НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри
Комп'ютерних наук
_____ Голуб Б.Л.

"____" _____ 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему

Програмне забезпечення соціальної медіа-платформи

Спеціальність 121 – «Інженерія програмного забезпечення»

Гарант освітньої програми
К.т.н., доцент _____ Вайганг Г.О.

Керівник бакалаврської кваліфікаційної роботи
к.т.н., доцент _____ Вайганг Г.О.
науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Шадура А.І.
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерних наук
_____ Голуб Б.Л.

"____" _____ 2025 р.

ЗАВДАННЯ
на виконання бакалаврської кваліфікаційної роботи студенту

Шадурі Андрію Ігоровичу

Спеціальність 121 «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи: Програмне забезпечення соціальної медіа-платформи

затверджена наказом ректора НУБіП України № 2248 "С" від 16.12.2024

Термін подання завершеної роботи на кафедру 2025.05.28
(рік, місяць, число)

Вихідні дані до роботи: опис програмного забезпечення

Перелік питань що розглядаються:
1. Системний аналіз предметної області інформаційної системи
2. Проєктування інформаційного та програмного забезпечення
3. Розробка інформаційного та програмного забезпечення
4. Рекомендації щодо впровадження та експлуатації системи

Дата видачі завдання «_____»_____2025 р.
Керівник бакалаврської кваліфікаційної роботи

к.т.н., доцент	_____	Вайганг Г.О.
науковий ступінь та вчене звання)	(підпис)	(ПІБ)

Завдання прийняв до виконання _____ Шадура А.І.
(підпис) (ПІБ студента)

ЗМІСТ

ВСТУП

Настала нова ера – ера соціальних мереж. У сучасному світі нова людина не може жити без телефону, й соціальні мережі сприяють цьому. Вони слугують основними засобами спілкування, обміну контентом, створення спільнот і поширення інформації. З розвитком мобільних технологій зростає попит на мобільні додатки, які є зручними, функціональними та інтуїтивно зрозумілими. Дуже важко уявити молоду людину, яка немає свого профілю в якійсь мережі, також це актуально для малого бізнесу та якихось фахівців, які бажають показати свій креатив для великого кола людей, або для маркетингу та спілкування.

Зараз уже недостатньо просто зробити стабільний додаток – користувачі хочуть, щоб він був зручним, швидким, безпечним і приємним у користуванні. Людям важливо мати можливість обмінюватися повідомленнями та медіа без затримок, бачити контент, який підходить саме їм, і мати відчуття, що все працює чітко та надійно. А для цього розробникам доводиться дбати про багато речей: надійне збереження даних, швидкий обмін інформацією, зручну модерацію та таке побудування системи, щоб вона витримувала велике навантаження.

Щоб не витрачати зайвий час на створення окремих додатків для різних платформ, розробники часто використовують інструменти на кшталт Flutter. Це дозволяє писати один код, який працює і на Android, і на iOS, що, в свою чергу, економить час і ресурси на оновлення. А якщо до Flutter додати ще й Firebase, то можна досить легко реалізувати авторизацію користувачів, завантаження фото, обмін контентом у реальному часі та навіть аналітику – і все це без складних серверних рішень.

У зв’язку з цим, актуальною є розробка сучасної соціальної медіа-платформи, здатної забезпечити якісний користувацький досвід у поєднанні з надійною інфраструктурою обробки та зберігання даних. Зокрема, мобільні додатки, що працюють у режимі реального часу, потребують ефективних рішень для реалізації основного функціоналу – публікацій, взаємодії між користувачами, модерації контенту та масштабованості.

Об’єктом дослідження є програмна система для мобільного соціального медіа-додатку. **Предметом** дослідження є процеси розробки та впровадження функціональних компонентів такої системи із використанням Flutter і Firebase.

Мета роботи полягає у створенні кросплатформного мобільного застосунку соціального спрямування, що забезпечує користувачам можливість публікувати текстовий та візуальний контент, взаємодіяти з публікаціями інших користувачів, управляти профілем, а також використовувати безпечні та масштабовані засоби автентифікації та зберігання даних.

Для досягнення цієї мети були поставлені наступні **завдання**:

- здійснити системний аналіз предметної області та визначити вимоги до інформаційної системи;
- змоделювати архітектуру додатку з використанням UML-діаграм і ER-моделі;
- розробити логічну модель даних, реалізовану в Firebase;
- реалізувати функціональні модулі мобільного застосунку на Flutter з використанням архітектурного патерну Bloc;
- забезпечити функціональність реєстрації, авторизації, створення та перегляду публікацій, коментування, лайків, підписок;
- здійснити тестування системи та сформулювати вимоги до її впровадження.

Проєктна розробка зосереджена на створенні інтерактивної, зручної та стабільної соціальної медіа-платформи, яка поєднує сучасні підходи до розробки мобільних додатків. Основна увага приділяється користувацькому інтерфейсу, реалізованому засобами Flutter, що забезпечує адаптивність і комфорт у користуванні на різних типах пристроїв. Для обробки даних і реалізації серверної логіки використовується хмарна платформа Firebase, яка дозволяє реалізувати реєстрацію та автентифікацію користувачів, а також надійне зберігання публікацій і мультимедійного контенту. Управління станами в системі побудоване на основі архітектурного патерну Bloc, що гарантує чітку структуру, високу масштабованість і зручність при тестуванні. Функціональність

мультимедійної взаємодії охоплює завантаження фотографій, перегляд профілів користувачів і формування стрічки новин.

У розробці застосунку використано мову програмування Dart у складі фреймворку Flutter. Бекенд реалізований за допомогою компонентів Firebase, таких як Authentication, Firestore, Storage і Cloud Functions. У структурі додатку застосовано архітектурні рішення Bloc і Provider, що сприяє підтримуваності, розширюваності та ефективній організації логіки роботи системи.

Запропонований програмний продукт має на меті створити інтуїтивно зрозуміле, стабільне та масштабоване середовище для соціальної взаємодії користувачів, що відповідає сучасним вимогам до мобільних додатків.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

У сучасному цифровому середовищі соціальні медіа-платформи стали невід'ємною частиною інформаційної інфраструктури, забезпечуючи засоби обміну візуальним та текстовим контентом, комунікації між користувачами, формування спільнот і персоналізованого інформаційного простору. Такі системи функціонують як програмно-технічні комплекси, що об’єднують базу даних, інтерфейс користувача, засоби обробки запитів та алгоритми взаємодії з контентом [4].

Предметна область, що розглядається у цій роботі, охоплює функціонування мобільного застосунку соціальної медіа-платформи, розробленого з використанням кросплатформного фреймворку Flutter та хмарної інфраструктури Firebase. Основним призначенням системи є надання можливості користувачам створювати публікації, взаємодіяти з контентом інших користувачів, забезпечувати комунікацію та реалізовувати механізми захисту й модерації.

Інформаційна система включає декілька взаємопов’язаних підсистем, кожна з яких виконує специфічні функції. Зокрема, користувацький інтерфейс реалізує доступ до функцій платформи через адаптивну мобільну оболонку; функціональні компоненти відповідають за обробку операцій зі створення, перегляду, оцінювання та коментування постів; серверна частина забезпечує зберігання даних, автентифікацію, авторизацію та захист персональних відомостей.

Загальну структурну модель предметної області подано у таблиці 1.1. У ній систематизовано ключові компоненти платформи, їхню роль та взаємозв’язок із користувачем і технічним середовищем.

Таблиця 1.1

Компоненти предметної області соціальної медіа-платформи

№	Компонент	Опис функціонального призначення
1	Користувацький інтерфейс	Забезпечує доступ до функцій платформи через адаптивний UI, реалізований у Flutter
2	Мобільний додаток	Клієнтська частина системи, що встановлюється на смартфон і забезпечує взаємодію з Firebase
3	Реєстрація та авторизація	Надання доступу до персоналізованого функціоналу за допомогою облікових записів
4	Контент-процеси	Створення, редагування, перегляд, вподобання та коментування постів
5	Пошук користувачів	Механізм знаходження профілів інших зареєстрованих користувачів
6	Модерація контенту	Видалення небажаних матеріалів з метою підтримки етичних норм у платформі
7	Захист персональних даних	Реалізований на базі Firebase Authentication та захищених протоколів передачі інформації
8	Система зберігання даних	Включає Firestore та Firebase Storage для структурованого зберігання публікацій і медіафайлів
9	Управління станами	Організовано на базі Bloc-патерну для чіткого поділу між логікою і візуальними компонентами

Представлена таблиця узагальнює структурну будову предметної області, ілюструючи взаємозв’язок між функціональними модулями, технологічними інструментами та роллю користувача. Кожен компонент відіграє важливу роль у забезпеченні цілісної, масштабованої та безпечної роботи системи.

На завершення варто зазначити, що ефективність предметної області соціальної медіа-платформи значною мірою залежить від узгодженої взаємодії між її компонентами. Інтеграція сучасних технологій, зокрема Flutter, Firebase та архітектурного патерну Bloc, дозволяє досягти високої продуктивності, стабільності та користувацької привабливості, що є ключовими чинниками успішної реалізації та впровадження подібних програмних продуктів.

1.2 Аналіз вимог до програмної системи

Функціональні вимоги є основою для розробки будь-якої інформаційної системи, оскільки вони визначають, що саме система повинна виконувати для досягнення поставленої мети. У контексті соціальної медіа-платформи такі вимоги формують логіку взаємодії користувача з додатком, регламентують функціональність обробки контенту, доступ до профілів, комунікацію та адміністрування.

Програмне забезпечення має забезпечувати реєстрацію нових користувачів, вхід до особистого кабінету, створення та редагування текстових і візуальних публікацій, реагування на чужі пости (через вподобання або коментарі), перегляд стрічки публікацій у хронологічному або персоналізованому порядку. Окрім того, важливими є функції пошуку інших учасників платформи, перегляд профілів, а також модерація, що дозволяє видаляти публікації, які порушують встановлені правила.

Для систематизації викладеного у таблиці 1.2 наведено узагальнену специфікацію основних функціональних вимог до соціальної медіа-платформи.

Таблиця 1.2

Специфікація функціональних вимог до системи

№	Основна вимога	Призначення та функціональне застосування
1	Реєстрація та авторизація	Надання користувачам доступу до функцій системи через створення облікового запису
2	Створення публікацій	Можливість публікувати текстові повідомлення з прикріпленими зображеннями
3	Перегляд стрічки	Отримання користувачем доступу до оновлень у стрічці новин
4	Взаємодія з контентом	Реалізація лайків, коментування, збереження постів
5	Видалення публікацій модератором	Здійснення контролю над контентом і підтримка правил платформи
6	Пошук користувачів	Забезпечення швидкого доступу до профілів інших учасників системи
7	Відображення профілю	Перегляд особистої сторінки та профілів інших користувачів з переліком опублікованого контенту

Систематизація функціональних вимог дає змогу виокремити ключові дії, що забезпечують базову і розширену взаємодію користувача з системою, а також підтримують керованість і якість контенту. Ці вимоги є мінімально необхідними для повноцінного функціонування соціальної медіа-платформи.

У подальшій розробці важливо дотримуватися принципу відповідності кожного компоненту зазначеним функціональним вимогам. Наприклад, модуль створення публікацій має підтримувати не лише введення тексту та завантаження фото, а й обробку помилок користувача, попередній перегляд та можливість редагування. Інтерфейс перегляду стрічки повинен реалізовувати адаптивну подачу контенту, а механізми авторизації – інтегрувати перевірку автентичності збереження даних.

Нефункціональні вимоги є критично важливим елементом проєктування програмного забезпечення, адже вони визначають не лише можливості системи, а й те, як ці можливості реалізуються на практиці. У контексті соціальної медіа-платформи, що працює з великими обсягами контенту й передбачає постійну взаємодію користувачів у реальному часі, саме якісні характеристики визначають рівень зручності, швидкодії, безпеки та масштабованості.

Платформа повинна не тільки забезпечувати функціональність (наприклад, створення постів чи коментування), але й гарантувати її виконання з мінімальною затримкою, збереженням цілісності даних і приємним користувацьким досвідом. Безпека, стабільність, здатність до розширення і сумісність з іншими сервісами – це ті характеристики, які впливають на загальну надійність і конкурентоспроможність цифрового продукту.

Щоб структуровано представити основні нефункціональні вимоги, у таблиці 1.3 наведено їх специфікацію з поясненням призначення та шляхів реалізації в межах обраної архітектури.

Наведені у таблиці вимоги формують основу якісної реалізації застосунку. Вони забезпечують очікувану поведінку системи в умовах реального навантаження, підтримують стабільну роботу, забезпечують захист даних і сприяють зручності користування.

Таблиця 1.3

Специфікація нефункціональних вимог соціальної медіа-платформи

№	Вимога	Призначення	Засоби реалізації
1	Безпека	Захист персональних даних користувачів від витоку або несанкціонованого доступу	Firebase Authentication, протокол HTTPS, контроль доступу, шифрування
2	Продуктивність	Забезпечення швидкого завантаження контенту та миттєвого відгуку інтерфейсу	Кешування, оптимізовані запити, Firebase Cloud Functions
3	Масштабованість	Можливість обслуговування зростаючої кількості користувачів та контенту	Хмарні сервіси Firebase, автоматичне масштабування сховища та баз даних
4	Надійність	Безперервна робота системи навіть у випадку збоїв або навантаження	Резервне копіювання, моніторинг, Firebase Realtime Database
5	Інтероперабельність	Забезпечення взаємодії з іншими системами та платформами	REST API, підтримка JSON, інтеграція через Firebase API
6	Зручність (юзабіліті)	Забезпечення інтуїтивного й комфортного використання	Адаптивний дизайн, UX-тестування, інтерфейс на базі Flutter

Розкриваючи окремі аспекти, варто акцентувати на безпеці – платформі, що обробляє особисті дані, необхідно гарантувати їх захист на всіх рівнях. Реалізація автентифікації через Firebase, використання захищених з'єднань та розмежування прав доступу дозволяють уникнути витоку або несанкціонованого втручання.

Продуктивність системи є критично важливою для мобільного середовища: повільна робота або затримки у відповіді призводять до втрати користувачів. Швидке завантаження контенту досягається завдяки кешуванню, а обробка логіки – через асинхронні функції в хмарі.

Масштабованість платформи – це запорука її подальшого розвитку. Система повинна залишатися стабільною навіть при збільшенні кількості користувачів у десятки чи сотні разів. Використання хмарних інфраструктур, таких як Firebase, забезпечує автоматичне масштабування бази даних, сховища й обчислювальних ресурсів.

Надійність означає відсутність критичних збоїв, втрати даних чи зупинки сервісу. Це особливо важливо у соціальних платформах, де безперервність взаємодії є ключовою. Моніторинг, резервне копіювання та failover-механізми є необхідними елементами.

Не менш важливою є здатність системи до інтеграції з іншими платформами – авторизація через сторонні акаунти, обмін через API, підтримка відкритих форматів обміну (наприклад, JSON). Це підвищує гнучкість системи.

Завершальним фактором є зручність інтерфейсу. Якщо користувач може інтуїтивно зрозуміти, як працює система, без зайвих пояснень або інструкцій, то рівень прийняття платформи суттєво зростає.

Таким чином, сукупність нефункціональних вимог визначає якість, надійність і конкурентоспроможність мобільного застосунку. Їхнє повноцінне врахування ще на етапі проєктування дозволяє закласти фундамент для стабільної та масштабованої соціальної медіа-платформи, орієнтованої на довгострокову експлуатацію та позитивний користувацький досвід.

1.3 Моделювання предметної області

Моделювання предметної області є важливою складовою розробки інформаційної системи, оскільки дає змогу сформувати формалізоване уявлення про об’єкти, процеси та їхні взаємозв’язки, що мають місце у реальному середовищі, яке система буде обслуговувати. У межах проєктування програмного забезпечення соціальної медіа-платформи цей етап дозволяє виокремити ключові логічні сутності, визначити характер їх взаємодії та сформувати основу для архітектурного проєктування.

Предметна область у даному контексті охоплює множину елементів соціальної взаємодії – користувачів, публікацій, вподобань, коментарів, підписок – кожен з яких має власну роль у забезпеченні цілісного функціонування платформи. Моделювання дає змогу не лише відобразити ці сутності у вигляді об'єктів, а й формалізувати сценарії їх використання та залежностей. Це створює підґрунтя для впровадження бізнес-логіки, управління станами і визначення можливих виключень або виняткових ситуацій.

Для візуалізації структури предметної області та поведінкових характеристик системи застосовується мова уніфікованого моделювання UML (Unified Modeling Language). Вона забезпечує стандартизовані засоби представлення об'єктів і процесів, що відбуваються у межах системи, та дозволяє зручно передавати проєктні рішення між членами команди. Зокрема, для соціальної платформи доцільно використовувати діаграми прецедентів, діаграми послідовності та діаграми активності – кожна з яких відображає окремий аспект взаємодії користувачів із системою [2].

Застосування UML дає змогу організувати проєктування на основі кількох принципів – абстрагування, багаторівневості та ієрархічної деталізації. Це дозволяє поступово переходити від загальної моделі функціональності до конкретних механізмів реалізації, адаптованих до інтерфейсу, бази даних і архітектури застосунку. Результатом такого підходу є точне і послідовне визначення логіки поведінки системи, що є запорукою її ефективної реалізації.

Для детального представлення функціональних сценаріїв взаємодії користувачів із системою доцільно використати діаграму прецедентів (use case diagram), яка відображає зовнішні суб’єкти (акторів) і типові варіанти використання системи з їхньої точки зору. Такий підхід дозволяє на ранньому етапі проєктування виявити основні очікування користувачів, сформувати логіку поведінки системи й описати її функціональність без прив’язки до конкретних технічних рішень.

Актори на діаграмі є зовнішніми по відношенню до системи сутностями, що ініціюють або підтримують взаємодію. У контексті соціальної медіа-

платформи такими акторами є звичайні користувачі, які створюють контент і взаємодіють із ним, та модератори, що відповідають за підтримання стандартів платформи. Прецеденти (варіанти використання) описують типові дії, які користувачі можуть виконувати: реєстрація, створення постів, коментування, вподобання тощо. Крім того, діаграма враховує відношення розширення (<<extend>>), які дозволяють деталізувати взаємодії або вказати додаткові дії, пов’язані з основним процесом.

Зображення системи у вигляді діаграми прецедентів дає змогу відокремити зовнішні ролі від внутрішніх процесів, що є ефективним інструментом при створенні технічного завдання. Діаграма на рис. 1.1 демонструє основні сценарії використання платформи та взаємозв’язки між користувачами й функціональністю.

Рис.1.1 Діаграма прецедентів для додатку соціальної медіа-платформи

Наведена діаграма прецедентів дозволяє узагальнити ключові дії, які виконуються в системі, та встановити логічний зв’язок між різними типами

користувачів і функціональністю додатку. Такий візуальний опис спрощує подальше моделювання бізнес-логіки, сприяє виявленню потенційних недоліків у взаємодії та є основою для формалізації функціональних вимог.

Для систематизації функціональних сценаріїв, відображених на діаграмі прецедентів, доцільно представити акторів та відповідні їм варіанти використання у вигляді таблиці. Такий підхід дозволяє структурувати інформацію про ролі користувачів у системі та визначити їхню участь у реалізації ключових функцій. У таблиці 1.4 наведено основні актори соціальної медіа-платформи та відповідні їм прецеденти, що відображають характер взаємодії з додатком.

