

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Комп'ютерних наук і технологій
(повне найменування факультету)

Комп'ютерні системи та мережі
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалаврський
(ступінь вищої освіти)

на тему: РОЗРОБКА КОМП'ЮТЕРНОЇ СИСТЕМИ ДЛЯ ПОШУКУ
ДРУЗІВ ЗА ІНТЕРЕСАМИ ДЛЯ ANDROID

Виконав(ла): студент(ка) 4 курсу,
групи КНТ-511

Спеціальності
123 Комп'ютерна інженерія
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Комп'ютерна інженерія
(назва освітньої програми (спеціалізації))

ПОНОМАРЕНКО М.В.
(ПРИЗВИЩЕ та ініціали)

Керівник СКРУПСЬКИЙ С.Ю.
(ПРИЗВИЩЕ та ініціали)

Рецензент ЄФИМЕНКО М.В.
(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет Комп'ютерних наук і технологій
Кафедра комп'ютерних систем та мереж
Ступінь вищої освіти бакалаврський
Спеціальність 123 Комп'ютерна інженерія
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерна інженерія
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри КУДЕРМЕТОВ Р.К.

«14» квітня 2025 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ПОНОМАРЕНКО Миколи Володимировича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Комп'ютерна система для пошуку друзів за інтересами для платформи Android

керівник проєкту (роботи) к.т.н., доцент, СКРУПСЬКИЙ С.Ю.

(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від «08» квітня 2025 року № 151

2. Строк подання студентом проєкту (роботи) 01.06.2025 р.

3. Вихідні дані до проєкту (роботи) опис предметної області, технології розробки клієнтської та серверної частини

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1) Визначення технологій розробки;

2) Розробка вимог до програмного забезпечення;

3) Моделювання програмного забезпечення;

4) Розробка системи;

5) Випробування комп'ютерної системи.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4	СКРУПСЬКИЙ С.Ю.		
Нормоконтроль	ПОЛЬСЬКА О.В.		

7. Дата видачі завдання «14» квітня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	П римітка
1	Аналіз предметної області	до 18.04.2025	
2	Визначення та аналіз вимог до програмного забезпечення	до 22.04.2025	
3	Розробка специфікації вимог SRS	до 24.04.2025	
4	Розробка діаграми варіантів використання	до 28.04.2025	
5	Розробка діаграми послідовності	до 01.05.2025	
6	Розробка описів прецедентів	до 05.05.2025	
7	Розробка системи	до 12.05.2025	
8	Розробка користувацького інтерфейсу	до 20.05.2025	
9	Випробування комп'ютерної системи	до 22.05.2025	
10	Оформлення пояснювальної записки	до 25.05.2025	
11	Проходження нормоконтролю	до 01.06.2025	
12	Перевірка на наявність академічного плагіату	до 03.06.2025	
13	Проходження рецензування	до 10.06.2025	

Студент(ка)

(підпис)

Микола ПОНОМАРЕНКО

(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис)

Степан СКРУПСЬКИЙ

(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
103 с., 7 табл., 17 рис., 1 дод., 19 джерел.

КОМП'ЮТЕРНА СИСТЕМА ДЛЯ ANDROID, ПОШУК ДРУЗІВ, KOTLIN,
JETPACK COMPOSE, NODE.JS, NEST.JS, POSTGRESQL

Об'єкт розробки – мобільний додаток для пошуку друзів за інтересами на платформі Android.

Мета роботи – створення додатку для автоматизації пошуку соціальних зв'язків за інтересами з зручним управлінням профілями та дружніми зв'язками.

У процесі розробки проаналізовано технічне завдання, спроектовано систему, розроблено базу даних, алгоритми роботи та макет інтерфейсу. Для бази даних обрано SQLite з використанням Android Studio. Клієнтська частина створена на Kotlin з Jetpack Compose, серверна – на JavaScript з Node.js та Nest.js.

Додаток дозволяє створювати профілі, налаштовувати фільтри, надсилати запити на дружбу та переглядати стрічку. База даних містить таблиці UserCore, UserProfile, UserFilters, FriendshipRequest, Friend. Інтерфейс інтуїтивний, включає сторінки профілю, фільтрів та друзів.

Тестування підтвердило стабільну роботу модулів автентифікації, профілю, фільтрів та управління друзями, відповідаючи вимогам.

ЗМІСТ

Вступ.....	6
1 Визначення технологій розробки.....	7
1.1 Вибір архітектури системи.....	7
1.2 Вибір технологій для фронтенду та бекенду.....	11
1.3 Вибір бази даних	23
1.4 Висновки щодо вибору СУБД.....	30
2 Розробка вимог до програмного забезпечення.....	31
2.1 Визначення та аналіз вимог до програмного забезпечення.....	31
2.2 Розробка специфікації вимог SRS	31
2.3 Діаграми варіантів використання	36
2.4 Діаграми послідовності	39
2.5 Розгорнуті описи прецедентів.....	42
3 Розробка системи.....	47
3.1 Розробка клієнтської частини системи.....	47
3.2 Розробка серверної частини системи	62
4 Випробування комп'ютерної системи.....	77
4.1 Випробування користувацького інтерфейсу	77
4.2 Функціональне випробування.....	83
Висновки	86
Перелік джерел посилання	88
Додаток А	90

ВСТУП

Стрімкий розвиток інформаційних технологій у сучасному світі відкриває нові можливості для вдосконалення різних аспектів повсякденного життя, зокрема у сфері соціальної взаємодії. Зі зростанням урбанізації, глобалізації та цифровізації виникає актуальна потреба в інструментах, які б допомагали людям налагоджувати дружні стосунки, знаходити однодумців і розширювати коло спілкування.

Платформи для пошуку друзів за інтересами стають важливим засобом для формування нових соціальних зв'язків, підтримки емоційної стабільності та розвитку особистих навичок спілкування в цифрову епоху. В умовах зростання популярності мобільних пристроїв особливого значення набуває створення зручних та доступних мобільних додатків, орієнтованих на різноманітну аудиторію користувачів.

