зокрема гнучкий блоковий редактор публікацій, деревоподібну структуру багаторівневих коментарів та мультимедійні чати з підтримкою запису й відтворення голосових повідомлень за допомогою жестів.

Комплексне налагодження та функціональне тестування системи методом «чорної скриньки» підтвердило повну відповідність застосунку заявленим вимогам. Було успішно розв’язано платформозалежні проблеми ОС Android із відображенням клавіатури, а перевірка правил безпеки Firestore Security Rules довела надійність захисту від несанкціонованої зміни чужих даних чи ескалації привілеїв. Практичне значення отриманих результатів полягає у створенні робочої та масштабованої моделі соціальної мережі, що дозволяє суттєво скоротити час виходу продукту на ринок та мінімізувати ресурси на адміністрування інфраструктури.

ПЕРЕЛІК ПОСИЛАНЬ

1. Аллан С. Статистика соціальних мереж 2026: останні тенденції та дані. *DigiExe*. URL: https://digiexe.com/uk/blog/social-media-statistics/ (дата звернення: 12.02.2026).

2. Creating social media campaigns that deliver results. *Hijack Creative Agency*. URL: https://www.hijackcreative.co.uk/10-social-media-trends-for-2025/ (дата звернення: 12.02.2026).

3. Borkovich J. G., Breese-Vitelli J. Social media implosion: context collapse! *Issues in information systems*. 2016. Vol. 17, Issue 4. P. 167–177. URL: https://doi.org/10.48009/4_iis_2016_167-177 (дата звернення: 16.02.2026).

4. Social media trends 2025: 6 predictions and how to leverage them for your business. *Done*. URL: https://done.lu/social-media-trends-2025-6-predictions-and-how-to-leverage-them-for-your-business/ (дата звернення: 16.02.2026).

5. Contributors to Wikimedia projects. *Amino (app) – wikipedia. Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/Amino_(app) (дата звернення: 16.02.2026).

6. Неплях Л. Популярні соціальні мережі в Україні 2025: статистика, тренди та вплив. *Vector*. URL: https://vctr.media/ua/populyarni-soczialni-merezhi-v-ukrayini-2025-statistika-267232/ (дата звернення: 17.02.2026).

7. Massanari A. L. Participatory culture, community, and play: learning from reddit. Lang AG International Academic Publishers, 2014. 215 p.

8. Venkata B. Evolving cloud-native architectures: leveraging serverless computing for flexibility and scalability in applications. *Journal of scientific and engineering research*. 2022. Vol. 9, no. 9. P. 126–135. URL: https://doi.org/10.5281/zenodo.15044896 (дата звернення: 16.03.2026).

9. Firebase documentation. *Firebase*. URL: https://firebase.google.com/docs (дата звернення: 16.03.2026).

10. Fowler M., Sadalage P. NoSQL distilled: a brief guide to the emerging world of polyglot persistence. Upper Saddle River, NJ : Addison-Wesley, 2013. 188 p.

11. Core components and native components react. *React Native*. URL: https://reactnative.dev/docs/intro-react-native-components (дата звернення: 17.03.2026).

12. React native firebase. URL: https://rnfirebase.io/ (дата звернення: 17.03.2026).

13. Expo documentation. *Expo*. URL: https://docs.expo.dev/ (дата звернення: 17.03.2026).

14. Kato D. Micro state management with react hooks: explore custom hooks libraries like zustand, jotai, and valtio to manage global states. Packt Publishing, 2022. 254 p.

15. Cloudinary image & video API platform documentation. *Cloudinary*. URL: https://cloudinary.com/documentation (дата звернення: 18.03.2026).

ДОДАТОК А

Повна ER-діаграма бази даних

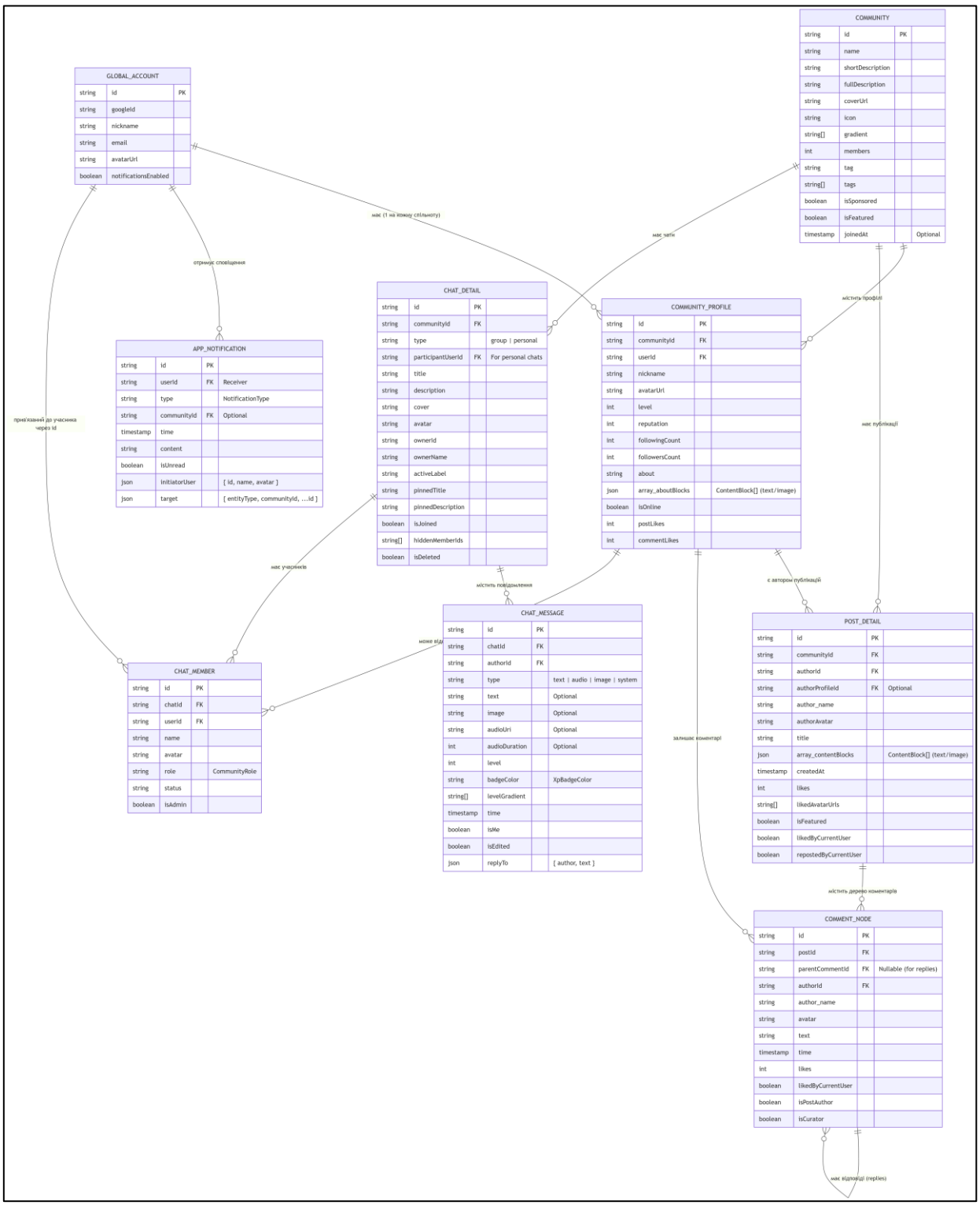


Рисунок А.1 – Повна ER-діаграма системи