Таблиця 1.4

Актори та прецеденти соціальної медіа-платформи

№	Актор	Прецедент	Опис дії
1	Користувач	Створення посту	Ініціація створення нової публікації з можливістю додавання тексту та зображення
2	Користувач	<<extend>> Додавання опису та фото	Розширення сценарію створення посту – включає опис публікації та прикріплення фото
3	Користувач	Уподобання посту	Реакція на публікацію у вигляді "лайку"
4	Користувач	Перегляд посту	Перехід до перегляду змісту окремої публікації
5	Користувач	Коментування для фото	Додавання текстового коментаря до зображення
6	Користувач	Перегляд стрічки	Ознайомлення з новинами користувачів, на яких підписаний
7	Модератор	Модерація контенту	Перевірка та видалення публікацій, що не відповідають правилам спільноти

Представлена таблиця відображає ролі двох ключових акторів – користувача та модератора – у межах роботи системи, а також відповідні їм прецеденти. Вона дозволяє чітко зрозуміти, які функції доступні кожному типу користувача та які сценарії взаємодії формують логіку роботи платформи. Ця

структура стане основою для подальшого детального моделювання поведінки системи за допомогою діаграм послідовності й активності.

Наступним етапом моделювання є побудова діаграми послідовності, яка дозволяє деталізувати сценарії взаємодії, описані на діаграмі прецедентів. На відміну від статичних моделей, діаграми послідовності фокусуються на часовій послідовності повідомлень, що передаються між об’єктами, які беруть участь у виконанні певного функціонального сценарію. Завдяки цьому стає можливим точно простежити, як розгортається взаємодія між компонентами системи протягом виконання операцій.

Кожен об’єкт, що бере участь у сценарії, відображається на діаграмі вертикальною лінією життя. Ці об’єкти можуть бути як користувачами, так і частинами програмного забезпечення – наприклад, інтерфейсами, контролерами або модулями бази даних. Обмін повідомленнями між ними представлено у вигляді горизонтальних стрілок, що показують виклики методів або відповідні реакції. Така візуалізація дозволяє зосередитись на динаміці обробки даних і забезпечує глибше розуміння внутрішньої логіки системи.

Діаграми послідовності також сприяють виявленню потенційних логічних помилок ще на етапі проєктування, коли виправлення є найменш затратним. Завдяки чіткому розмежуванню ролей і дій, вони дозволяють перевірити відповідність між очікуваною поведінкою користувачів і реакцією системи. Це особливо важливо для складних інформаційних систем, до яких належать і соціальні медіа-платформи.

Для наочності на рис. 1.2 подано діаграму послідовності, що демонструє виконання двох базових сценаріїв: створення нового поста користувачем та видалення існуючого поста адміністратором.

У зображеному сценарії користувач ініціює створення нового поста, після чого система приймає запит, обробляє дані, зберігає їх у базі й надсилає підтвердження. В іншій гілці взаємодії адміністратор виконує перегляд існуючих публікацій і надсилає запит на видалення, який система виконує, підтверджуючи дію.

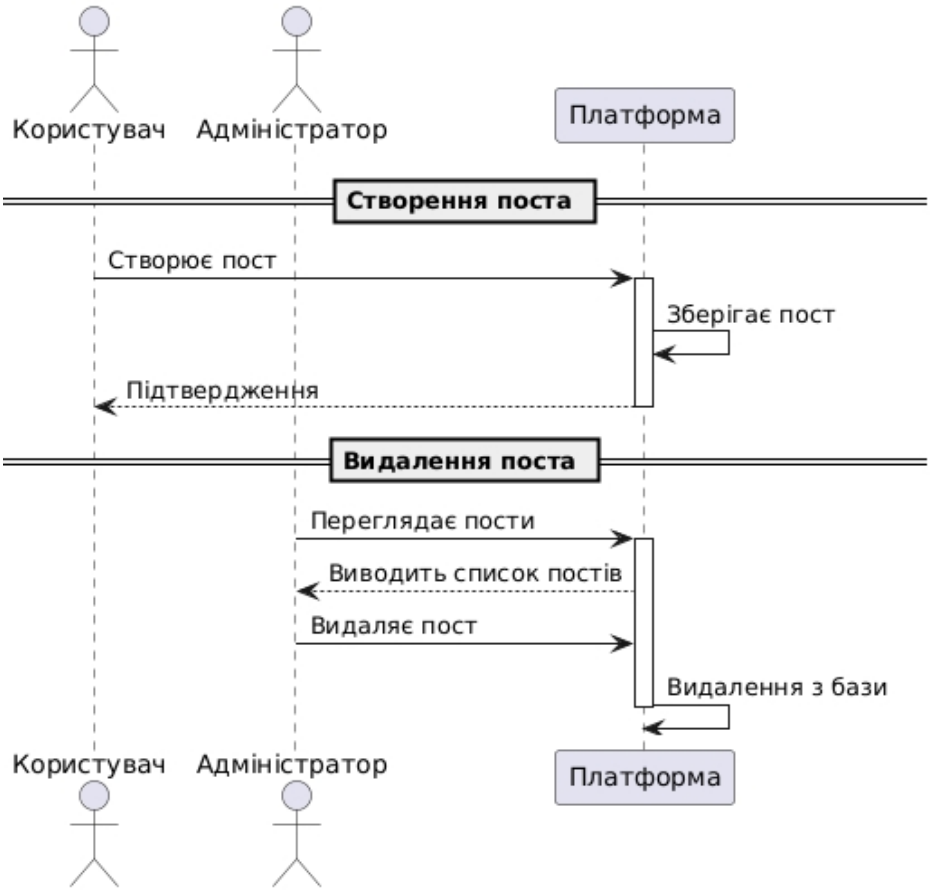


Рис. 1.2 Діаграма послідовності створення та видалення публікації в соціальній медіа-платформі

Діаграма дозволяє простежити весь цикл подій у часовій послідовності, надаючи чітке уявлення про логіку взаємодії між об’єктами системи.

Для систематизації функціональних сценаріїв, реалізованих у межах діаграми послідовності (рис. 1.2), доцільно представити взаємодію між об’єктами у вигляді табличної моделі (табл. 1.5). Це дозволяє наочно і логічно відобразити ролі учасників, характер повідомлень і їхню послідовність у межах виконання процесів створення та видалення постів.

Таблиця відображає хронологічну послідовність повідомлень, що реалізуються у процесах створення та модерації контенту. Такий підхід дозволяє структуровано проаналізувати логіку взаємодії між учасниками та компонентами системи.

Таблиця 1.6

Послідовність повідомлень у сценаріях взаємодії в соціальній медіа-платформі

№	Сценарій	Етап дії	Відправник	Одержувач	Зміст повідомлення / Дія
1	Створення поста	Ініціація створення	Користувач	Платформа	Надсилається запит на створення нового поста з контентом
2		Обробка і збереження	Платформа	Платформа	Валідація даних, обробка мультимедіа, збереження до бази даних
3		Підтвердження	Платформа	Користувач	Інформування про успішне створення та публікацію
4	Видалення поста	Запит на перегляд	Адміністратор	Платформа	Ініціація запиту на перегляд наявних постів
5		Отримання списку	Платформа	Адміністратор	Виведення списку постів із бази даних
6		Запит на видалення	Адміністратор	Платформа	Надсилання вказівки на видалення конкретного поста
7		Обробка видалення	Платформа	Платформа	Видалення обраного поста з бази даних та оновлення стану системи

Наявність чітко визначених повідомлень забезпечує правильну реакцію системи на дії користувача чи адміністратора, а також дозволяє формалізувати вимоги до реалізації ключових функціональних сценаріїв. Це важливо для наступного етапу – побудови логіки реалізації програмних компонентів.

У процесі розробки соціальної медіа-платформи важливо не лише змоделювати структуру системи й сценарії взаємодії, а й описати логіку виконання окремих функціональних процесів. Для цього застосовується діаграма активності – засіб графічного представлення алгоритму поведінки системи, який акцентує увагу на послідовності виконання дій та умовах переходу між ними. Такий тип діаграми дозволяє візуалізувати послідовний потік обробки даних, ураховуючи як прямі дії користувача, так і внутрішні логічні розгалуження або перевірки.

Діаграма активності за структурою подібна до блок-схеми, однак орієнтована на представлення поведінки об’єктів у межах програмної системи. Кожен елемент на ній відповідає окремій дії або перевірці, а стрілки вказують на напрям і логіку переходів між цими діями. У контексті соціальної платформи це дозволяє наочно відобразити, наприклад, процес подання скарги на порушення й подальшу модерацію контенту.

На рис. 1.3 зображено типовий процес взаємодії між користувачем і модератором у разі виявлення неприйнятного контенту. Діаграма демонструє, як відбувається подання скарги, її обробка системою, ухвалення рішення модератором та можливе видалення контенту або відхилення скарги.

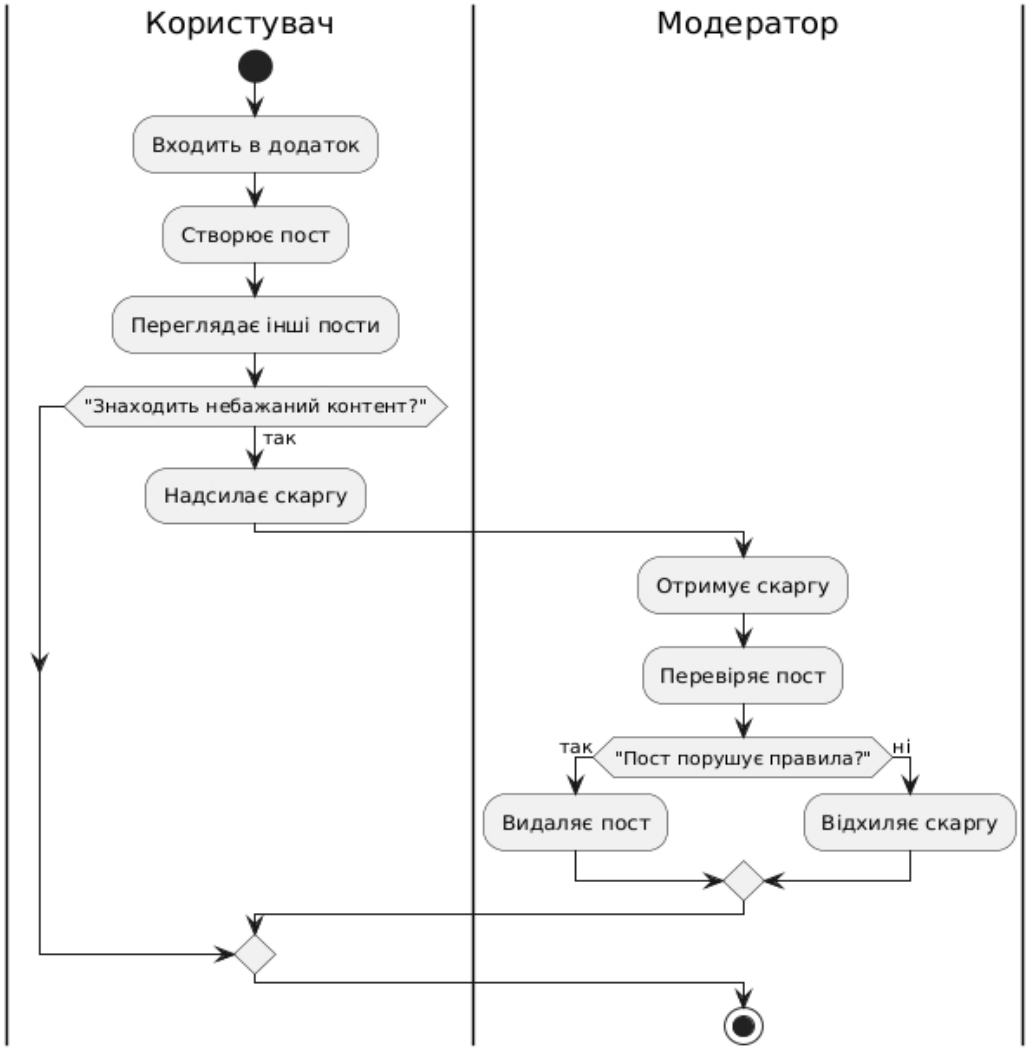


Рис. 1.3 Діаграма активності процесу модерації у соціальній медіа-платформі

Запропонована діаграма активності ілюструє послідовність операцій, що виконуються при обробці скарги на контент: від виявлення проблеми користувачем до остаточного рішення адміністратора. Така візуалізація дає змогу формалізувати логіку дій, виявити можливі точки гальмування процесу або ризик конфлікту рішень, а також слугує ефективним інструментом для технічної реалізації алгоритмів модерації в межах платформи.

Для деталізації логіки процесу модерації у межах діаграми активності (рис. 1.3) доцільно представити послідовність дій користувача та модератора у вигляді таблиці 1.7, яка дозволяє чітко структурувати ролі та кроки учасників процесу, що спрощує подальшу реалізацію бізнес-логіки системи.

Таблиця 1.7

Послідовність дій користувача та модератора у процесі модерації контенту

№	Учасник	Етап / дія	Опис дії
1	Користувач	Вхід у додаток (Start Node)	Ініціація сесії шляхом авторизації у мобільному застосунку
2	Користувач	Створення поста	Публікація власного контенту (текст, фото, відео)
3	Користувач	Перегляд публікацій	Ознайомлення з постами інших користувачів
4	Користувач	Рішення: "Знайдено неприйнятний контент?"	Прийняття рішення про доцільність подання скарги
5	Користувач	Надсилання скарги	Ініціювання модерації шляхом подання скарги через інтерфейс
6	Модератор	Отримання скарги	Повідомлення про нову скаргу надходить до модератора
7	Модератор	Перевірка публікації	Аналіз контенту, на який подано скаргу, щодо порушення правил
8	Модератор	Рішення: "Контент порушує правила?"	Прийняття рішення про відповідність публікації політиці платформи
9	Модератор	Видалення поста	У разі підтвердження порушення – видалення контенту
10	Модератор	Завершення модерації (Final Node)	Завершення процесу після обробки скарги

Дана таблиця дозволяє наочно представити етапи процесу модерації, підкреслюючи розмежування відповідальності між користувачем і модератором. Вона відображає типовий алгоритм взаємодії із системою у випадку виявлення порушення, забезпечуючи зрозуміле формалізоване представлення динаміки процесу.

1.4 Огляд інформаційних джерел та існуючих рішень

На етапі передпроєктного аналізу розробки соціальної медіа-платформи важливо провести порівняльне дослідження аналогічних інформаційних систем, проаналізувати сучасні технології реалізації та оцінити ключові користувацькі очікування. Такий підхід дозволяє зосередитись не лише на технічних можливостях, а й на факторі зручності, ефективності та відповідності поточним ринковим тенденціям. Особливу увагу приділено мобільним платформам, які забезпечують швидкий доступ до контенту та високий рівень персоналізації.

Серед найвідоміших реалізацій варто виокремити Instagram – платформу, орієнтовану на візуальний контент, з інтуїтивним користувацьким інтерфейсом та широким спектром взаємодій: публікація фото, відео, коментування, особисті повідомлення, лайки, історії, reels (рис. 1.4). Сильними сторонами Instagram є адаптивний дизайн, ефективна система рекомендацій і підтримка монетизації через рекламу та партнерства [5].

Рис. 1.4 Інтерфейс Instagram

Інша впливова система – **TikTok**, яка спеціалізується на динамічному короткому відеоконтенті та демонструє високу залученість користувачів завдяки вбудованому редактору, алгоритмічному формуванню стрічки й підтримці інтерактивного контенту (рис. 1.5). Платформа ідеально оптимізована під мобільні пристрої й дозволяє в режимі реального часу передавати та обробляти відео [6].

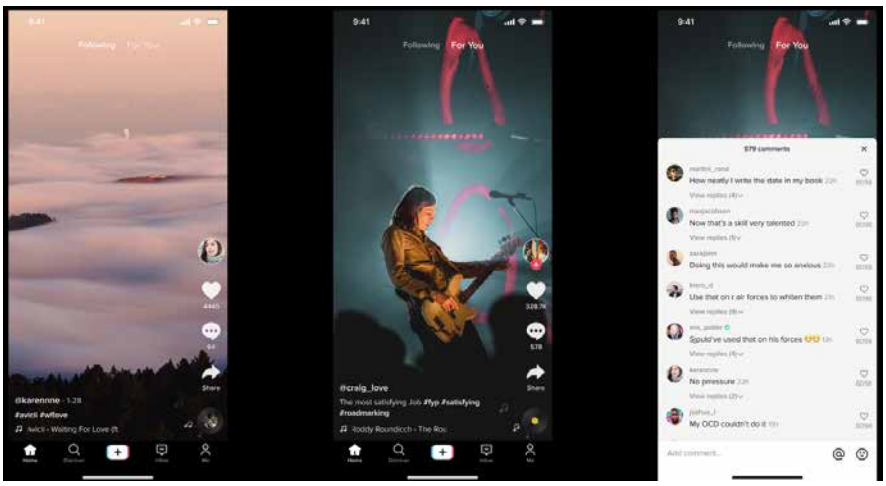


Рис.1.5 Інтерфейс TikTok

Ще один приклад – Threads від компанії Meta. Цей застосунок є текстовим доповненням до Instagram і реалізує ідеї мікроблогінгу в мінімалістичному оформленні (рис. 1.6). Основний акцент зроблено на створенні коротких текстових повідомлень і формуванні стрічки за інтересами, аналогічно до Twitter. Threads демонструє концепцію розвитку функціоналу поступово – від базового до розширеного [7].

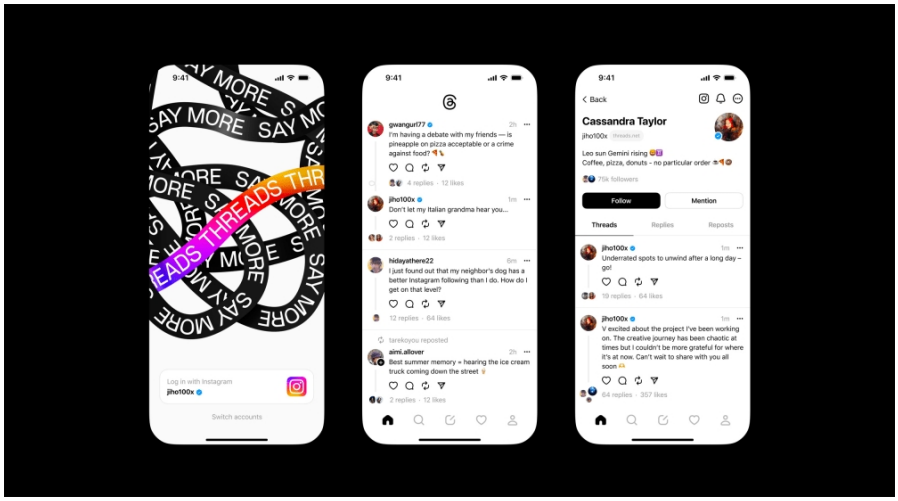


Рис. 1.6 Інтерфейс Threads

Facebook є багатофункціональним комунікаційним інструментом, що поєднує можливості особистого профілю, груп, сторінок брендів, бізнесу, подій, медіа та обміну повідомленнями (рис. 17). Його досвід важливий для розуміння

масштабування функціоналу. Водночас надмірна складність навігації стала одним із факторів, що вплинули на зниження популярності серед молодої аудиторії [8].