Дипломна робота присвячена розробці мобільного застосунку для пошуку друзів за спільними інтересами. Основною метою проєкту є спрощення процесу встановлення нових контактів та створення платформи, що об'єднуватиме людей із подібними захопленнями. Особливий акцент у розробці зроблено на інтуїтивність інтерфейсу, високу швидкодію та актуальність наданої інформації, що забезпечить комфортне користування додатком.

Проєкт включає аналіз існуючих аналогів, формулювання вимог до майбутньої системи, побудову її архітектури, розробку інтерфейсу користувача та тестування створеного прототипу. Завдання роботи полягає у розробці ефективного інструменту, який відповідатиме сучасним вимогам мобільної розробки, враховуватиме особливості взаємодії користувачів та забезпечуватиме позитивний досвід використання.

1 ВИЗНАЧЕННЯ ТЕХНОЛОГІЙ РОЗРОБКИ

1.1 Вибір архітектури системи

Архітектура програмного забезпечення формує фундаментальні рішення, що визначають спосіб задоволення технічних та операційних потреб проєкту. Завдяки правильному вибору архітектури можна досягти оптимізації основних характеристик системи - таких як безпека, продуктивність і керованість, що безпосередньо впливають на рівень обслуговування, загальну ефективність та успіх кінцевого продукту на ринку. Грамотно спроектована архітектура створює стабільну базу для подальшого масштабування і дозволяє легко адаптувати систему до змінних вимог користувачів та бізнес-середовища [1].

Прийняття рішення щодо архітектурного підходу має критичне значення на всіх етапах життєвого циклу розробки програмних систем. Від цього залежить, наскільки легко компоненти зможуть взаємодіяти між собою, а також наскільки ефективно система відповідатиме технічним стандартам і очікуванням кінцевих користувачів. Вибір відповідної архітектури сприяє створенню рішень, що легко масштабуються, обслуговуються та модернізуються, водночас дозволяючи гнучко реагувати на складні виклики, які виникають у динамічному бізнес-середовищі [2].

На сьогоднішній день існує велика різноманітність архітектурних стилів, кожен з яких має свої переваги і обмеження, що залежать від конкретних вимог і особливостей розроблюваного проєкту. Найпоширенішими типами архітектур є: монолітна, мікросервісна, клієнт-серверна, Serverless-архітектура.

1.1.1 Монолітна архітектура

Монолітна архітектура програмного забезпечення передбачає об'єднання всіх основних компонентів системи - клієнтського інтерфейсу, бізнес-логіки, модуля обробки даних та серверної частини - в єдину цілісну програму. Усі елементи працюють у межах одного процесу, що забезпечує тісну інтеграцію і високу ефективність їх взаємодії.

Однією з головних переваг монолітної архітектури є спрощення процесів розробки, тестування та розгортання системи. Всі компоненти зберігаються в одному кодовому репозиторії, що полегшує виявлення помилок, інтеграційне тестування та спрощує управління проектом на ранніх стадіях. Крім того, цей підхід дозволяє знизити витрати на інфраструктуру, оскільки немає потреби у складних розподілених середовищах [3].

У більшості випадків у межах монолітної архітектури застосовується одна мова програмування. Це пояснюється тісною інтеграцією усіх частин системи, де бізнес-логіка, обробка запитів і презентаційний рівень тісно взаємопов'язані і працюють в рамках одного середовища виконання.

Проте зі зростанням обсягу програми та ускладненням її функціональності монолітна архітектура може стати перешкодою. Масштабування окремої функції часто вимагає масштабування всієї системи загалом, що призводить до неефективного використання ресурсів. Адаптація до нових технологій або змін у структурі додатку може вимагати значних переробок коду. Крім того, критична помилка в одному модулі може паралізувати роботу всього застосунку, що збільшує ризики простою системи.

Монолітна архітектура є доцільним рішенням у тих випадках, коли важливими є швидкість розробки, простота розгортання і відсутність необхідності у високому рівні масштабованості. Такий підхід оптимально підходить для невеликих або середніх проектів, де вся функціональність може бути створена та протестована централізовано без значних витрат на інфраструктуру і міжкомпонентну взаємодію.

1.1.2 Мікросервісна архітектура

Мікросервісна архітектура сформувалася як відповідь на обмеження класичних монолітних рішень і стала одним із найпопулярніших підходів для побудови масштабованих та адаптивних систем. Основна її ідея полягає у розбитті застосунку на набір незалежних сервісів, кожен із яких виконує окрему бізнес-функцію і може розгортатися, масштабуватися та оновлюватися автономно.

Однією з головних переваг мікросервісного підходу є висока гнучкість у розробці та експлуатації системи. Завдяки ізоляції сервісів можна окремо масштабувати тільки ті компоненти, які зазнають підвищеного навантаження, що дає змогу оптимально використовувати ресурси та зменшувати витрати на інфраструктуру [4]. Крім того, незалежність мікросервісів дозволяє паралельно працювати над різними частинами системи кільком командам розробників, що пришвидшує вихід нових функціональностей на ринок.

Втім, мікросервісна архітектура також має низку викликів. Складність системи суттєво зростає через необхідність налагодження міжсервісної комунікації, синхронізації даних і забезпечення консистентності. Кожен сервіс може використовувати свій стек технологій, що ускладнює супровід та вимагає ретельної координації між командами. Також значно зростає навантаження на мережу, оскільки сервіси взаємодіють через API або брокери повідомлень, що потребує належної оптимізації взаємодії.

Оскільки для ефективної роботи мікросервісної архітектури критично важливі процеси автоматичного розгортання, моніторингу, логування та масштабування сервісів, розробник повинен володіти практичними навичками в області DevOps та CI/CD практик.

Таким чином, вибір мікросервісної архітектури виправданий для систем із високими вимогами до масштабованості, швидкості виходу продукту на ринок і гнучкості оновлень. Проте для її впровадження необхідно мати достатньо ресурсів на побудову відповідної інфраструктури та забезпечення ефективної взаємодії між сервісами.

1.1.3 Клієнт-серверна архітектура

Клієнт-серверна архітектура - це одна з найпоширеніших моделей побудови інформаційних систем, яка передбачає розподіл функціональності між двома основними компонентами: клієнтом, який ініціює запити, та сервером, який обробляє ці запити і надсилає відповідь. Такий підхід забезпечує чітку структуру взаємодії, централізоване управління даними та ізоляцію логіки обробки від інтерфейсної частини.

У даній архітектурі клієнт зазвичай є програмним забезпеченням, що виконується на пристрої користувача (наприклад, мобільний додаток або веббраузер), а сервер - це потужніша система, яка обробляє запити, зберігає дані, управляє доступом, авторизацією та виконує бізнес-логіку. Обмін між клієнтом і сервером зазвичай відбувається через мережу (наприклад, інтернет) за допомогою стандартних протоколів - найчастіше HTTP(S).

Переваги клієнт-серверної архітектури:

- централізоване управління даними, що спрощує контроль доступу, оновлення інформації та резервне копіювання;
- чіткий розподіл відповідальності між клієнтом і сервером, що сприяє масштабуванню і підтримці системи;
- можливість масштабування сервера - серверна частина може обробляти велику кількість клієнтів через балансування навантаження;

Однак дана архітектура також має певні обмеження:

- залежність від постійного мережевого з'єднання - у разі втрати доступу до сервера клієнтський функціонал може бути обмежений;
- єдина точка відмови - збій на сервері призводить до недоступності всієї системи;
- навантаження на сервер при великій кількості одночасних запитів може призвести до затримок або зниження продуктивності без належного масштабування.

Клієнт-серверна модель є базовою архітектурою у розробці більшості сучасних інформаційних систем - від вебзастосунків і мобільних платформ до корпоративних сервісів. Саме вона часто виступає фундаментом для подальшого переходу до більш складних підходів, таких як мікросервіси або розподілені обчислення, коли зростають вимоги до масштабованості та відмовостійкості.

1.1.4 Serverless архітектура

Serverless архітектура змінює традиційний підхід до розробки застосунків, дозволяючи розробникам фокусуватися виключно на реалізації бізнес-логіки без необхідності самотійно управляти інфраструктурою. У цьому випадку

відповідальність за розгортання, масштабування та адміністрування серверів бере на себе хмарний провайдер, що значно спрощує процес розробки та скорочує час виведення продукту на ринок [5].

Однією з ключових переваг цього підходу є модель оплати за фактичне використання ресурсів. Завдяки цьому розробники та бізнес можуть мінімізувати витрати, сплачуючи лише за обчислювальні потужності, що реально використовуються під час виконання функцій. Автоматичне масштабування, вбудоване в сервіси Serverless, дозволяє системі динамічно адаптуватися до змін навантаження без необхідності ручного втручання чи попереднього планування масштабування [6].

Проте, незважаючи на низку очевидних переваг, Serverless має свої обмеження. Розробники втрачають контроль над низькорівневими процесами, що ускладнює налагодження, тестування і виявлення проблем на продакшн-середовищах. Крім того, довіра до стороннього постачальника інфраструктури створює додаткові ризики для безпеки та конфіденційності даних.

Serverless архітектура найкраще підходить для застосунків, орієнтованих на обробку подій у реальному часі, сервісів із нерівномірним трафіком або для побудови окремих мікросервісних компонентів. Проте перед впровадженням цього підходу необхідно провести ретельний аналіз вимог до системи, оцінити характер навантаження та допустимі затримки у виконанні запитів.

1.2 Вибір технологій для фронтенду та бекенду

Сучасний розвиток мобільних технологій пропонує розробникам широкий спектр інструментів для створення функціональних і зручних мобільних додатків. При розробці Android-застосунків особлива увага приділяється вибору технологій для реалізації інтерфейсу користувача (фронтенду) та серверної логіки (бекенду).

Технології для фронтенду мобільних додатків орієнтовані на створення інтуїтивно зрозумілих інтерфейсів та забезпечення комфортної взаємодії користувача із системою. Традиційно для Android-розробки використовували комбінацію мови Java та розмітки інтерфейсу у форматі XML. Однак із розвитком технологій все більше проєктів переходять на Kotlin у поєднанні з декларативною системою побудови інтерфейсів Jetpack Compose, яка значно спрощує та прискорює процес розробки.

Крім нативних рішень, існують також кросплатформені технології, такі як React Native, які дозволяють створювати мобільні додатки для Android та iOS із використанням єдиного коду на JavaScript. Це може зменшити витрати на розробку, проте накладає певні обмеження на доступ до нативних можливостей платформи.

Щодо бекенд-розробки, для побудови серверної частини застосунків використовуються різні сучасні технології. Одними з популярних рішень є Node.js з використанням фреймворків, таких як Nest.js, що дозволяють створювати масштабовані та структуровані серверні API. Крім Node.js, для бекенду часто застосовуються також такі технології як Django (Python) та Laravel (PHP), кожна з яких має свої сильні сторони в залежності від вимог до проєкту.

Слід зауважити, що в сучасній мобільній розробці практично не використовується написання серверної логіки або інтерфейсів "з нуля". Перевага надається використанню перевірених фреймворків і бібліотек, які значно пришвидшують розробку, забезпечують масштабованість і сприяють підвищенню якості коду.

Таким чином, правильний вибір технологій для фронтенду та бекенду є визначальним фактором для створення якісного, надійного та масштабованого мобільного застосунку.

1.2.1 Kotlin

Kotlin - це сучасна статично типізована мова програмування, розроблена компанією JetBrains для розробки багатофункціональних програмних продуктів, з особливим акцентом на мобільні додатки під Android. У 2017 році компанія Google

офіційно визнала Kotlin як рекомендовану мову для створення Android-застосунків, що сприяло стрімкому зростанню її популярності серед розробників [7].