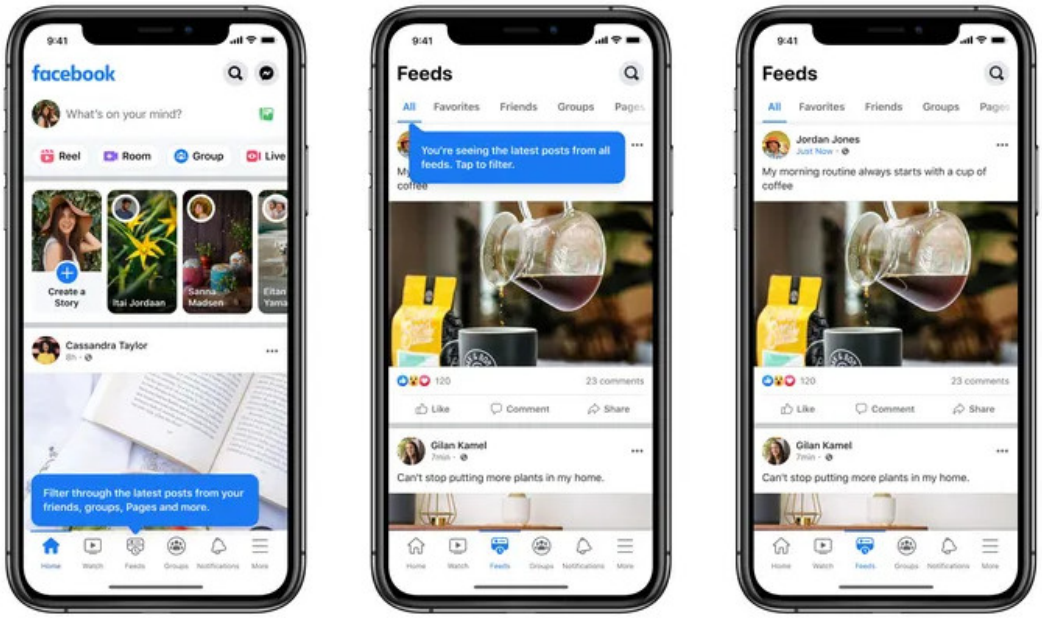


Рис. 1.7 Інтерфейс Facebook

Платформа BeReal демонструє інноваційний підхід до автентичності контенту, обмежуючи користувача в часі на публікацію (рис. 1.8). Протягом випадкового моменту дня надсилається сповіщення, після якого є лише 2 хвилини на публікацію реального фото. Такий механізм створює довіру до контенту та знижує рівень ідеалізованого зображення життя, що часто присутнє в інших мережах [9].

Рис. 1.8 Інтерфейс BeReal

Для узагальнення було проведено порівняльний аналіз платформ за основними критеріями, результати якого наведено в таблиці 1.8.

Таблиця 1.8

Порівняльний аналіз соціальних медіа-платформ

№	Платформа	Основні переваги	Основні недоліки
1	Instagram	UX-дизайн, інструменти монетизації, мультимедійна підтримка	Недостатня кастомізація профілю, питання конфіденційності
2	TikTok	Відеоредактор, AI-рекомендації, швидкий ріст аудиторії	Контент низької якості, ризики для підлітків
3	Threads	Простота, текстова модель, інтеграція з Instagram	Обмежений функціонал, відсутність особистих повідомлень
4	Facebook	Широкий набір функцій, гнучке адміністрування, масштабованість	Складна навігація, перевантаження інтерфейсу
5	BeReal	Унікальна концепція «реального моменту», автентичність контенту	Обмежена аудиторія, мінімальний функціонал

Аналіз існуючих інформаційних рішень показав, що найуспішніші соціальні платформи поєднують простоту у використанні, глибоку персоналізацію, мультимедійність і підтримку активної взаємодії користувачів. Отримані результати дозволяють сформувати бачення майбутньої системи, адаптуючи перевірені підходи до нових потреб цільової аудиторії. Досвід провідних платформ слугує джерелом як для технічного, так і для функціонального проєктування інтерфейсу та логіки взаємодії.

1.5 Постановка завдання

Після аналізу предметної області, вивчення функціональних і нефункціональних вимог, а також порівняння з існуючими аналогами було сформульовано конкретну постановку завдання, яке визначає цілі, очікувану функціональність і технічні аспекти реалізації програмного забезпечення соціальної медіа-платформи.

У контексті швидкого розвитку мобільних комунікацій і зростаючої потреби у платформах для обміну контентом, постає завдання створення кросплатформного застосунку, що забезпечує зручну, швидку та безпечну взаємодію користувачів через публікації, лайки, коментарі й персоналізовану стрічку новин. В основі системи повинна лежати модульна архітектура з чітким поділом на компоненти інтерфейсу, логіки взаємодії та управління даними, що реалізується засобами Flutter, Firebase та архітектурного патерну Bloc.

З урахуванням проведеного аналізу, завдання передбачає розробку кросплатформного мобільного додатка, який забезпечує реєстрацію, авторизацію, створення, перегляд і взаємодію з публікаціями (через лайки, коментарі, підписки) та управління особистим профілем, із підтримкою хмарної обробки та зберігання даних за допомогою сервісів Firebase Authentication, Firestore і Storage; передбачає реалізацію зручного, адаптивного та інтуїтивного інтерфейсу користувача на основі Flutter, впровадження механізму модерації контенту для адміністраторів, моделювання системи з використанням UML-діаграм і побудову логічної моделі бази даних у третій нормальній формі, проведення тестування функціональних сценаріїв з метою перевірки працездатності й надійності системи, а також формування рекомендацій щодо впровадження й визначення апаратно-програмних вимог для всіх категорій користувачів.

З огляду на викладене, завданням є створення цілісного, багатофункціонального, масштабованого програмного рішення з відкритою архітектурою, яке забезпечить високий рівень залучення користувачів та стабільну роботу за умов зростаючого навантаження. Передбачено також документування структури системи, її компонентів і логіки взаємодії, що забезпечить можливість її подальшого розвитку та адаптації під потреби спільноти користувачів.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Проєктування бази даних є одним із ключових етапів розробки програмного забезпечення, що забезпечує надійне зберігання, обробку та цілісність інформації. Для формалізації структури даних та їхніх взаємозв'язків на концептуальному рівні використовується ER-модель (Entity–Relationship), яка відображає сутності предметної області, атрибути та зв'язки між ними. Такий підхід дає змогу заздалегідь виявити логічні помилки в побудові системи, уникнути надмірного дублювання даних та забезпечити адаптивність архітектури до можливого розширення функціоналу.

ER-модель є універсальним інструментом для побудови логічної структури як реляційних, так і об'єктно-орієнтованих баз даних. Вона дозволяє описати систему у вигляді графічної діаграми, де вузлами виступають сутності (наприклад, користувач, пост, коментар), а зв'язки між ними – це логічні взаємодії, які визначають структуру інформаційного середовища. Перевагою такого підходу є його абстрактність: одна й та сама ER-модель може бути реалізована різними способами залежно від середовища розробки та технічних обмежень.

Для створення діаграми "сутність – зв'язок" можуть використовуватись різні інструменти. Одним із найбільш функціональних середовищ для побудови таких моделей є CA ERwin Data Modeler – спеціалізований CASE-засіб, який дозволяє створювати логічні та фізичні моделі даних, візуалізувати зв'язки між сутностями, документувати структуру бази та формувати SQL-скрипти на основі моделі. Інтерфейс ERwin дозволяє адаптувати моделі до конкретної предметної області, а також підтримує командну роботу, що є особливо корисним при розробці складних інформаційних систем. Як альтернативу, можна застосовувати універсальні вебінструменти, зокрема DrawIO, який дозволяє

створювати базові ER-діаграми онлайн, забезпечуючи швидку візуалізацію структури даних і зручну спільну роботу над проєктом.

Застосування ER-моделі у проєктуванні бази даних соціальної медіа-платформи дозволяє формалізувати основні сутності системи (користувачі, пости, лайки, коментарі), визначити типи їхніх атрибутів та зв’язків, а також закласти підґрунтя для реалізації масштабованої, стабільної та ефективної системи збереження й обробки даних. Побудована логічна модель охоплює ключові аспекти взаємодії користувачів із контентом, що дозволяє забезпечити узгодженість між бізнес-логікою додатку та структурою бази даних.

У ході розробки було побудовано ER-діаграму, яка представлена на рис. 2.1. Вона охоплює базові сутності та типові зв’язки між ними, зокрема зв’язок між користувачем і публікаціями, між публікацією та коментарями, а також між користувачем і лайками. Всі зв’язки реалізовані з урахуванням кардинальностей і забезпечують повноту логічної структури.

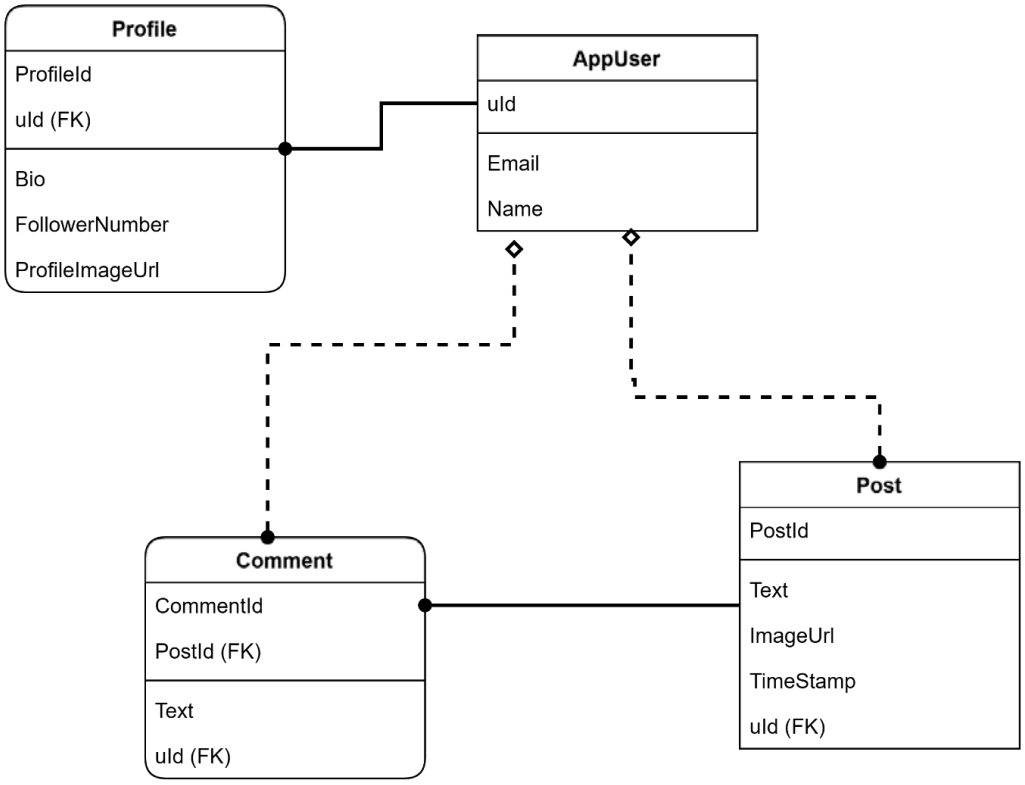


Рис.2.1 ER-діаграма для соціальної медіа-платформи

Для зручності аналізу основних сутностей і відповідних атрибутів у таблиці 2.1 представлено їх короткий опис. Це дозволяє швидко ознайомитися зі складом інформаційної моделі без необхідності перегляду графічної структури.

Таблиця 2.1

Основні сутності логічної моделі бази даних

Сутність	Первинний ключ	Зовнішні ключі	Інші атрибути	Зв'язки з іншими сутностями
Profile	ProfileId	uid (до AppUser)	Bio, FollowerNumber, ProfileImageUrl	Один-до-одного (1:1) з AppUser
AppUser	uid	None	Email, Name	Один-до-одного (1:1) з Profile Один-до-багатьох (1:N) з Post Один-до-багатьох (1:N) з Comment
Post	PostId	uid (до AppUser)	Text, ImageUrl, TimeStamp	Багато-до-одного (N:1) з AppUser Один-до-багатьох (1:N) з Comment
Comment	CommentId	PostId (до Post), uid (до AppUser)	Text	Багато-до-одного (N:1) з Post Багато-до-одного (N:1) з AppUser

ER-модель є критично важливим інструментом під час проєктування структури даних, оскільки дозволяє абстраговано, але точно описати логіку зберігання та зв’язків інформації в межах системи. Вона забезпечує основу для розробки реляційної моделі, сприяє уникненню логічних помилок, дозволяє масштабувати структуру бази та формалізує взаємодію між об’єктами предметної області. Отримана логічна модель відповідає вимогам до зберігання даних соціальної медіа-платформи й орієнтована на забезпечення її функціональної цілісності, продуктивності та гнучкості при подальшому розширенні функціоналу.

Побудована модель бази даних повністю відповідає вимогам нормалізованого підходу до проєктування інформаційних систем. Зокрема, при аналізі структури було підтверджено, що всі сутності реалізовані у відповідності до третьої нормальної форми (3НФ), яка є стандартом для забезпечення логічної узгодженості, мінімізації надмірності даних та ефективної обробки запитів.

На першому рівні нормалізації (1НФ) усі атрибути в таблицях мають атомарні значення, тобто відсутні повторювані групи або багатозначні поля. Це забезпечує цілісність кожного запису й спрощує механізми доступу до окремих одиниць даних. Усі записи містять фіксовану кількість атрибутів, і кожен стовпець має єдину інтерпретацію.

Другий рівень (2НФ) передбачає, що всі неключові атрибути залежать від повного первинного ключа, а не лише від його частини. Наприклад, у таблиці Followers первинний ключ є складеним і включає два зовнішні ключі – follower_id та following_id, від яких залежить логіка запису. Усі атрибути таблиць Post, Comment та ProfileUser також функціонально залежать виключно від їхніх ключів, що підтверджує відповідність вимогам 2НФ.

Щодо третьої нормальної форми (3НФ), то структура кожної таблиці була проаналізована на предмет транзитивних залежностей між неключовими атрибутами. У таблицях, де присутні додаткові поля (наприклад, bio, profileImageUrl, text, timestamp), всі вони безпосередньо залежать від первинного ключа або зовнішнього ключа, що відповідає сутності. Відсутність залежностей між неключовими атрибутами підтверджує відповідність моделі вимогам 3НФ.

Таким чином, логічна модель даних для розроблюваної соціальної медіа-платформи має оптимальну структуру, що відповідає всім критеріям третьої нормальної форми. Це забезпечує не лише ефективне зберігання та обробку інформації, а й високий рівень логічної цілісності даних. Вибраний підхід дозволяє уникнути помилок, пов’язаних із дублюванням або суперечливістю даних, забезпечує масштабованість системи та створює надійну основу для розширення функціональності в майбутньому.

2.2 Діаграма класів та кооперацій

У процесі проєктування програмної архітектури важливим етапом є побудова діаграми класів, яка дозволяє формалізувати структуру системи в термінах об'єктно-орієнтованого підходу. Діаграма класів в UML (Unified Modeling Language) відображає ключові класи, їхні атрибути, методи та взаємозв’язки, що є основою для розробки коду й комунікації між розробниками. Такий тип моделі є надзвичайно корисним для уточнення логіки функціонування об’єктів і забезпечення цілісності при реалізації програмного інтерфейсу [1].

Класова модель визначає, які об’єкти існують у системі, яку інформацію вони містять та як взаємодіють між собою. У межах соціальної медіа-платформи це дозволяє сформувати чітку структуру – від користувача до модератора, від профілю до публікації, – забезпечуючи логічне розділення відповідальностей між елементами. Такий підхід дозволяє організувати архітектуру системи в межах принципів повторного використання коду, інкапсуляції та підтримки масштабованості.

У таблиці 2.2 наведено основні цілі використання діаграм класів у програмному проєктуванні, що дозволяє системно зрозуміти їхню роль у життєвому циклі розробки.

Таблиця 2.2

Основні цілі щодо використання діаграми класів

№	Ціль	Опис
1	2	3
1	Визначення структури системи	Діаграма класів показує основні об'єкти (класи), які будуть використовуватись у системі.
2	Опис атрибутів та методів класів	Дає змогу формалізувати, які дані зберігаються в кожному класі та які функції вони виконують.
3	Встановлення відносин між класами	Візуалізуються асоціації, агрегації, композиції та наслідування між класами.
4	Полегшення командної роботи	Сприяє кращому розумінню архітектури системи усіма учасниками проєкту.

Таблиця 2.2 (продовження)

1	2	3
5	Основи для реалізації програмного коду	Створення шаблонів об’єктів для наступного програмного впровадження.
6	Підтримка документування та тестування	Забезпечення формалізації взаємодій для наступного модульного тестування.

У процесі розробки платформи було побудовано діаграму класів, що включає п’ять основних класів: «Користувач», «Профіль», «Публікація», «Коментар» та «Модератор». Їхня структура узагальнено у таблиці 2.3.

Таблиця 2.3

Структура діаграми класів

№	Клас	Властивості	Методи
1	Профіль	Пошта та ім'я	Створити користувача, Зареєструватися, Авторизуватися, Оновити профіль, Видалити акаунт
2	Користувач	Ідентифікатор, профіль	Створити профіль, Отримати профіль, Підписатися, Відписатися, Переглянути підписників
3	Публікація	Текст, зображення, дата, автор	Створити пост, Редагувати пост, Видалити пост
4	Коментар	Текст коментаря, автор	Додати коментар, Видалити коментар
5	Модератор	Пошта, ім’я	Видалити пост

Діаграма класів, наведена на рис. 2.2, відображає загальну архітектуру програмної моделі, де класи взаємодіють між собою через асоціативні та спадкові зв’язки. Клас «Користувач» асоційований із класом «Профіль», клас «Публікація» пов’язаний із класами «Користувач» і «Коментар», тоді як «Модератор» виконує контроль над постами. Така модель дає змогу логічно поділити відповідальність між об’єктами системи, що значно спрощує її розгортання, масштабування та підтримку.