Однією з головних особливостей Kotlin є його лаконічний і водночас виразний синтаксис. Завдяки скороченню кількості шаблонного коду (boilerplate), Kotlin дозволяє зменшити розмір програмної бази без втрати читабельності, що сприяє підвищенню продуктивності розробників. Мова підтримує об'єктно-орієнтоване та функціональне програмування, що забезпечує високу гнучкість у побудові архітектури застосунків.

Kotlin також інтегрується з існуючими Java-проектами без необхідності повної міграції. Код на Kotlin може співіснувати з Java у межах одного проекту, що дозволяє поетапно переводити застосунок на нову мову, мінімізуючи ризики і витрати часу.

Мова підтримує важливі сучасні концепції, такі як:

- розширення функцій (extension functions), які дозволяють додавати нові можливості до існуючих класів без зміни їхнього вихідного коду;
- корутини для асинхронного програмування, що значно спрощує роботу з багатопотоковістю;
- вбудовану підтримку властивостей, що скорочує кількість коду для геттерів та сетерів;
- оператори розширення і "data class" для спрощеної роботи з об'єктами даних.

Одним із найбільш важливих напрямків розвитку Kotlin є Kotlin Multiplatform, що дозволяє розробляти спільний код для Android, iOS, вебу та бекенду, проте для Android-проектів Kotlin використовується передусім у комбінації з Jetpack Compose для побудови ефективного і сучасного користувацького інтерфейсу.

Завдяки поєднанню безпечності, читабельності та потужних мовних конструкцій Kotlin сьогодні вважається однією з найефективніших мов для створення нативних мобільних застосунків. Він дає змогу розробникам писати надійний, масштабований код із мінімальними витратами часу на рутинні задачі,

що особливо важливо при розробці мобільних продуктів у динамічному середовищі.

1.2.2 Jetpack Compose

Jetpack Compose - це сучасний декларативний інструментарій від Google для створення користувацьких інтерфейсів (UI) в Android-додатках, який кардинально змінює підхід до розробки UI порівняно з традиційними View-системами. Представлений у 2019 році та офіційно рекомендований Google для Android-розробки з 2021 року, Jetpack Compose швидко здобув популярність завдяки своїй простоті, гнучкості та інтеграції з мовою Kotlin [8]. Цей інструментарій дозволяє розробникам створювати інтуїтивно зрозумілі, ефективні та візуально привабливі інтерфейси з меншою кількістю коду.

Однією з ключових особливостей Jetpack Compose є його декларативний синтаксис, який дозволяє описувати UI як функцію стану (state). Замість імперативного маніпулювання компонентами, як у View-системі, розробники визначають, як інтерфейс має виглядати залежно від даних, а Compose автоматично оновлює UI при зміні стану. Це значно спрощує керування динамічними інтерфейсами та зменшує ймовірність помилок, пов'язаних із ручним оновленням компонентів.

Jetpack Compose тісно інтегрований із Kotlin, використовуючи його потужні можливості, такі як корутини для асинхронної роботи та лаконічний синтаксис для визначення компонентів. Наприклад, функції з анотацією `@Composable` є основою для створення UI-елементів, що дозволяє писати читабельний і модульний код. Завдяки цьому інструментарію розробники можуть створювати складні інтерфейси, використовуючи лише кілька рядків коду, що підвищує продуктивність і полегшує підтримку проєктів.

Безпека та надійність також є важливими аспектами Jetpack Compose. Інструментарій використовує сильні сторони Kotlin, такі як підтримка null-безпеки, що допомагає уникати поширених помилок, наприклад, доступу до null-об'єктів під час роботи з UI-компонентами. Крім того, Compose пропонує вбудовані механізми для обробки стану (State і ViewModel), що забезпечують передбачувану поведінку

інтерфейсу навіть у складних сценаріях, таких як зміна конфігурації пристрою (наприклад, поворот екрана).

Серед ключових можливостей Jetpack Compose варто виділити:

- модульність і повторне використання завдяки Composables, які легко комбінуються та перевикористовуються, що спрощує створення консистентних інтерфейсів;

- вбудована підтримка анімацій через Animation API дозволяє створювати плавні переходи та інтерактивні ефекти без складної логіки;

- інструменти для попереднього перегляду, такі як анотації `@Preview`, дозволяють переглядати UI-компоненти в Android Studio без запуску програми, що прискорює ітеративну розробку.

Одним із перспективних напрямків розвитку Jetpack Compose є його використання в рамках Kotlin Multiplatform. Завдяки цьому інструментарію розробники можуть створювати спільний UI-код для Android та інших платформ, таких як iOS або Desktop, що зменшує дублювання зусиль і сприяє економії ресурсів.

Jetpack Compose також оптимізовано працювати з сучасними архітектурними підходами, такими як MVVM або MVI, що робить його ідеальним вибором для створення масштабованих додатків. Інтеграція з Jetpack-бібліотеками, такими як Navigation, LiveData та Room, забезпечує безшовну взаємодію між UI та бізнес-логікою.

Завдяки поєднанню декларативного підходу, тісної інтеграції з Kotlin і потужного набору інструментів Jetpack Compose став стандартом для розробки сучасних Android-додатків. Він дозволяє створювати швидкі, надійні та візуально привабливі інтерфейси, мінімізуючи рутинну роботу та даючи розробникам більше часу для реалізації креативних ідей у динамічному світі мобільної розробки.

1.2.4 React Native

React Native - це потужний фреймворк із відкритим кодом, розроблений компанією Facebook (нині Meta) у 2015 році, який дозволяє створювати нативні мобільні додатки для iOS і Android за допомогою JavaScript і React. Завдяки

можливості використовувати єдину кодову базу для кількох платформ, React Native значно прискорює розробку кросплатформених додатків, зберігаючи при цьому продуктивність і користувацький досвід, близькі до нативних. Фреймворк набув популярності серед розробників завдяки своїй гнучкості, великій екосистемі та підтримці від спільноти [9].

Основною особливістю React Native є компонентно-орієнтований підхід, успадкований від React. Розробники створюють інтерфейси користувача (UI) за допомогою модульних компонентів, які описують, як виглядатиме додаток залежно від стану даних. Цей декларативний стиль дозволяє легко керувати складними інтерфейсами та забезпечує передбачувані оновлення UI при зміні даних. React Native трансліує ці компоненти в нативні елементи платформи, такі як UIView для iOS або View для Android, що забезпечує високу продуктивність.

React Native використовує JavaScript (або TypeScript) як основну мову програмування, що робить фреймворк доступним для веб-розробників, знайомих із React. Водночас фреймворк дозволяє інтегрувати нативні модулі, написані на Swift, Objective-C або Java, для доступу до специфічних функцій платформи, таких як камера чи геолокація. Це забезпечує баланс між кросплатформною уніфікацією та нативною функціональністю.

Однією з ключових переваг React Native є економія часу та ресурсів. Завдяки спільній кодовій базі розробники можуть створювати додатки для iOS і Android одночасно, зменшуючи дублювання зусиль. Функція «гарячого перезавантаження» (Hot Reload) дозволяє миттєво бачити зміни в коді без перекомпіляції додатка, що значно прискорює ітеративну розробку.

React Native також інтегрується з сучасними інструментами та бібліотеками. Наприклад, він підтримує Redux для керування станом, React Navigation для навігації та Axios для мережевих запитів. Фреймворк сумісний із популярними IDE, такими як Visual Studio Code, і має розвинену екосистему плагінів, що спрощують додавання функціональності, наприклад, push-повідомлень чи аналітики.

Серед ключових можливостей React Native варто виділити:

- кросплатформна розробка та єдина кодова база для iOS і Android зменшує витрати часу та ресурсів на створення й підтримку додатків;
- використання нативних компонентів забезпечує швидкість і плавність інтерфейсу, близьку до повністю нативних додатків;
- hot reload та live reload дозволяють розробникам швидко тестувати зміни в реальному часі;
- інтеграція з нативним кодом дає доступ до специфічних функцій платформи без обмежень.

React Native активно розвивається, отримуючи регулярні оновлення від Meta та спільноти. Останні версії фреймворку, такі як New Architecture (Fabric і Turbo Modules), покращують продуктивність і спрощують інтеграцію з нативними компонентами. Крім того, React Native підтримує розробку не лише мобільних додатків, але й додатків для настільних систем (через проєкти, такі як React Native Windows) і вебплатформ, що розширює його універсальність.

У контексті мобільної розробки React Native часто порівнюють із Flutter і Xamarin, але він вирізняється простотою входження для веб-розробників і великою кількістю готових бібліотек. Фреймворк використовується в багатьох популярних додатках, таких як Facebook, Instagram і Airbnb, що підтверджує його надійність і масштабованість.

Завдяки поєднанню кросплатформної ефективності, простоти використання та потужної екосистеми React Native є одним із найкращих виборів для створення сучасних мобільних додатків. Він дозволяє розробникам швидко створювати надійні, продуктивні та візуально привабливі додатки, мінімізуючи витрати на розробку та підтримку.

1.2.5 Node.js

Node.js - це програмна платформа з відкритим вихідним кодом, яка дозволяє запускати JavaScript-код поза межами браузера. Вона була створена у 2009 році Райаном Далем на основі рушія V8 від Google, що використовується у браузері Chrome. Завдяки цьому Node.js забезпечує високу продуктивність та швидке

виконання коду, що зробило його популярним інструментом для побудови серверних додатків [10].

Основною особливістю Node.js є його неблокуюча, асинхронна модель введення/виведення (non-blocking I/O), яка дозволяє обробляти велику кількість одночасних з'єднань без блокування потоку. Це робить платформу надзвичайно ефективною для систем, що працюють із великою кількістю клієнтів у режимі реального часу - наприклад, чат-додатків, стрімінгових сервісів або веб-серверів.

Node.js використовує єдиний потік обробки подій (event loop), що дозволяє досягти високої продуктивності при мінімальному споживанні ресурсів. Розробники мають змогу використовувати JavaScript не лише для створення клієнтської частини застосунку, а й для реалізації серверної логіки, що сприяє уніфікації стеку технологій і зниженню порогу входу.

Серед переваг Node.js також варто виділити:

- npm (Node Package Manager) - один із найбільших у світі менеджерів пакунків із десятками тисяч готових модулів, які суттєво прискорюють розробку;
- активну спільноту, яка підтримує розвиток нових бібліотек та фреймворків;
- гнучкість, що дозволяє створювати як монолітні, так і мікросервісні архітектури.

Node.js є платформою, яку часто використовують у поєднанні з популярними фреймворками, такими як Express.js або Nest.js, для побудови RESTful API, GraphQL-серверів або WebSocket-з'єднань. Завдяки своїй модульній природі, система легко розширюється та адаптується до потреб конкретного проєкту.

Проте, як і будь-яка технологія, Node.js має свої обмеження. Наприклад, для задач із великим обсягом обчислень або блокуючих операцій варто використовувати worker-потоки або зовнішні сервіси, оскільки тривалий процес у головному потоці може заблокувати обробку інших подій.

Незважаючи на ці нюанси, Node.js залишається одним із найефективніших інструментів для створення сучасних бекенд-систем, особливо у випадках, коли потрібна висока швидкодія, масштабованість і тісна інтеграція з фронтендом на

JavaScript. Його гнучкість і продуктивність роблять його одним з найкращих рішень для розробки серверної частини мобільних застосунків.

1.2.6 Nest.js

Nest.js - це прогресивний фреймворк для розробки серверної частини застосунків, створений на базі платформи Node.js з використанням мови TypeScript. Його головною метою є надання архітектурної структури, подібної до тієї, що використовується у таких фреймворках, як Angular, що дозволяє організовувати великий код у масштабовані та підтримувані модулі [11].