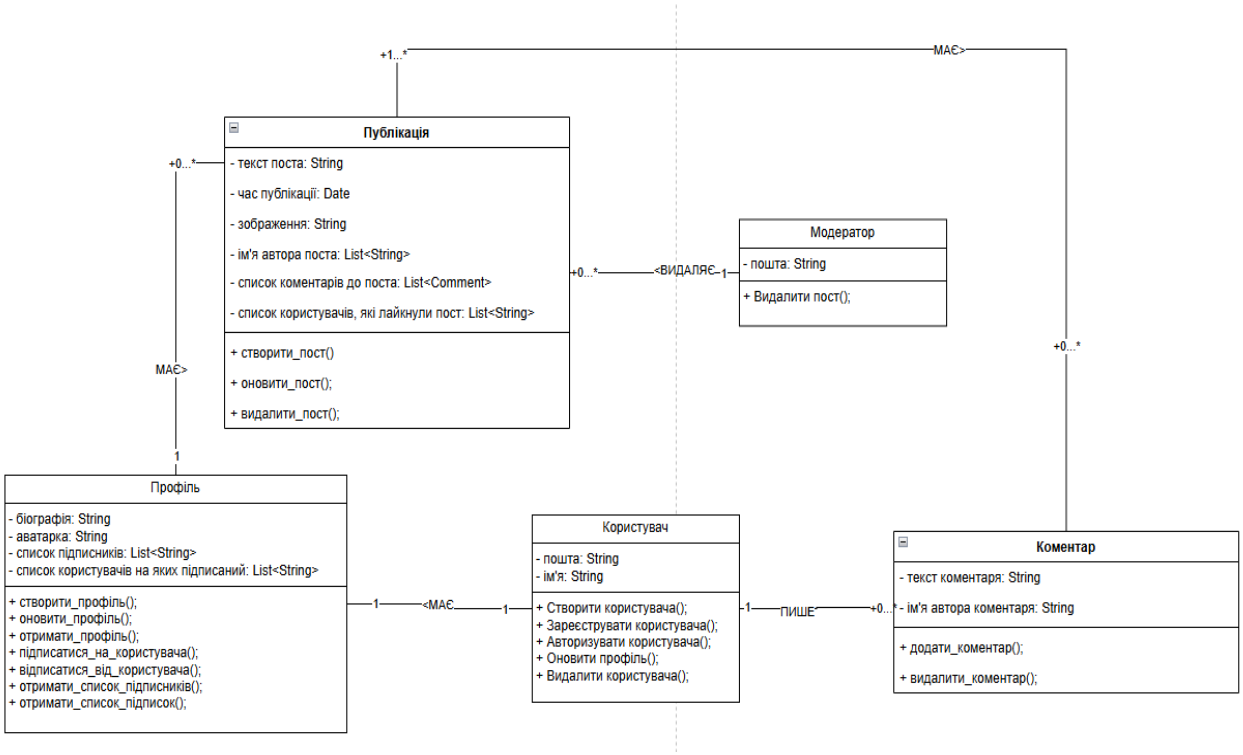


Рис. 2.2 Діаграма класів для соціальної медіа-платформи

Діаграма класів забезпечує глибоке розуміння логічної структури системи, формалізує ключові об’єкти та взаємозв’язки між ними, а також слугує основою для подальшої реалізації програмного коду. Вона виконує функцію комунікаційного інструмента в команді розробників і дозволяє ефективно трансформувати концептуальну модель у технічну реалізацію. У наступному розділі буде представлено діаграму кооперацій, що деталізує взаємодії між об'єктами під час виконання певних сценаріїв.

Окрім побудови діаграми класів, яка визначає загальну структуру об'єктів системи, важливо також моделювати конкретні сценарії взаємодії між її елементами. У цьому контексті застосовується діаграма кооперації (cooperation diagram, також відома як collaboration diagram), яка дозволяє візуалізувати динамічну поведінку об’єктів під час виконання окремих функціональних завдань.

На відміну від діаграми класів, яка представляє абстрактні класи та їхні атрибути, діаграма кооперації демонструє взаємодію конкретних об'єктів, що створені на основі відповідних класів. Її мета полягає у відображенні

послідовності та напряму обміну повідомленнями між об’єктами, які безпосередньо беруть участь у реалізації певного сценарію. Для цього використовується нотація з іменованими стрілками, що позначають передавання запитів, і зв’язками, які відображають асоціації, встановлені між об’єктами на основі класових взаємозв’язків.

Кожна така діаграма відображає один конкретний сценарій використання, відповідно до якого обираються лише релевантні об’єкти. Це дозволяє деталізувати логіку взаємодії між частинами системи на прикладі окремої функції, наприклад створення профілю користувача, публікації контенту або дій модератора щодо контролю змісту. При цьому навіть однакові об’єкти можуть виступати в різних ролях залежно від поставленої задачі.

На рис. 2.3 представлено кооперацію об’єктів у процесі створення профілю, де користувач ініціює запит, платформа створює профіль у базі даних, після чого результат надсилається назад користувачеві.

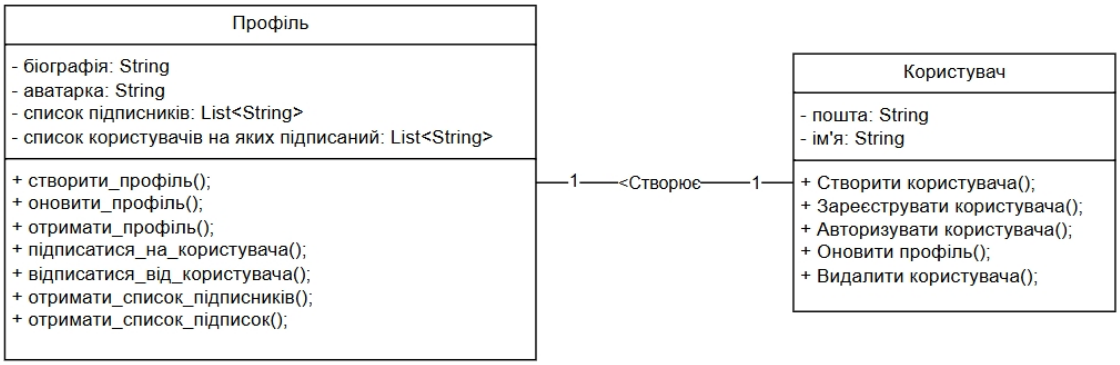


Рис. 2.3 Діаграма кооперації для сценарію створення профілю

Інший приклад, наведений на рис. 2.4, демонструє сценарій створення нового поста. У цьому випадку об’єкти «Користувач», «Пост» і «Сховище» взаємодіють через послідовну передачу повідомлень, що охоплює завантаження медіаконтенту, збереження метаданих та підтвердження успішного створення публікації.

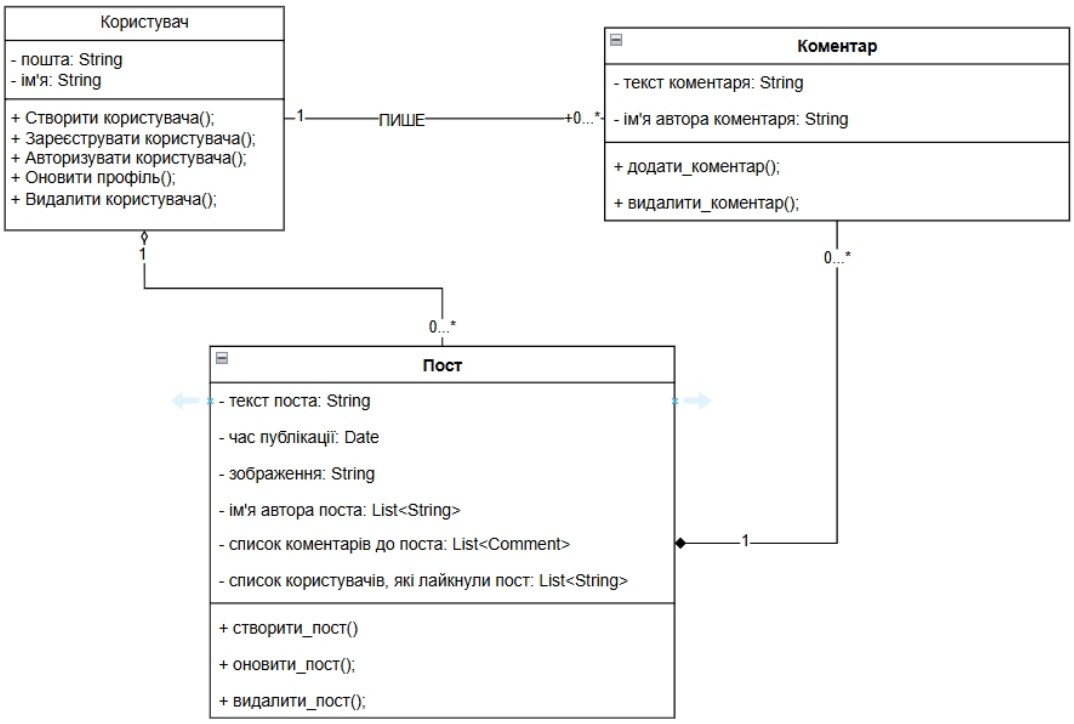


Рис. 2.4 Діаграма кооперації для сценарію створення поста

На рис. 2.5 представлено діаграму кооперації для сценарію модерації контенту. У цьому випадку ініціатива надходить від модератора, який взаємодіє з системою для перегляду, аналізу та можливого видалення вмісту, що порушує правила платформи. Схема демонструє, як здійснюється контроль за змістом та реалізується механізм видалення постів.

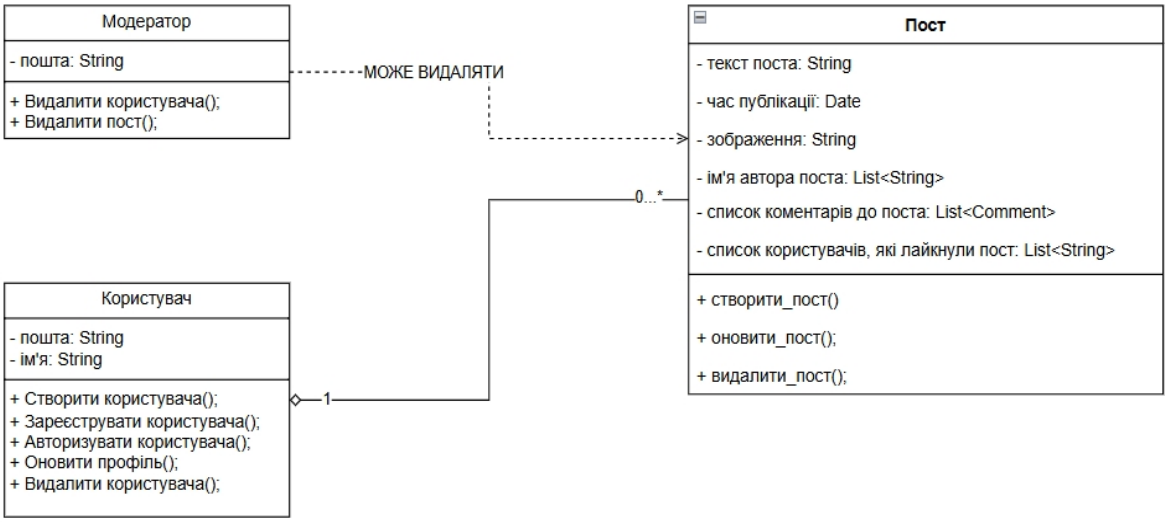


Рис. 2.5 Діаграма кооперації для сценарію модерації контенту

Діаграми кооперацій є важливим доповненням до статичних моделей, оскільки дозволяють перейти від абстрактного опису класів до конкретного представлення механізму взаємодії між їхніми екземплярами. Завдяки таким діаграмам можна глибше зрозуміти логіку функціонування системи, перевірити послідовність викликів методів і виявити можливі невідповідності у поведінці об’єктів. Вони є ефективним інструментом при розробці, тестуванні та комунікації функціональних сценаріїв між учасниками проєкту, сприяючи досягненню узгодженості між бізнес-логікою та її технічною реалізацією.

Розглянуте структурне і поведінкове моделювання системи за допомогою діаграм класів і кооперацій дозволяє сформувати цілісне уявлення про внутрішню архітектуру платформи та механізми взаємодії її об'єктів. Таке моделювання є необхідною передумовою для подальшої ефективної реалізації, тестування та супроводу програмного забезпечення.

2.3 Діаграма пакетів

Для ефективного управління складною структурою програмного забезпечення та забезпечення зрозумілого поділу його компонентів на логічні підсистеми у UML використовується діаграма пакетів. Цей тип діаграми дозволяє наочно представити організацію архітектури системи через групування пов’язаних елементів моделі – класів, інтерфейсів, підсистем – у пакети, що функціонують як логічні контейнери.

Пакет у UML виконує роль високорівневого компонента, який дозволяє впорядковувати елементи системи за функціональним або структурним принципом. Усі елементи, що включені до пакета, є його складовими, а отже, при видаленні пакета автоматично видаляються й усі його елементи. Це підсилює логічну цілісність моделі та забезпечує її цілеспрямовану організацію, що є критичним на етапі масштабування або супроводу складних систем.

Застосування діаграми пакетів особливо актуальне в умовах об'єктно-орієнтованої розробки, де структура додатка містить велику кількість класів і

модулів. У цьому випадку пакети дозволяють зменшити складність системи шляхом ієрархічної декомпозиції, сприяючи зрозумілій навігації та модульності.

На рис. 2.6 зображено діаграму пакетів програмного забезпечення соціальної медіа-платформи, в якій структура застосунку поділена на кілька логічно взаємопов’язаних частин. Центральним елементом є головний пакет «ПЗ соціальної мережі», який містить підпакети, відповідальні за окремі функціональні аспекти: інтерфейс користувача, модуль автентифікації, керування публікаціями, роботу з базою даних, обробку зображень, модерацію контенту тощо.

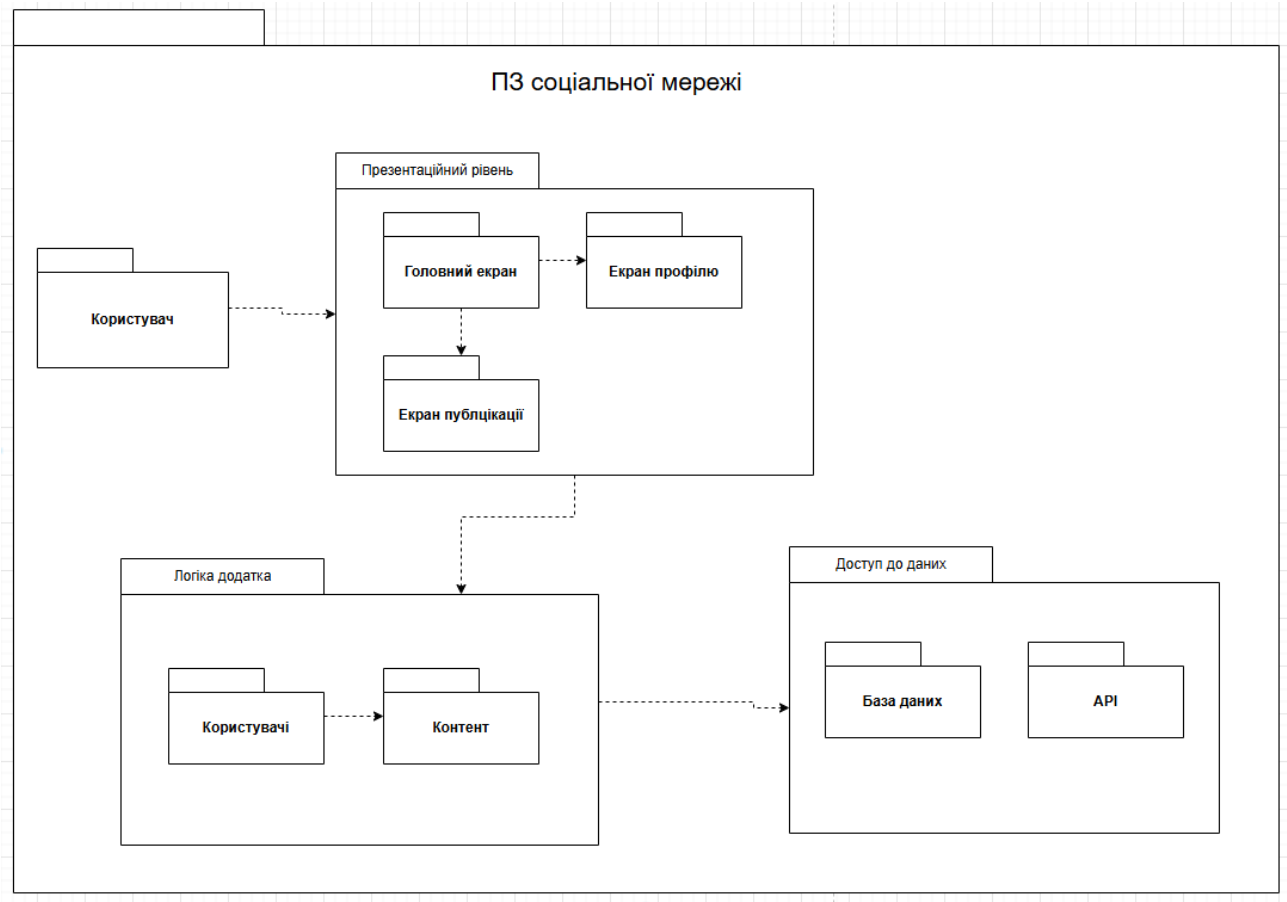


Рис. 2.6 Діаграма пакетів програмного забезпечення соціальної медіа-платформи

Діаграма пакетів слугує інструментом структурного впорядкування системи, дозволяючи логічно організувати її модулі та забезпечити чіткий розподіл відповідальностей між компонентами. Така модель підтримує

принципи слабкої зв'язаності та високої когерентності, що значно спрощує супровід, повторне використання компонентів і тестування системи. У контексті проєктування соціальної медіа-платформи вона відіграє ключову роль у формуванні високорівневої архітектури програмного продукту.

Для більш детального пояснення логіки побудови архітектури, відображеної на діаграмі пакетів (рис. 2.6), доцільно представити її компонентний склад у табличному форматі (табл.2.4). Це дозволяє систематизувати інформацію про кожен рівень системи, уточнити функціональне призначення відповідних пакетів та проілюструвати зв’язки між ними.

Таблиця 2.4

Ієрархічна структура пакетів програмного забезпечення соціальної медіа-платформи

№	Рівень системи	Назва пакета	Призначення та зв’язки
1	Головний контейнер	ПЗ соціальної мережі	Кореневий пакет, який об’єднує всі інші структурні компоненти
2	Презентаційний рівень	Користувач	Інтерфейсні елементи взаємодії користувача; має залежність до "Головний екран"
3		Головний екран	Основний екран додатку; залежить від "Екран публікації"
4		Екран профілю	Відображення та редагування персональної інформації користувача
5		Екран публікації	Створення/перегляд постів; має залежність до пакета "Контент" рівня логіки додатка
6	Логіка додатка	Користувачі	Управління автентифікацією, профілями, даними користувачів
7		Контент	Робота з публікаціями, коментарями, лайками; залежить від рівня доступу до даних
8	Доступ до даних	База даних	Сховище інформації про користувачів і контент; реалізоване у вигляді БД
9		API	Програмний інтерфейс взаємодії з даними для рівня логіки додатка

Структура таблиці охоплює основні рівні архітектури – презентаційний, логіки додатка та доступу до даних – і демонструє, як саме реалізується ієрархія модулів усередині програмного забезпечення. Такий підхід забезпечує наочність та спрощує аналіз архітектурних рішень.

Запропонована структура пакетів відображає чітке багаторівневе розділення відповідальностей у системі, що сприяє її модульності, зручності супроводу та масштабованості. Поділ на рівні презентації, бізнес-логіки та доступу до даних є типовим для архітектурних рішень із високою структурною організацією.

2.4 Діаграма компонентів

У рамках архітектурного моделювання програмного забезпечення важливою складовою є представлення взаємозв’язків між окремими частинами системи. Для цього у мові UML застосовується діаграма компонентів, яка дозволяє візуалізувати структуру програмного забезпечення як сукупність незалежних, замінюваних та ізольованих модулів, які взаємодіють між собою через чітко визначені інтерфейси.

Кожен компонент у такій діаграмі репрезентує логічну частину системи, що реалізує певну функціональність і при цьому взаємодіє з іншими виключно через контракти доступу – інтерфейси. Це забезпечує модульність, високу ступінь повторного використання та спрощує підтримку коду. Компоненти можуть містити як внутрішню логіку, так і бути проксі до зовнішніх сервісів або джерел даних. Саме така структура дозволяє проєктувати масштабовані й гнучкі програмні рішення.

На рис. 2.7 представлено діаграму компонентів соціальної медіа-платформи, архітектура якої поділена на чотири логічні рівні: Presentation Layer, Business Layer, Data Access Layer та External Services. Цей поділ забезпечує дотримання принципу розділення відповідальності та спрощує структурування коду за логічною приналежністю.

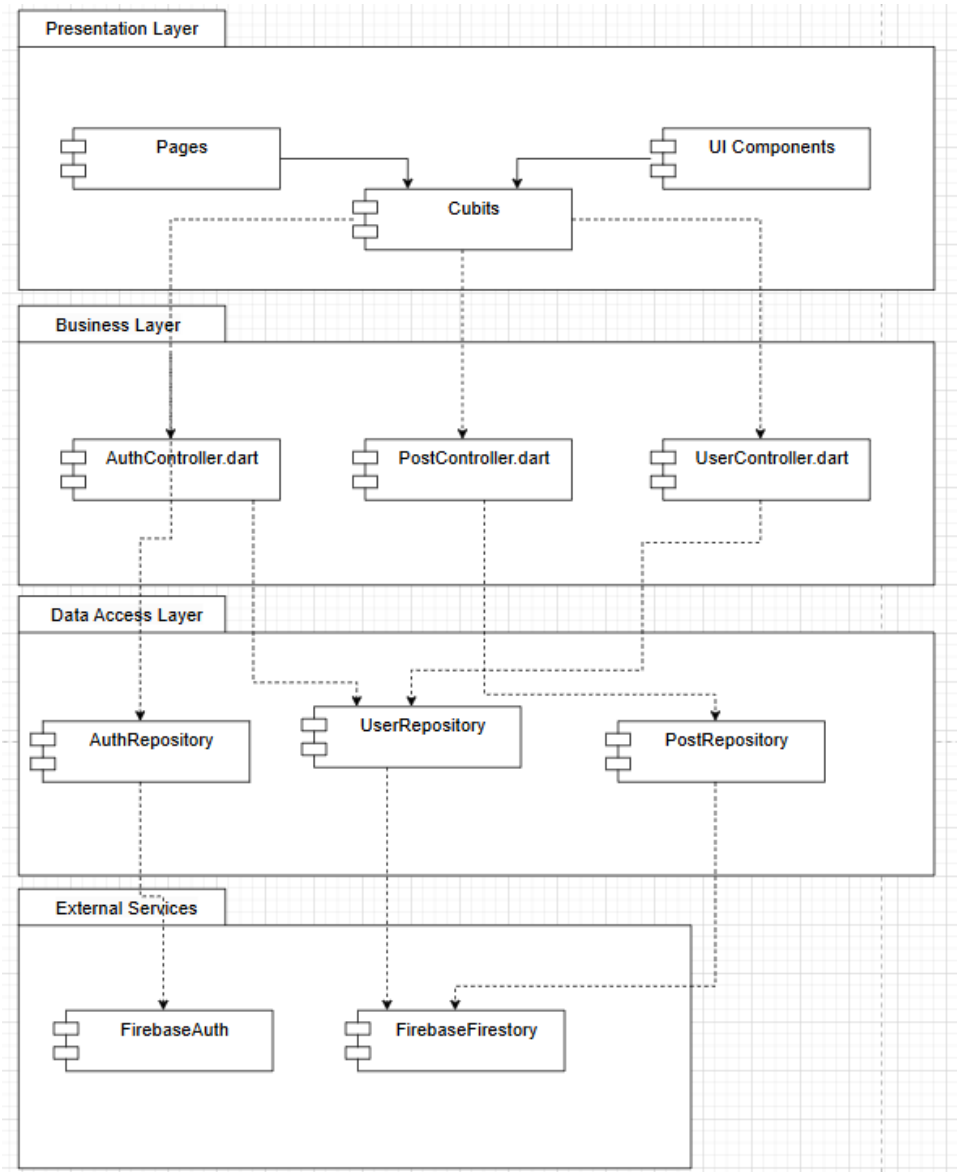


Рис.2.7 Діаграма компонентів для соціальної медіа-платформи

Для узагальнення вмісту кожного шару та їх взаємозв’язків у табл. 2.5 наведено опис усіх компонентів діаграми із зазначенням їх функціонального призначення, рівня розміщення та залежностей.

Згідно з представленою структурою, компоненти користувацького інтерфейсу взаємодіють із шаром управління станами (Cubits), які своєю чергою звертаються до бізнес-компонентів (Controllers). Ці компоненти обробляють бізнес-логіку й делегують доступ до даних репозиторіям, що інкапсулюють деталі взаємодії з Firebase-сервісами.

Таблиця 2.5

Структура та залежності компонентів програмного забезпечення соціальної мережі

№	Рівень	Назва компонента	Призначення	Залежність від інших рівнів
1	Presentation Layer	Pages	Екрани інтерфейсу користувача (головна, профіль, публікації)	Cubits
2	Presentation Layer	UI Components	Повторно використовувані елементи UI	Cubits
3	Presentation Layer	Cubits	Управління станами інтерфейсу, обробка подій	Business Layer
4	Business Layer	AuthController.dart	Логіка реєстрації, входу, виходу	AuthRepository
5	Business Layer	PostController.dart	Створення, редагування, видалення публікацій	PostRepository
6	Business Layer	UserController.dart	Управління профілями, підписками, оновлення інформації	UserRepository
7	Data Access Layer	AuthRepository	Доступ до даних автентифікації	FirebaseAuth
8	Data Access Layer	UserRepository	Обробка даних користувача	FirebaseFirestore
9	Data Access Layer	PostRepository	Доступ до публікацій	FirebaseFirestore
10	External Services	FirebaseAuth	Зовнішній сервіс автентифікації	–
11	External Services	FirebaseFirestore	Хмарна NoSQL-база даних для зберігання контенту	–

Така архітектура чітко поділяє зони відповідальності та забезпечує незалежність рівнів, що полегшує підтримку, тестування та розширення функціональності.

Побудована діаграма компонентів є надійною основою для технічного проєктування програмного забезпечення. Вона дозволяє узгодити логічну модель з реальним поділом на модулі, визначити залежності між ними та сформувати чітке уявлення про структуру взаємодії між частинами системи. Такий підхід сприяє забезпеченню масштабованості, модульності та гнучкості в подальшому циклі розробки соціальної медіа-платформи.

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Система управління інформаційною базою

Сучасна розробка інформаційних систем неможлива без використання ефективної системи управління базами даних (СУБД), яка забезпечує централізоване зберігання, захист і доступ до структурованої інформації. СУБД виступає не лише сховищем даних, а й інструментом управління їх цілісністю, узгодженістю та доступністю. Вона дозволяє зручно організовувати взаємодію між програмними компонентами системи та кінцевими користувачами, забезпечуючи необхідний рівень продуктивності та надійності.

Однією з ключових функцій СУБД є забезпечення захисту даних. Це реалізується через механізми автентифікації, авторизації, шифрування, моніторингу та відновлення після збоїв. Крім того, СУБД гарантує мінімізацію дублювання інформації та підтримку транзакцій, що критично важливо для підтримання актуальності та узгодженості даних. В архітектурі сучасних систем з високим навантаженням СУБД відіграє центральну роль, забезпечуючи швидкий обмін даними між користувачем та сервером [3].

У межах реалізації даного проєкту використано хмарну NoSQL СУБД Firebase, що є оптимальним рішенням для мобільних додатків із високою динамікою оновлення контенту, характерною для соціальних мереж. Firebase забезпечує не тільки зберігання, а й обробку мультимедійних даних, синхронізацію в реальному часі, керування правами доступу та інтеграцію з іншими сервісами (аутентифікація, аналітика, хмарні функції тощо).

У таблиці 3.1 узагальнено основні переваги використання Firebase як інформаційної платформи для даного програмного продукту. Firebase як хмарна СУБД особливо ефективна в умовах мобільного застосування завдяки своїй масштабованості, гнучкій структурі та підтримці великої кількості клієнтів одночасно.

Таблиця 3.1

Основні переваги Firebase у межах розробки соціальної платформи

№	Перевага	Опис
1	Висока продуктивність та масштабованість	Додаток може обробляти великий потік даних без затримок, що критично для соціальної мережі.
2	Зручне управління мультимедійним контентом	Firebase Storage дозволяє швидко завантажувати та зберігати фото та відео користувачів.
3	Автоматичне оновлення контенту	Firestore дозволяє миттєво оновлювати контент без необхідності запитів на сервер.
4	Менше часу на бекенд-розробку	Firebase значно зменшує час розробки, адже не потрібно створювати окремий сервер для обробки запитів.
5	Високий рівень безпеки	Firebase Security Rules дозволяють налаштувати детальні права доступу до даних, що важливо для конфіденційності особистих даних користувачів.

У контексті мобільного рішення її переваги проявляються ще виразніше – від простоти інтеграції до автоматичного керування навантаженням.

У таблиці 3.2 наведено розширений аналіз ключових переваг використання Firebase саме у сфері мобільної розробки.

Таблиця 3.2

Переваги Firebase у розробці мобільного додатку

№	Перевага	Опис
1	2	3
1	Гнучка архітектура NoSQL (Firestore та Realtime DB)	Firebase використовує документо-орієнтовану модель даних, яка добре підходить для зберігання постів, коментарів, лайків, профілів. Легко масштабується.
2	Синхронізація в реальному часі	Дані автоматично оновлюються у всіх користувачів без перезавантаження інтерфейсу, що критично для миттєвих реакцій (лайки, коментарі).
3	Хмарне зберігання мультимедіа (Firebase Storage)	Зручне зберігання та оптимізація зображень. Можлива автоматична обробка (наприклад, створення мініатюр) через Cloud Functions.

Таблиця 3.2 (продовження)

1	2	3
4	Проста авторизація (Firebase Authentication)	Підтримка входу через соціальні мережі (Google, Facebook тощо) та email/пароль. Спрощує реалізацію системи безпеки.
5	Безпека та масштабованість	Cloud Firestore Security Rules дозволяють гнучко налаштовувати доступ. База даних автоматично масштабується разом зі збільшенням кількості користувачів.
6	Аналітика та моніторинг (Firebase Analytics + Crashlytics)	Відстеження поведінки користувачів, виявлення та виправлення помилок, що сприяє підвищенню якості продукту.

Firebase як обрана система управління інформаційною базою у проєкті соціальної медіа-платформи повністю відповідає вимогам до масштабованості, швидкодії, зручності використання та безпеки. Завдяки своїй архітектурі вона дозволяє реалізувати складну взаємодію користувачів у режимі реального часу без перевантаження інтерфейсу або сервера. Хмарна модель управління, інтегровані інструменти для аналітики й авторизації, а також автоматична синхронізація контенту роблять цю СУБД ідеальним вибором для динамічного мобільного додатку з високим рівнем залучення користувачів.

Таким чином, вибір Firebase як основи для управління даними в межах проєкту був зумовлений не лише зручністю реалізації, а й архітектурною відповідністю сучасним вимогам до мобільних, кросплатформних і хмарноорієнтованих систем. Такий підхід дозволяє досягти високої якості сервісу при мінімальних затратах на розгортання, що особливо важливо для стартап-проєктів або платформ на етапі активного масштабування.

3.2 Розробка інформаційної бази

Розробка інформаційної бази для соціальної медіа-платформи базується на використанні хмарної NoSQL-технології – Firebase Cloud Firestore, яка є ключовим компонентом обраної архітектури. У традиційних реляційних системах можна чітко розділити логічний та фізичний рівень зберігання, однак у

випадку Firebase фізична реалізація структури даних прихована від розробника і повністю управляється інфраструктурою Google Cloud. Тому умовно фізичний рівень у цій системі розглядається як логічна організація колекцій, документів і полів, що зберігають у собі фактичні значення атрибутів [14].

Структура Firestore побудована за ієрархічною моделлю: Collection → Document → Fields. Кожна сутність, що моделює об'єкт предметної області (наприклад, користувач, пост, коментар), зберігається у вигляді документа всередині відповідної колекції. Це забезпечує гнучку структуру даних, можливість динамічного розширення та масштабування. На прикладі створення колекції users на рис. 3.1 можна побачити, як організовано зберігання користувацьких профілів.

Рис. 3.1 Створена колекція “users”

Щоб відобразити сутність AppUser у вигляді об'єкта, у програмному коді створюється відповідний клас Dart, який описує ключові атрибути користувача. Його реалізація наведена на рис. 3.2.

```dart
class AppUser {
  final String uid;
  final String email;
  final String name;
  final bool isAdmin;

  AppUser({
    requir Loading... d,
    required this.email,
    required this.name,
    this.isAdmin = false, // за замовчуванням користувач не адмін
  });

  // Конвертація в JSON
  Map<String, dynamic> toJson() {
    return {'uid': uid, 'email': email, 'name': name, 'isAdmin': isAdmin};
  }

  // Створення з JSON
  factory AppUser.fromJson(Map<String, dynamic> jsonUser) {
    return AppUser(
      uid: jsonUser['uid'],
      email: jsonUser['email'],
      name: jsonUser['name'],
      isAdmin: jsonUser['isAdmin'] ?? false,
    );
  }
```

Рис. 3.2 Реалізація сутності AppUser в коді

Визначені поля класу відповідають збереженим полям у документі Firestore, що дозволяє однозначно синхронізувати дані між застосунком і базою.

Під час збереження даних користувача структура документа автоматично перетворюється у формат JSON, що є стандартним представленням для взаємодії з NoSQL-базами. У результаті маємо дані, що представлені в серіалізованому вигляді, як зображено на рис. 3.3.

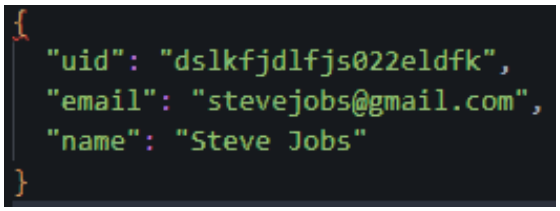


Рис.3.3 Представлення даних користувача у форматі JSON

Для взаємодії з Firebase реалізується інтерфейс репозиторію AuthRepo, який задає контракт для всіх дій, пов'язаних із автентифікацією користувачів. Цей інтерфейс не містить конкретної реалізації, а лише визначає методи, які повинні бути реалізовані в майбутніх класах (рис. 3.4).

```
abstract class AuthRepo {
  Future<AppUser?> loginWithEmailPassword(String email, String password);
  Future<AppUser?> registerWithEmailPassword(
    String name,
    String email,
    String password,
  );
  Future<void> logout();
  Future<AppUser?> getCurrentUser();
}
```

Рис. 3.4 Абстрактний репозиторій для автентифікації

На рис. 3.4 представлено абстрактний репозиторій AuthRepo, який виступає інтерфейсом для взаємодії з системою автентифікації. Він визначає набір методів без реалізації, які мають бути імплементовані у відповідних класах. Такий підхід дозволяє легко змінювати механізм автентифікації без впливу на інші частини системи, забезпечуючи гнучкість, модульність і дотримання принципів чистої архітектури.

Потім створюється конкретна реалізація – клас FirebaseAuthRepo, який реалізує описані методи інтерфейсу AuthRepo і безпосередньо взаємодіє з Firebase Authentication та Firestore. Ключовим завданням класу є автентифікація користувачів, зберігання їхніх даних після реєстрації, обробка запитів на вхід, вихід і перевірку поточного користувача (рис. 3.5).

```dart
class FirebaseAuthRepo implements AuthRepo {
  final FirebaseAuth firebaseAuth = FirebaseAuth.instance;
  final FirebaseFirestore firebaseFirestore = FirebaseFirestore.instance;

  @override
  Future<AppUser?> loginWithEmailPassword(String email, String password) async {…

  @override
  Future<AppUser?> registerWithEmailPassword(…

  @override
  Future<void> logout() async {
    await firebaseAuth.signOut();
  }

  @override
  Future<AppUser?> getCurrentUser() async {
    final FirebaseUser = firebaseAuth.currentUser;

    if (FirebaseUser == null) {
      return null;
    }

    DocumentSnapshot userDoc =
        await firebaseFirestore.collection("users").doc(FirebaseUser.uid).get();

    if (!userDoc.exists) {
      return null;
    }

    return AppUser(
      uid: FirebaseUser.uid,
      email: FirebaseUser.email!,
      name: userDoc['name'],
      isAdmin: userDoc['isAdmin'] ?? false,
    );
  }
}
```

Рис. 3.5 Клас FirebaseAuthRepo для керування автентифікацією

На рис. 3.5 показано клас FirebaseAuthRepo, який реалізує функціональність аутентифікації користувачів на основі сервісів Firebase Authentication та Firestore. Основні методи цього класу включають: вхід у систему (логін) із використанням електронної пошти та пароля з подальшим

отриманням даних користувача з бази; реєстрацію нового користувача із збереженням його профілю у Firestore; вихід із системи; а також перевірку наявності поточно авторизованого користувача та завантаження його даних. Ця реалізація забезпечує повний цикл роботи з обліковими записами в межах хмарної платформи.

У розробленій реалізації враховані всі основні сценарії взаємодії з інформаційною базою, що забезпечує надійний обмін даними між інтерфейсом додатку та Firestore. Кожна сутність зберігається у відповідній колекції з чітко визначеним форматом документа. На рис. 3.6 подано повну структуру створеної колекції users, яка включає всі ключові поля, необхідні для функціонування профілю користувача.

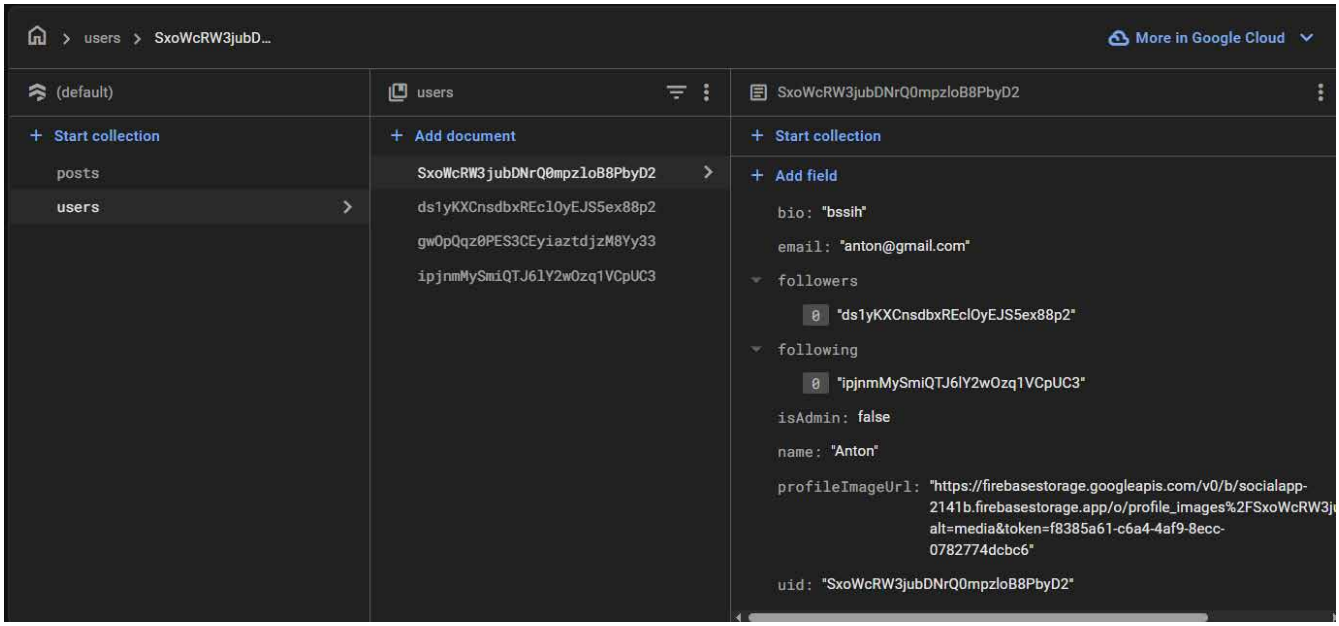


Рис. 3.6 Повна структура колекції “users” у Firestore

Розробка інформаційної бази на основі Firebase Cloud Firestore забезпечила реалізацію гнучкої, масштабованої та надійної системи зберігання даних, що повністю відповідає вимогам сучасних соціальних додатків. Структура колекцій та документів дозволяє ефективно організувати збереження профілів, постів, коментарів та інших елементів взаємодії, при цьому зберігаючи простоту доступу та високу швидкодію.