Однією з ключових особливостей Nest.js є використання модульної архітектури, що забезпечує логічне розділення відповідальностей між різними частинами додатку. Фреймворк базується на об'єктно-орієнтованих, функціональних та реактивних підходах до програмування. Він також активно використовує концепції з шаблонів SOLID, інверсії залежностей (Dependency Injection) та декораторів, що дозволяє створювати чистий, легко тестований код.

Nest.js будує свій HTTP-рівень поверх Express.js або Fastify - двох найпопулярніших HTTP-серверів для Node.js. Завдяки цьому розробники отримують гнучкість вибору між мінімалістичним або високопродуктивним сервером, зберігаючи при цьому єдину структуру програми. Можливість швидкої інтеграції з WebSocket, GraphQL, REST, мікросервісами та навіть gRPC робить Nest.js потужним інструментом для побудови комплексних API-систем.

Серед основних переваг Nest.js варто виділити:

- TypeScript за замовчуванням, що забезпечує статичну типізацію, автодоповнення в IDE та підвищення якості коду;
- чітка модульність, яка сприяє масштабуванню та повторному використанню компонентів;
- вбудована підтримка інжекції залежностей, що значно полегшує тестування і впровадження бізнес-логіки;
- висока сумісність з іншими бібліотеками Node.js завдяки збереженню сумісності з екосистемою Express.

Nest.js ідеально підходить для реалізації мікросервісної або монолітної архітектури, оскільки забезпечує гнучкий набір інструментів для побудови як малих, так і великих проєктів. Завдяки підтримці middleware, фільтрів, гвардій (guards), пайпів (pipes) та інтерсепторів (interceptors), фреймворк надає розробнику повний контроль над обробкою запитів.

Попри всі переваги, робота з Nest.js вимагає певного рівня володіння концепціями TypeScript та принципами архітектури застосунків. Проте це компенсується високим рівнем підтримки, детальною документацією та активною спільнотою, що полегшує процес навчання та впровадження фреймворку в нові проєкти.

Таким чином, Nest.js виступає як потужний та гнучкий інструмент для створення масштабованих бекенд-систем, забезпечуючи високу організованість коду, зручність розробки і довготривалу підтримку проєкту.

1.2.7 PHP

PHP (Hypertext Preprocessor) - це серверна мова програмування, яка з моменту свого створення у 1995 році залишається однією з найпоширеніших технологій для веб-розробки. Вона була спеціально створена для генерації динамічного HTML-контенту на стороні сервера, а з часом перетворилась на повноцінну мову загального призначення, яка активно використовується для побудови складних веб-систем [12].

PHP є інтерпретованою мовою з відкритим вихідним кодом, що забезпечує її широку доступність, гнучкість у розробці та легку інтеграцію з більшістю веб-серверів (зокрема Apache та Nginx). Однією з головних переваг PHP є її простота - синтаксис мови зрозумілий навіть початківцям, що знижує поріг входу для нових розробників.

PHP активно підтримує великий набір функцій для роботи з базами даних, обробки форм, керування сесіями, роботи з файлами, що дозволяє швидко створювати повноцінні серверні частини застосунків. Розширення у вигляді розвинених фреймворків, таких як Laravel, Symfony, CodeIgniter та інші, значно

спрощують розробку складних веб-додатків, забезпечуючи архітектурну організацію, безпеку, масштабованість і зручність тестування.

Серед ключових переваг PHP можна виділити:

- широку підтримку хостинг-провайдерів і серверного середовища;
- низький поріг входу для нових розробників;
- велика кількість документації, туторіалів та бібліотек;
- активна спільнота, що постійно підтримує розвиток мови та її фреймворків;
- висока продуктивність при правильній оптимізації коду та використанні кешування.

Водночас PHP має і певні недоліки. Основні з них пов'язані з історичною відсутністю структурного підходу до розробки (у старих версіях) та з тим, що необачне використання мови може призвести до проблем із безпекою, якщо не дотримуватись стандартів кодування. Крім того, в проєктах із високими вимогами до продуктивності та мікросервісної архітектури PHP поступається Node.js або Go через менш ефективну асинхронну модель виконання.

Сучасні реалізації PHP (починаючи з версії 7 та вище) демонструють значне зростання продуктивності, покращення підтримки об'єктно-орієнтованого програмування та введення нових функцій, таких як типізація аргументів, атрибути, а також асинхронне програмування через сторонні бібліотеки.

У підсумку, PHP залишається надійним вибором для розробки серверних частин веб-застосунків, особливо у поєднанні з фреймворками типу Laravel, які забезпечують високий рівень абстракції, безпеки та структурованості.

1.2.8 Laravel

Laravel - це популярний веб-фреймворк з відкритим кодом для мови програмування PHP, створений у 2011 році Тейлором Отвеллом. Його головна мета - зробити процес розробки веб-додатків максимально простим, елегантним і ефективним. Laravel базується на принципах MVC (Model-View-Controller), що сприяє чіткому розділенню відповідальностей між логікою, представленням та обробкою даних [13].

Фреймворк надає розробнику зручну структуру проєкту, вбудовану маршрутизацію, ORM-систему Eloquent для роботи з базами даних, систему шаблонів Blade, засоби для побудови RESTful API, черги (queues), систему подій, планувальник задач (scheduler), механізми кешування та аутентифікації. Завдяки цьому Laravel виступає повноцінним інструментом для реалізації як невеликих веб-сайтів, так і складних корпоративних систем.

Однією з ключових особливостей Laravel є ORM Eloquent, яка дозволяє працювати з базами даних у вигляді об'єктів без необхідності написання сирих SQL-запитів. Це суттєво полегшує взаємодію з реляційними СУБД (MySQL, PostgreSQL, SQLite), а також підвищує читабельність і безпечність коду.