Завдяки використанню хмарної NoSQL-бази Firestore у поєднанні з чітко спроєктованими класами сутностей і репозиторіями, було створено логічну та ефективну інформаційну інфраструктуру для підтримки соціальної активності користувачів у мобільному додатку. Такий підхід забезпечує не лише стабільність і масштабованість, а й легкість у супроводі та подальшому розширенні функціоналу системи.

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

Побудова якісного мобільного застосунку передбачає застосування комплексу взаємопов’язаних технологій, кожна з яких виконує свою функцію на різних етапах життєвого циклу програмного забезпечення. При створенні мобільного застосунку соціальної мережі було обрано інструментарій, що поєднує сучасні рішення для дизайну, кодування, управління даними та реалізації інтерфейсу. Такий підхід забезпечує не лише високу продуктивність розробки, а й відповідність функціоналу вимогам користувача.

Для створення користувацького інтерфейсу (UI) та користувацького досвіду (UX) застосовано Figma, що являє собою – хмарний графічний редактор, який дозволив швидко розробити макети екранів, спроєктувати структуру навігації та протестувати інтерфейс через інтерактивні прототипи. Завдяки своїм можливостям щодо створення адаптивного дизайну Figma стала оптимальним вибором для реалізації багатоплатформного інтерфейсу.

Програмування додатку здійснювалося у середовищі Visual Studio Code, яке відзначається легкістю, швидкістю роботи та розширюваністю. VS Code забезпечує повну інтеграцію з мовою Dart і фреймворком Flutter, що значно полегшує розробку, тестування й налагодження коду. Завдяки інтеграції з Git розробник має змогу ефективно управляти версіями, а автозаповнення та аналітика коду прискорюють реалізацію логіки застосунку [18].

Для бекенду було вирішено використовувати Firebase, набір хмарних сервісів від Google. Firebase пропонує комплексний набір інструментів,

необхідних для розробки мобільних застосунків без клопоту з управлінням власною інфраструктурою бекенду. Він включає такі важливі сервіси, як Firestore для хмарних баз даних, Firebase Authentication для управління користувачами та Firebase Storage для обробки файлів. Кожен з цих елементів значно спрощує та прискорює процес реалізації функціональності, пов'язаної з користувачами та даними [16].

Реалізація застосунку відбувалася за допомогою Flutter – кросплатформного SDK, що дозволяє створювати нативні застосунки для Android та iOS з однієї кодової бази. Flutter забезпечує швидке оновлення інтерфейсу без втрати стану (hot reload), пропонує гнучкий набір віджетів і генерує високопродуктивний нативний код [13]. Це дозволяє досягти єдиного стандарту дизайну та поведінки застосунку на різних платформах.

У таблиці 3.3 наведено перелік основних інструментів, що були використані в межах розробки, із зазначенням їх ролі та переваг.

Таблиця 3.3

Обраний інструментарій для реалізації мобільного застосунку

№	Інструмент	Призначення	Ключові переваги
1	Figma	Створення макетів інтерфейсу, прототипів і візуального дизайну	Адаптивність, інтерактивність, спільна робота в реальному часі
2	Visual Studio Code	Розробка коду, інтеграція з Flutter, підтримка розширень	Легкість, багатофункціональність, підтримка Dart і Git
3	Flutter	Кросплатформна розробка інтерфейсу й логіки застосунку [17]	Єдина кодова база, нативна продуктивність, hot reload
4	Firebase Firestore	Зберігання даних у хмарі, робота з документами	Миттєве оновлення, масштабованість, простота структури
5	Firebase Authentication	Автентифікація та управління користувачами	Інтеграція з соцмережами, безпечна авторизація
6	Firebase Storage	Збереження зображень і відео користувачів	Хмарна обробка, інтеграція з Firestore

Обраний набір інструментів – Figma для дизайну, VS Code для розробки, Firebase для бекенду та Flutter для створення кросплатформного застосунку – є сучасним, ефективним та добре інтегрованим. Ця комбінація дозволяє мені ефективно досягати цілей, які я поставив перед розробкою моєї платформи соціальних мереж.

Кожна частина цього технологічного стеку сприяє більш плавному робочому процесу, роблячи мій процес розробки більш оптимізованим та продуктивним. Варто відзначити ефективну інтеграцію цих інструментів для створення застосунку, який не лише добре функціонує, але й забезпечує приємний досвід для користувачів. Обравши правильні інструменти, я заклав міцну основу для мого проєкту, гарантуючи, що я зможу впевнено подолати майбутні виклики.

Інтеграція Figma, Visual Studio Code, Flutter та Firebase у межах одного проєкту дала змогу реалізувати продуктивне, масштабоване і сучасне рішення для соціального застосунку. Такий вибір дозволив оптимізувати процес розробки, забезпечити високу якість продукту й підготувати технічне підґрунтя для подальшого розширення функціоналу системи.

3.4 Алгоритмізація та програмування програмних модулів

Ефективна розробка програмного забезпечення неможлива без чіткого розмежування логіки та відповідального підходу до модульності. Для побудови внутрішньої архітектури застосунку було обрано підхід Bloc (Business Logic Component), який дозволяє організувати систему у вигляді окремих функціональних блоків. Кожен блок містить набір модулів: логіку доступу до даних, доменну модель і інтерфейсну частину. Така структура дає змогу чітко розмежувати відповідальність, полегшити тестування й масштабування системи.

Загальна структура проєкту реалізована відповідно до архітектури "шарів" і представлена на рис. 3.7. У цій схемі кожен логічний блок (наприклад, post) містить окремі підмодулі для обробки даних, визначення сутностей та взаємодії з UI.

На прикладі модуля post, структура якого зображена на рис. 3.8, розглянемо деталізацію реалізації обробки даних публікацій. Тут виділено три ключові частини: data, domain та presentation. Особливу увагу приділено модулю data, який реалізує всі основні функції взаємодії з Firebase Firestore через клас FirebasePostRepo.

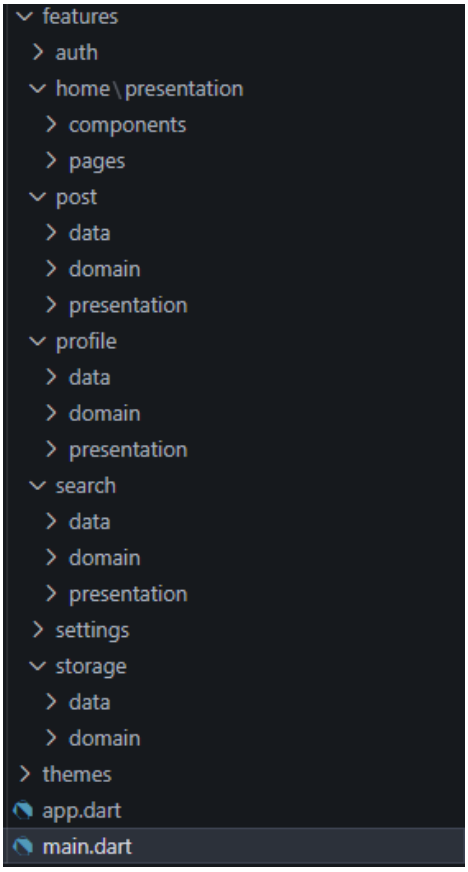


Рис. 3.7 Архітектура проєкту

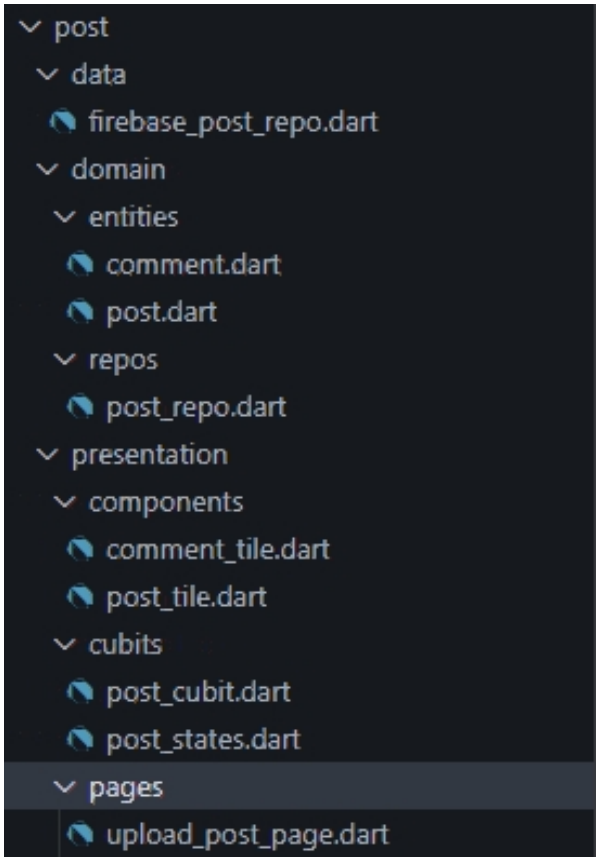
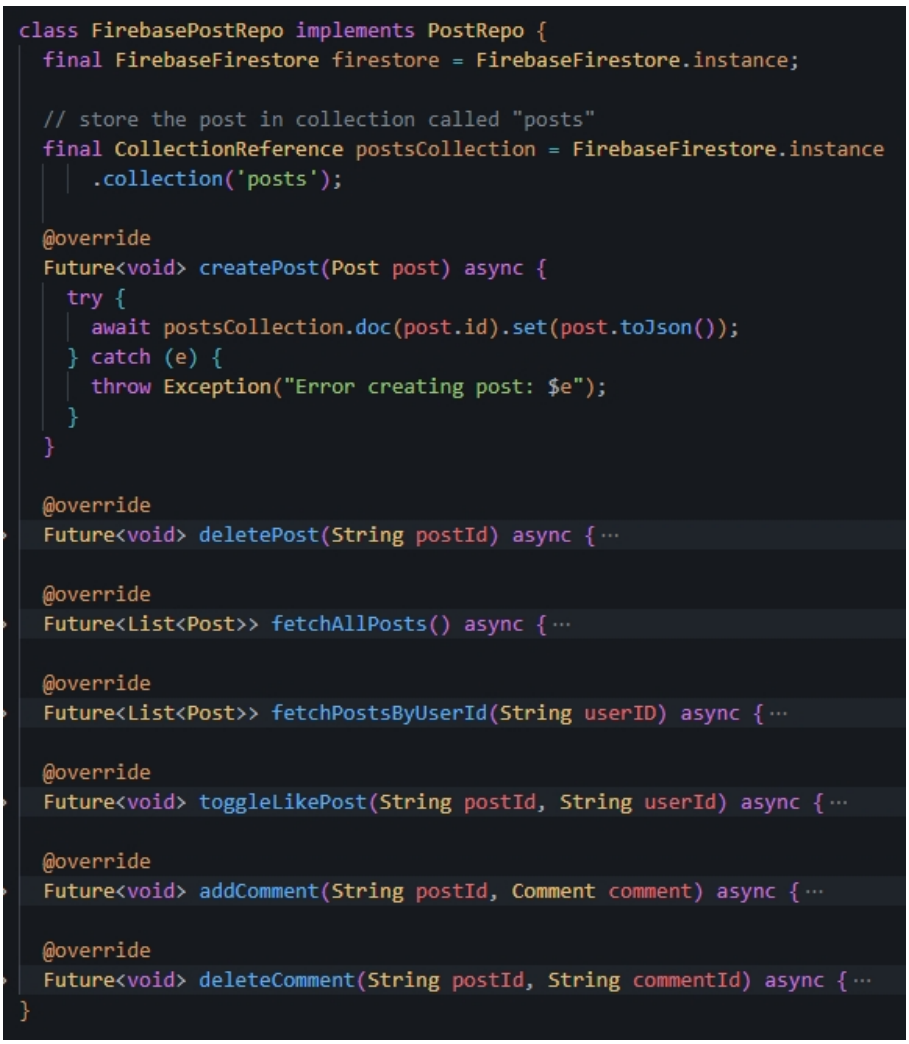


Рис. 3.8 Структура блоку “post”

Модуль data відповідає за реалізацію доступу до даних, тобто за взаємодію з зовнішніми джерелами даних (в даному випадку, Firebase Firestore) для виконання операцій, пов'язаних з постами. Клас FirebasePostRepo конкретно реалізує інтерфейс PostRepo (який знаходиться в доменному шарі) і надає функціональність для створення, видалення, отримання постів (всіх та за ID користувача), додавання/видалення лайків та коментарів.

Функціональність класу FirebasePostRepo охоплює повний набір CRUD-операцій для постів і коментарів, включаючи створення, видалення, отримання даних, керування лайками й коментарями. Лістинг коду подано на рис. 3.9.



```
class FirebasePostRepo implements PostRepo {
  final FirebaseFirestore firestore = FirebaseFirestore.instance;

  // store the post in collection called "posts"
  final CollectionReference postsCollection = FirebaseFirestore.instance
      .collection('posts');

  @override
  Future<void> createPost(Post post) async {
    try {
      await postsCollection.doc(post.id).set(post.toJson());
    } catch (e) {
      throw Exception("Error creating post: $e");
    }
  }

  @override
  Future<void> deletePost(String postId) async {···

  @override
  Future<List<Post>> fetchAllPosts() async {···

  @override
  Future<List<Post>> fetchPostsByUserId(String userID) async {···

  @override
  Future<void> toggleLikePost(String postId, String userId) async {···

  @override
  Future<void> addComment(String postId, Comment comment) async {···

  @override
  Future<void> deleteComment(String postId, String commentId) async {···
}
```

Рис. 3.9 Реалізація класу FirebasePostRepo (фрагмент)

Для зручності всі ключові алгоритми, реалізовані в FirebasePostRepo, узагальнено в таблиці 3.4 із коротким описом кожного методу.

Таблиця 3.4

Алгоритмічна реалізація функцій модуля роботи з постами

№	Метод	Короткий опис алгоритму реалізації
1	2	3
1	createPost(Post post)	Створює документ у колекції posts на основі JSON-представлення об’єкта Post
2	deletePost(String postId)	Видаляє документ за ID з колекції posts
3	fetchAllPosts()	Отримує всі пости, відсортовані за часом у зворотному порядку; конвертує документи у список об’єктів
4	fetchPostsByUserId(userId)	Повертає пости, автором яких є користувач із зазначеним userId

Таблиця 3.4 (продовження)

1	2	3
5	toggleLikePost(postId, userId)	Додає або видаляє лайк у списку likes відповідного поста, залежно від наявності userId у цьому списку
6	addComment(postId, comment)	Додає новий коментар до списку comments документа поста
7	deleteComment(postId, commentId)	Видаляє коментар із документа поста за ідентифікатором commentId

У межах доменного шару визначено базові сутності, які описують структуру об’єктів предметної області. Наприклад, сутність Post, представлена у файлі post.dart, визначає модель публікації з відповідними властивостями та методами серіалізації. Окрім того, у файлі post_repo.dart описано інтерфейс PostRepo, який задає перелік функцій для взаємодії з постами без зазначення конкретної реалізації.

У таблиці 3.5 наведено короткий опис вмісту обох зазначених файлів.

Таблиця 3.5

Опис файлів сутності поста та інтерфейсу репозиторію

№	Файл	Тип	Короткий опис
1	post.dart	Сутність	Визначає структуру об'єкта "Пост" (його властивості: id, автор, текст, зображення, час, лайки, коментарі) та методи для перетворення між об'єктом Post і JSON-форматом (для роботи з даними).
2	post_repo.dart	Інтерфейс	Описує, які дії можна виконувати з постами (створення, видалення, отримання всіх, отримання за автором, лайк, коментування, видалення коментаря), не вказуючи, як саме ці дії будуть реалізовані. Слугує контрактом для шару даних.

Використання чіткої модульної архітектури з розмежуванням на доменну модель, джерело даних та інтерфейсну частину дозволяє ефективно реалізувати основну бізнес-логіку застосунку, підтримуючи масштабованість та розширюваність системи (рис. 3.10). Асинхронна природа операцій з Firestore

дозволяє забезпечити високу продуктивність навіть при великій кількості одночасних запитів.

```dart
class Post {
  final String id;
  final String userId;
  final String userName;
  final String text;
  final String imageUrl;
  final DateTime timestamp;
  final List<String> likes;
  final List<Comment> comments;
  Post({
    required this.id,
    required this.userId,
    required this.userName,
    required this.text,
    required this.imageUrl,
    required this.timestamp,
    required this.likes,
    required this.comments,
  });
  Post copyWith({String? imageUrl}) {
    return Post(
      id: id,
      userId: userId,
      userName: userName,
      text: text,
      imageUrl: imageUrl ?? this.imageUrl,
      timestamp: timestamp,
      likes: likes,
      comments: comments,
    );
  }
  Map<String, dynamic> toJson() {
    return {
      'id': id,
      'userId': userId,
      'name': userName,
      'text': text,
      'imageUrl': imageUrl,
      'timestamp': Timestamp.fromDate(timestamp),
      'likes': likes,
      'comments': comments.map((comment) => comment.toJson()).toList(),
    };
  }
```

Рис. 3.10 Фрагмент коду сутності Post

Алгоритмізація програмних модулів із застосуванням шаблону Bloc та реалізація взаємодії з Firebase на основі контрактно-орієнтованого підходу дозволили створити гнучку, легко підтримувану архітектуру. Кожен модуль системи виконує чітко визначену роль і взаємодіє з іншими компонентами через стандартизовані інтерфейси, що відповідає принципам чистої архітектури та забезпечує якісну реалізацію функціоналу соціального застосунку.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

У процесі створення сучасного програмного забезпечення важливим етапом життєвого циклу є тестування, оскільки саме воно дозволяє переконатися у відповідності реалізованої системи функціональним вимогам, забезпечити її стабільність, надійність та зручність використання. Для соціальної медіа-платформи, яка передбачає активну взаємодію між користувачами, оперативну обробку запитів, завантаження та зберігання контенту, особливо критичними є аспекти безпеки, продуктивності та сумісності з різними пристроями. Тестування дозволяє виявити можливі помилки ще на етапі розробки, попередити появу збоїв при роботі у реальному середовищі та вдосконалити інтерфейс користувача з урахуванням зворотного зв’язку.

Вибір методів тестування обумовлюється як технічними характеристиками застосунку, так і специфікою взаємодії з кінцевими користувачами. Зокрема, функціональне тестування є базовим і дозволяє перевірити виконання основних сценаріїв роботи системи, тоді як автоматизовані тести відіграють важливу роль у забезпеченні безперервної інтеграції та підтримки. Окрему увагу приділяють тестуванню продуктивності, яке дозволяє оцінити здатність системи обробляти навантаження, та юзабіліті-тестуванню, що фокусується на зручності взаємодії з інтерфейсом [10].

Таким чином, комплексне застосування різних видів тестування дає змогу забезпечити всебічну перевірку працездатності, безпеки та стабільності мобільного застосунку соціального спрямування відповідно до принципів, викладених у стандарті IEEE 829 [11]. Узагальнені відомості про методи тестування, які застосовуються в рамках розробки платформи, наведено у таблиці 4.1.

Таблиця 4.1

Методи тестування програмного забезпечення соціальної медіа-платформи

№	Метод тестування	Характеристика методу
1	Функціональне тестування	Перевіряє відповідність дій користувача очікуваній поведінці системи на рівні UI.
2	Тестування продуктивності	Оцінює здатність системи ефективно працювати при високому навантаженні.
3	Тестування безпеки	Аналізує механізми захисту даних і стійкість до зовнішніх атак та несанкціонованого доступу.
4	Тестування сумісності	Перевіряє коректність роботи додатку на різних пристроях, екранах і версіях ОС.
5	Юзабіліті-тестування	Виявляє труднощі у користуванні інтерфейсом та оцінює його інтуїтивність.
6	Автоматизоване тестування	Дозволяє швидко перевіряти критичний функціонал після змін у коді для запобігання регресіям.

Примітка: Складено класифікації методів тестування здійснено з урахуванням методичних підходів з [12; 15].

Запропонований комплекс методів тестування дозволяє оцінити систему як із технічної, так і з користувацької точок зору. Особливу увагу на поточному етапі було приділено функціональному тестуванню, що дало змогу переконатися у працездатності ключових сценаріїв взаємодії з додатком. У подальшому застосування автоматизованого тестування і тестів продуктивності підвищить масштабованість та якість підтримки платформи [20].

На завершальному етапі розробки соціальної медіа-платформи важливим кроком є перевірка коректності реалізованого функціоналу, відповідності поведінки додатку вимогам, а також забезпечення позитивного користувацького досвіду. З цією метою було застосовано функціональне тестування, яке передбачає перевірку основних сценаріїв взаємодії користувача з інтерфейсом додатку без прямого втручання у програмний код. Такий підхід дозволяє

зосередитися на відповідності фактичних результатів очікуваним при виконанні дій, що симулюють реальне використання системи.

Одним із ключових елементів тестування стала перевірка створення публікації. Для цього було здійснено перехід до сторінки профілю користувача, де на початковому етапі стрічка постів є порожньою (рис. 4.1). Після цього здійснюється навігація до форми створення нової публікації (рис. 4.2).

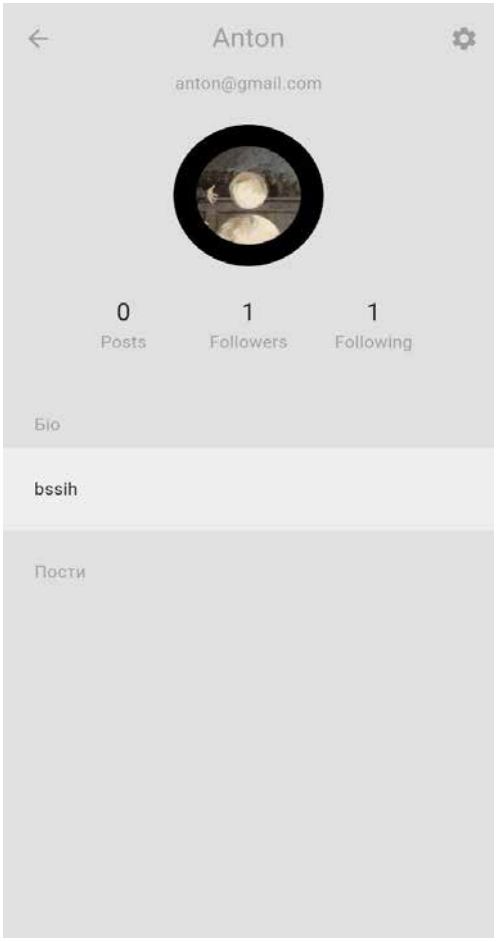


Рис. 4.1 Профіль користувача без постів

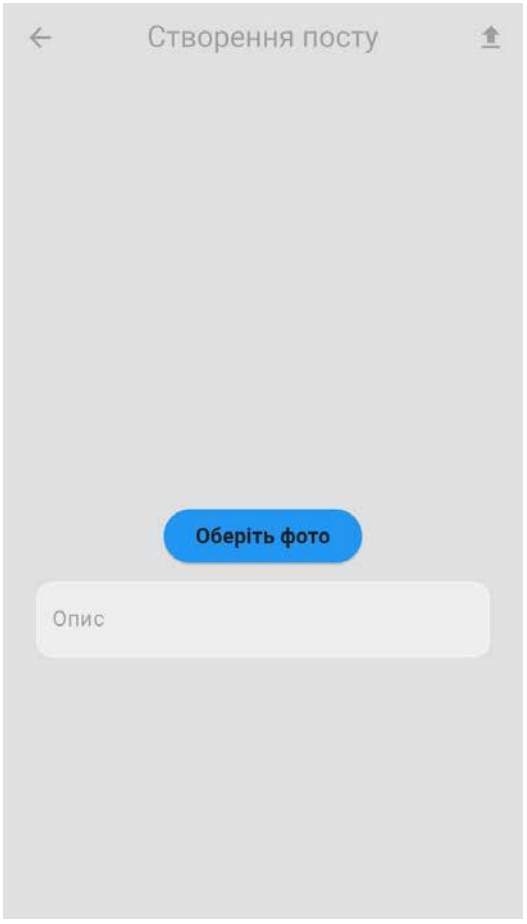


Рис. 4.2 Інтерфейс створення нового посту

Під час тестування виявлено, що при порушенні вимог до обов’язкових полів (наприклад, якщо прикріплено зображення без тексту або навпаки) система коректно реагує на помилки введення, попереджаючи користувача про необхідність заповнення обох полів (рис. 4.3).

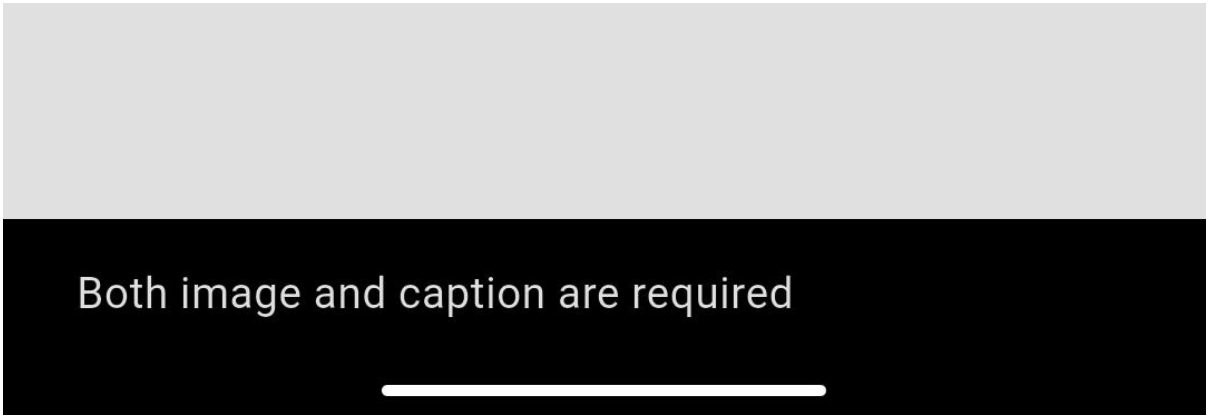


Рис. 4.3 Повідомлення про помилку під час створення публікації

У разі дотримання всіх умов публікація успішно зберігається та автоматично відображається як у профілі користувача, так і в стрічці новин (рис. 4.4). Це свідчить про коректне виконання логіки додавання поста та оновлення інтерфейсу після завершення операції.

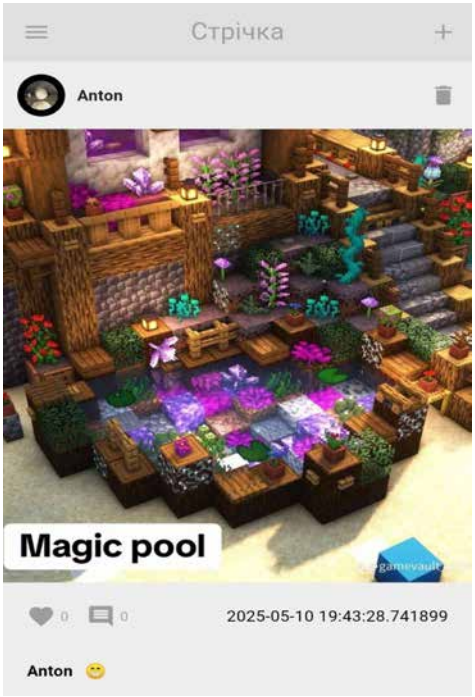


Рис.4.4 Відображення нового поста у стрічці новин

Основна частина реалізації цієї логіки наведена у фрагменті коду нижче. Метод createPost, що викликається при підтвердженні створення публікації,

виконує перевірку наявності зображення, завантажує його до хмарного сховища, формує об’єкт публікації й викликає метод оновлення стрічки (рис. 4.5).

```
Future<void> createPost(Post post, {String? imagePath}) async {
    String? imageUrl;
    try {
        if (imagePath != null) {
        emit(PostUploading());
        imageUrl = await storageRepo.uploadPostImageMobile(imagePath, post.id);
      }
      final newPost = post.copyWith(imageUrl: imageUrl);
      postRepo.createPost(newPost);
      fetchAllPosts();
    } catch (e) {
      emit(PostsError("Failed to create post: $e"));
    }
  }
```

Рис.4.5 Листинг метод оновлення стрічки

Для оновлення стрічки після створення поста викликається асинхронний метод fetchAllPosts, який наведений на рис. 4.6.

```
Future<void> fetchAllPosts() async {
    try {
      emit(PostsLoading());
      final posts = await postRepo.fetchAllPosts();
      emit(PostsLoaded(posts));
    } catch (e) {
      emit(PostsError("Failed to fetch posts: $e"));
    }
  }
```

Рис.4.6 Листинг виклику асинхронного методу fetchAllPosts

Окрім перевірки публікацій, функціональне тестування також охоплювало оновлення профілю користувача, зокрема зміну аватарки та біографії. Перед оновленням здійснюється валідація – перевіряється наявність користувача в базі даних та правильність переданих параметрів. Якщо зображення завантажене успішно, нові дані зберігаються у Firestore, після чого викликається оновлення

локального профілю. Відповідна логіка представлена у наступному фрагменті, який наведено на рис. 4.7.

```
Future<void> updateProfile({
    required String uid,
    String? newBio,
    String? imageMobilePath,
  }) async {
    emit(ProfileLoading());
    try {
      final currentUser = await profileRepo.fetchUserProfile(uid);
      if (currentUser == null) {
        emit(ProfileError("Failed to fetch user to profile update"));
        return;
      }
      String? imageDownloadUrl;
      if (imageMobilePath != null) {
          imageDownloadUrl = await storageRepo.uploadProfileImageMobile(
          imageMobilePath,
          uid,
        );
        if (imageDownloadUrl == null) {
          emit(ProfileError("Failed to upload image"));
        }
      }
      final updatedProfile = currentUser.copyWith(
        newBio: newBio ?? currentUser.bio,
        newProfileImageUrl: imageDownloadUrl ?? currentUser.profileImageUrl,
      );
      await profileRepo.updateProfile(updatedProfile);
      await fetchUserProfile(uid);
    } catch (e) {
      emit(ProfileError("Error updating profile: " + e.toString()));
    }
  }
```

Рис.4.7 Листинг виклику оновлення локального профілю

Після успішного оновлення даних інтерфейс користувача миттєво відображає зміни, що засвідчує узгоджену роботу між бізнес-логікою, збереженням інформації у Firestore та її подальшою візуалізацією (рис. 4.8).

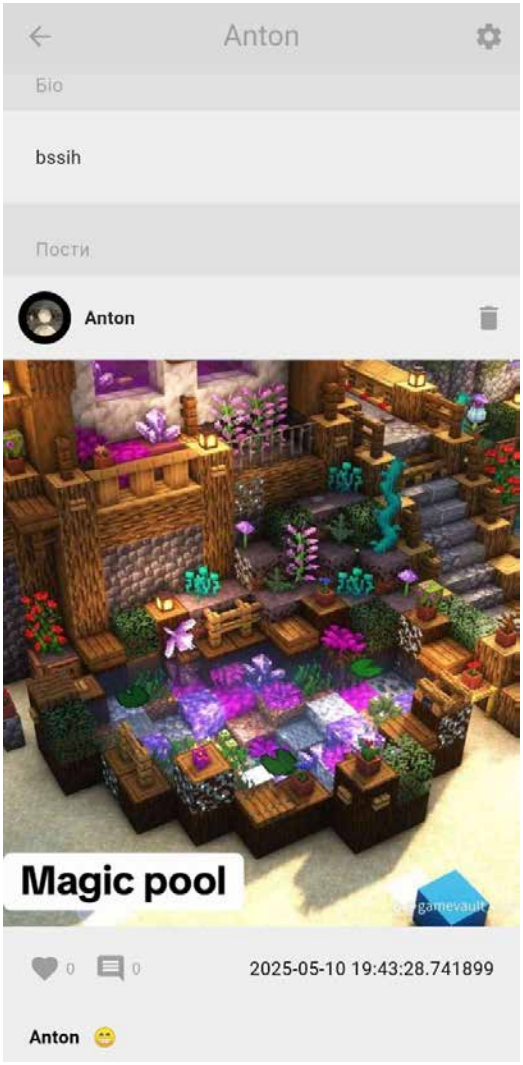


Рис.4.8 Оновлений профіль користувача після редагування

Відповідно функціональне тестування створення публікації було проведено успішно, усе працює добре та відображення UI частини здійснюється належним чином.

Таким чином, результати функціонального тестування підтвердили стабільну роботу реалізованих сценаріїв створення та редагування контенту. Застосування асинхронних методів і управління станами через Bloc забезпечує належну реакцію інтерфейсу на зміну даних, а вбудовані перевірки – цілісність і достовірність взаємодій. Проведене тестування є необхідною умовою для подальшого розгортання системи та її масштабування, підтверджуючи її відповідність основним експлуатаційним вимогам.

4.2 Вимоги до апаратного та програмного забезпечення

Успішне функціонування розробленої соціальної медіа-платформи потребує чітко визначених умов розгортання, що охоплюють як апаратні ресурси, так і програмні компоненти. Правильно сформульовані вимоги забезпечують не лише коректну роботу системи, але й її стабільність, масштабованість та безпечне обслуговування в процесі експлуатації. У цьому підрозділі розглянуто мінімальні та рекомендовані параметри, необхідні для налаштування та підтримки програмного комплексу з боку власника системи.

З огляду на архітектуру платформи, що базується на хмарних сервісах Firebase і кросплатформному фреймворку Flutter, основне навантаження припадає на мобільні пристрої кінцевих користувачів і хмарну інфраструктуру, яка забезпечує зберігання, автентифікацію, обробку та доставку даних. З боку власника системи (тобто особи або організації, яка адмініструє сервіс) ключові вимоги включають наявність стабільного інтернет-з’єднання, облікового запису Google з доступом до Firebase Console, а також комп’ютера з встановленим середовищем розробки (наприклад, Android Studio або Visual Studio Code) для супроводу та обслуговування проєкту. Важливим є також моніторинг продуктивності через консоль Firebase та інструменти для аналізу аналітики та безпеки.

Для кращого розуміння топології системи та розподілу її компонентів представлено діаграму розгортання (рис. 9), яка візуалізує розміщення основних функціональних модулів, взаємозв’язки між клієнтськими пристроями та хмарними сервісами, а також ролі користувача і адміністратора системи.

На діаграмі показано, що мобільний клієнт встановлюється на Android або iOS-пристрої, з якого здійснюється взаємодія з серверною частиною, реалізованою на основі Firebase. Сюди входять сервіси аутентифікації, бази даних Firestore, хмарне сховище та функції Cloud Functions, які обробляють події та виконують фонову логіку.

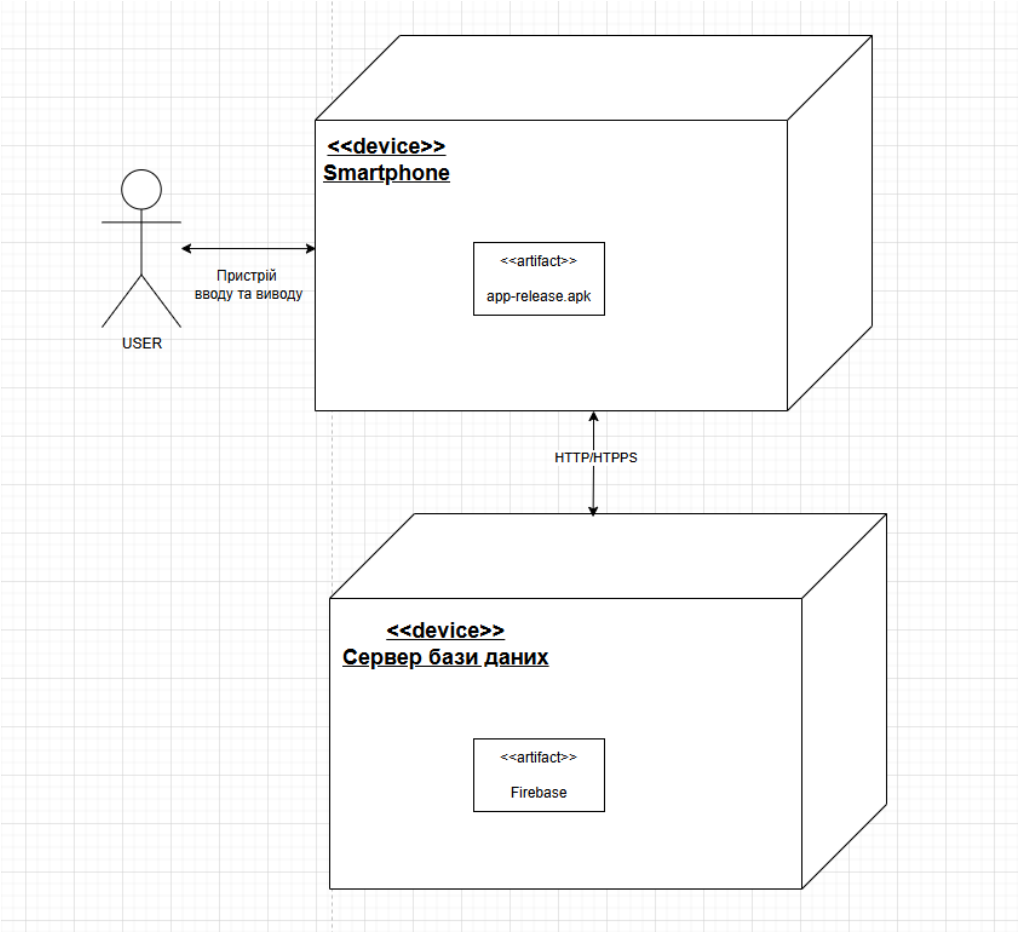


Рис.4.9 Діаграма розгортання системи

Адміністратор має доступ до консольної частини проєкту, де здійснює конфігурацію, моніторинг та аудит діяльності системи. Така архітектура забезпечує швидке масштабування, високу доступність і гнучкість при обслуговуванні, що є критичними перевагами для соціальних платформ із динамічним навантаженням.