Laravel також надає механізми міграцій та сидів, які дозволяють автоматизувати управління структурою бази даних та зручно створювати тестові дані. Інструмент Artisan CLI забезпечує швидке генерування типових структур коду, що пришвидшує розробку.

Фреймворк включає потужні засоби для тестування, зокрема підтримку PHPUnit, а також зручну роботу з middleware, механізми валідації даних, механізми авторизації на основі ролей, політик або шлюзів (gates). Інтеграція з популярними пакетами, такими як Laravel Sanctum, Passport або Jetstream, дозволяє легко реалізувати механізми аутентифікації, роботу з API та фронтендом.

Серед ключових переваг Laravel:

- високий рівень абстракції при збереженні гнучкості;
- велика активна спільнота та постійна підтримка;
- широка екосистема пакетів (Laravel Nova, Forge, Envoyer, Vapor тощо);
- добре структурована документація.

Водночас Laravel, як і будь-який фреймворк, має свої обмеження. Для високонавантажених систем із жорсткими вимогами до продуктивності може виникнути потреба у додатковій оптимізації або переході на асинхронні рішення. Також він залежить від особливостей PHP, що може впливати на вибір технології у проєктах із підвищеною складністю.

Загалом Laravel є одним із найкращих рішень для швидкої та організованої розробки бекенд-частини веб-застосунків, особливо у проєктах, де важлива зрозуміла структура, швидкий запуск MVP та хороша підтримка інструментів для API, баз даних і безпеки.

1.3 Вибір бази даних

У процесі створення інформаційних систем одним із ключових рішень є вибір відповідної системи управління базами даних (СУБД), оскільки вона безпосередньо впливає на ефективність зберігання, обробки та захисту інформації. Обрана база даних визначає не лише технічні характеристики системи, а й можливості її масштабування, стабільності, продуктивності та легкості підтримки у довгостроковій перспективі.

Під базою даних (БД) розуміють впорядковану сукупність взаємопов'язаних даних, організовану для зручного зберігання, пошуку, оновлення та аналізу. Зазвичай такі дані структуровано відповідно до певної логіки, що забезпечує швидкий доступ і можливість побудови зв'язків між елементами. Для взаємодії з базою даних використовується спеціальне програмне забезпечення - система управління базами даних (СУБД). Вона виступає посередником між користувачем, прикладною програмою та фізичним сховищем даних. [14]

У сучасній практиці бази даних поділяються на дві основні категорії: реляційні (SQL) та нереляційні (NoSQL).

Реляційна модель баз даних організована у вигляді таблиць (відношень), де кожен запис (рядок) відповідає певному об'єкту, а кожен стовпець - властивості цього об'єкта. Вона базується на строгій схемі, яка визначає структуру таблиць, їхні поля та типи даних. Реляційні СУБД підтримують мову структурованих запитів SQL, транзакційні механізми, індексацію, зв'язки між таблицями (ключі) і

нормалізацію даних. Це забезпечує високий рівень точності, узгодженості та надійності обробки даних.

Реляційні бази даних найкраще підходять для систем, де важлива складна аналітика, чітка структура даних і підтримка складних запитів - наприклад, у банківській сфері, бухгалтерії, CRM-системах або управлінні логістикою.

Натомість NoSQL-бази - це гнучкі, масштабовані системи, орієнтовані на роботу з великими обсягами неструктурованої або напівструктурованої інформації. Вони не потребують попередньо визначеної схеми та підтримують різні моделі зберігання: документну, ключ-значення, графову та колонкову. NoSQL добре підходять для рішень, які працюють із динамічними даними, що швидко змінюються, зокрема в аналітиці соцмереж, електронній комерції або геймінгу [15].

Попри свою гнучкість і продуктивність, NoSQL-бази можуть поступатися реляційним за рівнем транзакційної цілісності, підтримкою складних об'єднань даних та формуванням багатоступеневих звітів. Тому вибір між SQL і NoSQL повинен ґрунтуватися на специфіці проєкту, очікуваних навантаженнях, структурі інформації та вимогах до точності обробки.

У межах цього проєкту було розглянуто найпоширеніші СУБД, які репрезентують як реляційні, так і нереляційні підходи до зберігання даних. Серед них:

MySQL, PostgreSQL, OracleSQL, MongoDB, Cassandra, SQLite, Firebase / Firestore.

1.3.1 MySQL

MySQL - це одна з найвідоміших та найширше використовуваних реляційних систем управління базами даних з відкритим вихідним кодом. Вона була розроблена у середині 1990-х років компанією MySQL AB і згодом придбана корпорацією Oracle. MySQL є частиною класичного стеку веб-технологій LAMP (Linux, Apache, MySQL, PHP/Python), що зробило її стандартом для багатьох веб-проєктів [16].

Система побудована на основі реляційної моделі даних, що передбачає збереження інформації у вигляді таблиць з чітко визначеною структурою (схемою). MySQL підтримує транзакції, цілісність даних, зв'язки між таблицями, індексацію

та складні SQL-запити. Завдяки цьому вона ефективно використовується в системах, де важливі надійність, точність і передбачувана поведінка при обробці даних.

MySQL має декілька рушіїв зберігання, з яких найпопулярнішим є InnoDB. Цей рушій забезпечує підтримку транзакцій із властивостями ACID (атомарність, узгодженість, ізолюваність, довговічність), що гарантує стабільну та безпечну роботу з критично важливою інформацією.

Серед основних переваг MySQL можна виділити:

- високу швидкодію при обробці read-heavy навантажень (читацьких операцій);
- зрілість і стабільність системи, що пройшла випробування на тисячах великих проєктів;
- широку підтримку мов програмування і фреймворків (PHP, Python, Java, Node.js тощо);
- активну спільноту та велику кількість навчальних ресурсів;
- просту інтеграцію з багатьма популярними платформами, включно з WordPress, Drupal, Magento.