Після аналізу діаграми розгортання, представленої на рис. 4.9, доцільним є деталізований опис ролей, функціональних властивостей та технічних характеристик основних компонентів системи. Це дозволяє систематизувати вимоги до апаратного та програмного забезпечення з урахуванням взаємодії між користувачем, мобільним клієнтом і серверною інфраструктурою. Зведені відомості подано у таблиці 4.2.

Таблиця 4.2

Вимоги до апаратного та програмного забезпечення соціальної медіа-платформи

№	Компонент	Тип пристрою / середовища	Опис ролі	Програмні артефакти / сервіси	Основні функції
1	Користувач (User)	Людина	Кінцевий користувач системи	–	Перегляд стрічки, створення постів, вподобання, коментування
2	Мобільний пристрій	<<device>> Смартфон	Клієнтська частина	app-release.apk (Flutter-застосунок)	Візуалізація інтерфейсу, передача запитів до серверу
3	Сервер аутентифікації	<<device>> Firebase Auth	Авторизація та реєстрація	Firebase Authentication	Обробка логіну, паролю, токенів доступу
4	Сервер бази даних	<<device>> Firebase Firestore	Сховище структурованих даних	Firebase Cloud Firestore	Зберігання постів, коментарів, вподобань
5	Сервер зображень	<<device>> Firebase Storage	Сховище мультимедійного контенту	Firebase Storage	Завантаження, зберігання та відображення зображень
6	Мережева взаємодія	Протокол HTTP/HTTPS	Канал обміну даними між клієнтом і БД	–	Надсилання запитів до серверів, отримання відповідей у реальному часі
7	Адміністратор системи	Робоча станція розробника	Технічне обслуговування, супровід	Firebase Console, Android Studio, VS Code	Налаштування сервісів, моніторинг, аналітика, деплой оновлень

Представлена таблиця узагальнює мінімальні та функціональні вимоги до основних апаратних і програмних компонентів системи PicFlow. Така структура дозволяє забезпечити узгоджену роботу всіх модулів платформи, ефективний

обмін даними, безпечний доступ до функцій і масштабованість при зростанні кількості користувачів.

З технічної точки зору, окрім вимог, які ставляться до системи з боку користувачів та власника, важливо також врахувати потреби, що виникають у процесі розробки, підтримки й розгортання застосунку. Вимоги до апаратного та програмного забезпечення з боку розробника визначають можливість ефективного виконання повного циклу життєдіяльності програмного продукту – від проєктування до оновлень та супроводу.

Для реалізації застосунку розробнику необхідно мати робочу станцію з достатнім рівнем продуктивності, що дозволяє одночасно працювати з середовищем розробки (Android Studio або Visual Studio Code), емуляторами мобільних пристроїв та консоллю Firebase. Рекомендується використовувати систему з процесором не нижче Intel Core i3 або аналогом ARM, оперативною пам’яттю від 8 ГБ, SSD-диском для швидкого доступу до ресурсів проєкту та стабільним підключенням до Інтернету. З програмного боку обов’язковою є наявність встановленого Flutter SDK з відповідною версією Dart [13], а також CLI-інструментів Firebase для налаштування, аутентифікації й автоматизації розгортання (табл.4.3).

Таблиця 4.3

Програмні вимоги

Назва інструменту	Версія (рекомендована або мінімальна)	Додаткові зауваження
1	2	3
Операційна система	Windows 10/11, macOS 10.15+.	Рекомендується остання стабільна версія для кращої сумісності та безпеки.
Flutter SDK	Остання стабільна версія	Регулярне оновлення до останньої стабільної версії для отримання нових функцій та виправлень. Інструкції з налаштування середовища для розробника подано в [13].
Dart SDK	Входить до складу Flutter SDK	Версія Dart залежить від версії Flutter.

Таблиця 4.3 (продовження)

1	2	3
Visual Studio Code (VS Code)	Остання стабільна версія	Потрібно встановити розширення Flutter та Dart для кращого досвіду розробки.
Android Studio	Остання стабільна версія	Потрібен для емулятора Android та Android SDK, навіть якщо основна розробка ведеться у VS Code.
Xcode	Остання стабільна версія	Потрібен для розробки та емуляції iOS-додатків (доступний лише на macOS).
Firebase CLI	Остання стабільна версія	Для взаємодії з сервісами Firebase з командного рядка.

Примітка: Складено на основі документації Flutter SDK [13] та рекомендацій Google Firebase [14].

Крім того, важливо мати налаштоване середовище для контролю версій (наприклад, Git), що дозволяє командній роботі над проєктом та безпечному зберіганню змін у коді. У разі використання CI/CD-підходів – необхідно забезпечити підтримку автоматичних збірок і тестування, що значно підвищує надійність і гнучкість розробницького процесу. Усі ці вимоги створюють технічне підґрунтя для продуктивної і стабільної роботи розробника в середовищі повного циклу життєвого процесу системи.

У контексті розробки соціальної медіа-платформи важливим є врахування потреб та обмежень кінцевих користувачів – саме з їхньої точки зору формується очікування щодо зручності, швидкодії та доступності застосунку. Технічні вимоги до клієнтських пристроїв мають забезпечити безперебійну роботу основних функцій програми без надмірного навантаження на ресурси пристрою. Це особливо актуально для мобільних платформ, які є основним середовищем експлуатації таких застосунків.

З боку користувача необхідно лише мати сучасний смартфон з актуальною версією операційної системи, достатнім обсягом оперативної пам’яті та вільного простору у внутрішньому сховищі для встановлення додатку та зберігання тимчасових даних. Додаткові вимоги, зокрема дозвіл на використання камери,

галереї чи мережевого з’єднання, залежать від активності користувача у межах функціоналу – наприклад, створення публікацій із фото, перегляд стрічки чи коментування.

Узагальнені вимоги до клієнтських пристроїв з точки зору кінцевого користувача, наведено у таблиці 4.4.

Таблиця 4.4

Вимоги до клієнтських пристроїв з боку користувача

№	Категорія	Параметри та умови використання
1	Операційна система (Android)	Android 8.0 (Oreo) або новіша
2	Операційна система (iOS)	iOS 13 або новіша; підтримка з iPhone 8 / SE (2nd generation)
3	Обсяг оперативної пам’яті	Не менше 2 ГБ
4	Вільне місце у пам’яті	Мінімум 200 МБ для інсталяції, кешування даних і медіафайлів
5	Мережеві вимоги	Стабільне інтернет-з’єднання (Wi-Fi або мобільні дані 4G+)
6	Доступ до сервісів	Google-сервіси (для Android), App Store (для iOS)
7	Доступ до пристроїв	Дозвіл на використання камери та галереї для завантаження контенту

Наведені вимоги є узгодженими з сучасними технічними можливостями більшості мобільних пристроїв, що забезпечує широке охоплення аудиторії без необхідності в дорогому обладнанні. Такий підхід сприяє популяризації додатку серед користувачів і створює умови для його стабільного функціонування навіть на смартфонах середнього класу.

4.3 Склад інсталяційного пакету

Процес інсталяції програмного забезпечення передбачає підготовку повного набору компонентів, необхідних для запуску, компіляції та налаштування системи у середовищі розробника або на кінцевому пристрої. У випадку розробки соціальної медіа-платформи на базі Flutter та Firebase, інсталяційний пакет формується у вигляді структури проєкту, яка містить

вихідні файли програми, конфігураційні ресурси, а також файл із визначенням зовнішніх бібліотек та залежностей, що автоматично підключаються під час збірки.

На рис. 4.10 представлено вміст каталогу основного проєкту, сформованого за структурною моделлю фреймворку Flutter. Тут зберігаються вихідні файли інтерфейсу, логіки управління станами, шаблони для екранів, тестові модулі та конфігураційні каталоги (lib, assets, test, .idea, .vscode тощо). Саме ця структура забезпечує повний цикл збирання додатку – від завантаження ресурсів до створення інсталяційного APK-файлу.

Наступним обов'язковим елементом інсталяційного пакету є файл pubspec.yaml, що виконує роль конфігураційного менеджера залежностей у проєктах на Flutter. Він містить інформацію про всі підключені бібліотеки, включаючи офіційні пакети Flutter SDK (наприклад, flutter_bloc, cloud_firestore, firebase_auth) та зовнішні плагіни для роботи з інтерфейсом, маршрутизацією або хмарним сховищем. На рис. 4.11 показано фрагмент вмісту цього файлу, де зображено активні залежності, що підключаються при збиранні додатку.

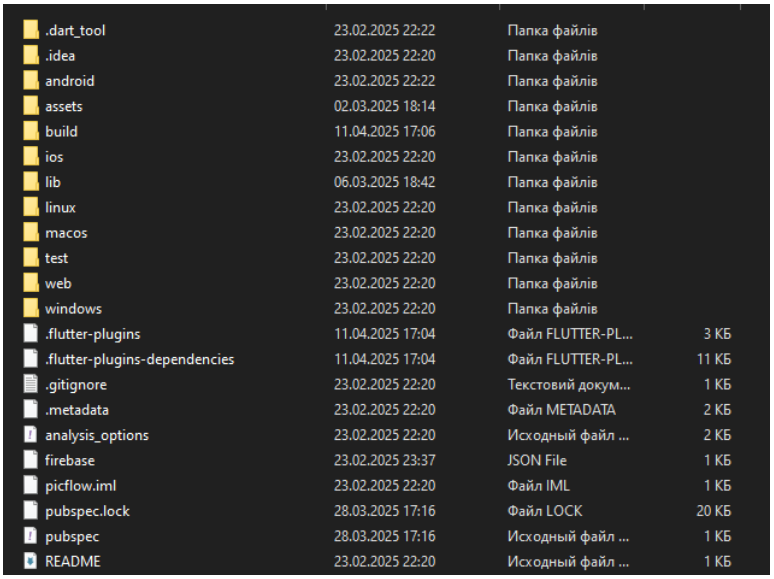


Рис.4.10 Структура директорії проєкту

```yaml
dependencies:
  flutter:
    sdk: flutter


  cupertino_icons: ^1.0.8
  firebase_core: ^3.12.0
  firebase_auth: ^5.5.0
  flutter_bloc: ^9.0.0
  cloud_firestore: ^5.6.5
  firebase_storage: ^12.4.4
  file_picker: ^9.2.2
  cached_network_image: ^3.4.1
  permission_handler: ^11.4.0
```

Рис.4.11 Вміст файлу `pubspec.yaml` із зазначеними залежностями

Для кращої систематизації основних компонентів інсталяційного пакету та зручності його використання подамо структуру у вигляді узагальненої табл. 4.5.

Таблиця 4.5

Основні компоненти інсталяційного пакету мобільного застосунку

№	Компонент	Призначення та функціональність
1	Каталог проєкту (lib, assets, test)	Містить вихідні файли інтерфейсу, логіки, ресурсів і тестування
2	pubspec.yaml	Декларація всіх бібліотек, плагінів і залежностей, необхідних для компіляції [19]
3	.gitignore, .metadata	Конфігураційні та службові файли для керування версіями та середовищем
4	android, ios	Платформенно-залежні модулі для генерації відповідних збірок (APK/IPA)
5	README.md, LICENSE	Документація, що описує процес розгортання, призначення проєкту та правовий статус
6	APK-файл (app-release.apk)	Результат збірки застосунку, що встановлюється на Android-пристрої

Склад інсталяційного пакету в рамках розробки мобільного застосунку є логічно впорядкованим і відповідає сучасним стандартам розгортання програм на платформі Flutter. Залучення системи керування залежностями (pubspec.yaml) дозволяє динамічно підключати необхідні бібліотеки без ручного копіювання файлів, що значно спрощує підтримку та оновлення проєкту і відповідає практикам CI/CD та DevOps для мобільних застосунків [15]. Таким чином, представлена структура забезпечує повну відтворюваність проєкту, дозволяє безперешкодно налаштувати середовище для повторного використання або внесення змін, а також є придатною як для локального розгортання, так і для CI/CD-середовищ.

ВИСНОВКИ

У результаті виконання бакалаврського проєкту було розроблено кросплатформний мобільний застосунок соціальної медіа-платформи, який забезпечує можливість публікації контенту, перегляду стрічки новин, взаємодії користувачів через лайки, коментарі та підписки, а також управління особистим профілем. Реалізовані функції відповідають базовим очікуванням сучасних користувачів соціальних мереж і формують технічну основу для подальшого розвитку застосунку.

У процесі розробки застосовано сучасні технології: Flutter як інструмент реалізації інтерфейсу користувача та логіки взаємодії, а також Firebase як хмарне середовище для зберігання даних, автентифікації та роботи з мультимедіа. Така конфігурація дозволила побудувати гнучку архітектуру, що поєднує масштабованість, високу продуктивність та зручність використання.

Особлива увага приділялася ергономіці інтерфейсу та структурі взаємодії з користувачем. Простий і інтуїтивно зрозумілий дизайн забезпечує легкість навігації та сприяє активному залученню користувачів до створення та обміну контентом. Це підсилює відчуття спільноти в межах платформи та відкриває перспективи для формування стабільного користувацького середовища.

Розробка додатку стала практичним підтвердженням моїх навичок у застосуванні принципів об’єктно-орієнтованого програмування, роботи з хмарними сервісами, побудови модульної архітектури та реалізації систем на основі шаблонів проєктування. Було успішно проведено проєктування інформаційної бази, створення логічної моделі даних, реалізацію алгоритмів для роботи з контентом, моделювання структури системи засобами UML і виконання тестування функціональних сценаріїв.

Створена система має потенціал для масштабування, розширення функціональності та інтеграції додаткових сервісів — від персоналізованих рекомендацій до систем аналітики активності. Таким чином, отриманий

результат може бути використаний як база для розвитку повноцінної соціальної платформи або адаптації її до конкретних тематичних спільнот.

Загалом, проєкт довів можливість реалізації повнофункціонального мобільного соціального застосунку із сучасною архітектурою, ефективною роботою з даними та зручним інтерфейсом. Це стало важливим практичним кроком у моєму професійному становленні як фахівця з розробки інформаційних систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. UML: що це таке та як працює [Електронний ресурс]. – Режим доступу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/
2. Що таке UML-діаграми? [Електронний ресурс]. – Режим доступу: https://evergreens.com.ua/ua/articles/uml-diagrams.html
3. Foxminded. Що таке СУБД і для чого вони потрібні [Електронний ресурс]. – Режим доступу: https://foxminded.ua/systema-upravlinnia-bazamy-danykh/
4. How to Build a Social Media Platform from Scratch? [Електронний ресурс]. – Режим доступу: https://www.apptunix.com/blog/how-to-build-a-social-media-platform-from-scratch/
5. Instagram. Офіційний сайт [Електронний ресурс]. – Режим доступу: https://www.instagram.com
6. TikTok. Офіційна сторінка [Електронний ресурс]. – Режим доступу: https://www.tiktok.com
7. Threads by Instagram. Офіційна сторінка [Електронний ресурс]. – Режим доступу: https://www.threads.net
8. Meta Platforms Inc. – Facebook Overview [Електронний ресурс]. – Режим доступу: https://www.facebook.com
9. BeReal. Офіційна інформація [Електронний ресурс]. – Режим доступу: https://bereal.com/en
10. DAN.IT. Види функціонального та нефункціонального тестування [Електронний ресурс]. – Режим доступу: https://dan-it.com.ua/uk/blog/vidy-funkcionalnogo-i-nefunkcionalnogo-testirovanija/
11. IEEE 829–2008. IEEE Standard for Software and System Test Documentation [Електронний ресурс]. – Режим доступу: https://standards.ieee.org
12. Гузенко А.І., Мельник С.А. Основи тестування програмного забезпечення: навч. посіб. – Київ: КНУ, 2021. – 144 с.
13. Sharma R. Flutter for Beginners. – Apress, 2021. – 290 p.

14. Firebase Architecture and Implementation Guide / Google Cloud [Електронний ресурс]. – Режим доступу: https://firebase.google.com/docs/architecture
15. Алексєєв Є.О. CI/CD для мобільної розробки [Електронний ресурс]. – Режим доступу: https://dou.ua/lenta/articles/cicd-mobile/
16. Firebase Documentation [Електронний ресурс]. – Режим доступу: https://firebase.google.com/docs?hl=ua
17. Flutter Documentation [Електронний ресурс]. – Режим доступу: https://docs.flutter.dev/
18. Dart Documentation [Електронний ресурс]. – Режим доступу: https://dart.dev/guides
19. pub.dev. Репозиторій пакетів [Електронний ресурс]. – Режим доступу: https://pub.dev/
20. GeeksforGeeks. What is Software Testing? [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/software-testing-basics/

ДОДАТОК А

Сторінок – 2

Програмний код всіх сутностей додатку

Сутність ProfileUser

```dart
import 'package:picflow/features/auth/domain/entities/app_user.dart';

class ProfileUser extends AppUser {
  final String bio;
  final String profileImageUrl;
  final List<String> followers;
  final List<String> following;

  ProfileUser({
    required super.uid,
    required super.email,
    required super.name,
    required this.bio,
    required this.profileImageUrl,
    required this.followers,
    required this.following,
  });

  //метод оновлення профілю
  ProfileUser copyWith({
    String? newBio,
    String? newProfileImageUrl,
    List<String>? newFollowers,
    List<String>? newFollowing,
  }) {
    return ProfileUser(
      uid: uid,
      email: email,
      name: name,
      bio: newBio ?? bio,
      profileImageUrl: newProfileImageUrl ?? profileImageUrl,
      followers: newFollowers ?? followers,
      following: newFollowing ?? following,
    );
  }

  // перетворення в JSon
  Map<String, dynamic> toJson() {
    return {
      'uid': uid,
      'email': email,
      'name': name,
      'bio': bio,
      'profileImageUrl': profileImageUrl,
      'followers': followers,
      'following': following,
    };
  }

  factory ProfileUser.fromJson(Map<String, dynamic> json) {
    print("📢 Конвертація з JSON: $json");
    return ProfileUser(
      uid: json['uid'],
      email: json['email'],
      name: json['name'],
      bio: json['bio'] ?? '',
      profileImageUrl: json['profileImageUrl'] ?? '',
```

```
      followers: List<String>.from(json['followers'] ?? []),
      following: List<String>.from(json['following'] ?? []),
    );
  }
}
```

Сутність Comment

```
import 'package:cloud_firestore/cloud_firestore.dart';

class Comment {
  final String id;
  final String postId;
  final String userId;
  final String userName;
  final String text;
  final DateTime timestamp;

  Comment({
    required this.id,
    required this.postId,
    required this.userId,
    required this.userName,
    required this.text,
    required this.timestamp,
  });

  Map<String, dynamic> toJson() {
    return {
      'id': id,
      'postId': postId,
      'userId': userId,
      'userName': userName,
      'text': text,
      'timestamp': Timestamp.fromDate(timestamp),
    };
  }

  factory Comment.fromJson(Map<String, dynamic> json) {
    return Comment(
      id: json['id'],
      postId: json['postId'],
      userId: json['userId'],
      userName: json['userName'],
      text: json['text'],
      timestamp: (json['timestamp'] as Timestamp).toDate(),
    );
  }
}
```