MySQL є кросплатформенною системою і може бути встановлена як на Linux-серверах, так і на Windows чи macOS. Крім того, вона підтримує реплікацію, шифрування, збережені процедури, тригери, перегляди (views), що дозволяє реалізовувати більш складні логіки взаємодії з даними.

Проте варто враховувати і деякі обмеження. MySQL менш ефективно обробляє складні аналітичні запити порівняно з такими СУБД, як PostgreSQL. Також підтримка JSON і повнотекстового пошуку є менш потужною, ніж у спеціалізованих або новіших рішень. У минулому система мала обмеження у транзакційності (через рушій MyISAM), однак сучасна версія з InnoDB значною мірою усунула ці недоліки.

У підсумку, MySQL - це надійне та добре задокументоване рішення для зберігання структурованих даних, особливо у веб-додатках із типовими CRUD-операціями. Проте для систем, де потрібна розширена обробка даних, повна

відповідність стандарту SQL або складна логіка, доцільно розглянути більш функціональні альтернативи, зокрема PostgreSQL.

1.3.2 PostgreSQL

PostgreSQL - це потужна об'єктно-реляційна система управління базами даних з відкритим вихідним кодом, яка активно розвивається ще з 1986 року, базуючись на проєкті Ingres. Вона вважається однією з найфункціональніших і надійних СУБД, які доступні на ринку безкоштовно, і користується популярністю серед розробників високонавантажених, складних систем, що потребують надійності, точності та масштабованості [17].

Однією з ключових відмінностей PostgreSQL є її повна відповідність стандарту SQL та підтримка розширених функцій, таких як:

- Common Table Expressions (CTE);
- вбудована підтримка JSONB, що дозволяє об'єднувати реляційні й документні структури в межах однієї бази;
- індекси типу GIN, GIST, BRIN, SP-GiST для оптимізації запитів;
- вбудована підтримка транзакційного контролю з властивостями ACID.

PostgreSQL також дозволяє створювати власні типи даних, функції, оператори та навіть використовувати інші мови програмування (PL/pgSQL, Python, C) для написання функцій безпосередньо в БД. Це робить систему надзвичайно гнучкою для проєктів зі складною бізнес-логікою.

Переваги PostgreSQL включають:

- високий рівень розширюваності та налаштуваності;
- відмінна продуктивність при складних запитах і великих обсягах даних;
- вбудовані засоби для паралельного виконання запитів і роботи з масивами даних;
- можливість масштабування як вертикального, так і горизонтального (через сторонні інструменти, такі як Citus);
- активна спільнота та регулярні оновлення з новими функціями.

Завдяки підтримці реплікації, резервного копіювання, контрольованого шифрування, авторизації на рівні рядків (Row-Level Security), PostgreSQL забезпечує високий рівень безпеки й відповідність вимогам корпоративного рівня.

Попри свої переваги, PostgreSQL вимагає глибшого розуміння внутрішньої архітектури при складному налаштуванні або оптимізації - особливо у випадках з великими обсягами даних чи кастомним розширенням. Проте ці особливості компенсуються великою кількістю документації та готових рішень.

1.3.3 MongoDB

MongoDB - це одна з найпоширеніших систем управління базами даних типу NoSQL, яка базується на документно-орієнтованій моделі зберігання. Вона була розроблена компанією MongoDB Inc. і вперше представлена у 2009 році як рішення для роботи з великими обсягами неструктурованих або слабо структурованих даних. MongoDB набуває все більшого поширення у проєктах, де пріоритетом є гнучкість, горизонтальне масштабування та швидкість розробки [18].

Основною одиницею зберігання в MongoDB є документ у форматі BSON (розширений двійковий формат JSON). Документи зберігаються у колекціях, які можна умовно порівняти з таблицями в реляційних базах, про те, на відміну від SQL-схем, структура документів у колекціях може бути довільною - кожен документ може мати різні поля й вкладені об'єкти.

MongoDB підтримує індексацію, агрегації, транзакції (починаючи з версії 4.0), шардінг для розподілу навантаження, реплікацію, TTL (time-to-live) колекції, повнотекстовий пошук та інші механізми, які дозволяють масштабувати систему відповідно до зростання даних або кількості користувачів.

Переваги MongoDB:

- гнучка структура даних, яка дозволяє швидко змінювати модель зберігання без потреби в міграціях;
- горизонтальне масштабування завдяки шардінгу;
- висока швидкодія при великих обсягах неструктурованої інформації;
- зручна інтеграція з JavaScript та Node.js, що прискорює розробку у фронт- і бекенд-проєктах;

- активна спільнота, велика кількість бібліотек, GUI-інструменти (Compass).

MongoDB ідеально підходить для застосунків, де дані мають динамічну структуру - зокрема це платформи електронної комерції, соціальні мережі, CMS, блоги, геймінг, реєстрація подій (event logging), IoT тощо. Її модель CRUD-операцій дуже зручна для розробників, оскільки нагадує роботу з JSON-об'єктами у сучасних мовах програмування.

Водночас, MongoDB не позбавлена недоліків:

- відсутність жорсткої схеми може призвести до неузгодженості даних при неправильному проектуванні;
- обмежена підтримка транзакцій, хоча останні версії суттєво її покращили;
- низька ефективність при складних об'єднаннях (join), що характерно для реляційних моделей;
- менша предикативна безпека та відсутність повної SQL-сумісності, що обмежує її застосування у фінансових або аналітичних системах.

MongoDB може бути цінним інструментом при побудові прототипів або мобільних застосунків. Проте у випадках, коли потрібна сувора нормалізація, складні зв'язки між сутностями або формування детальних звітів, більш доцільно застосовувати реляційні СУБД, зокрема PostgreSQL.

1.3.4 Apache Cassandra

Apache Cassandra - це високопродуктивна, горизонтально масштабована система управління базами даних типу NoSQL, яка орієнтована на обробку великих обсягів даних у розподіленому середовищі. Вона була спочатку розроблена у Facebook для реалізації системи повідомлень, а згодом передана до фонду Apache Software Foundation як проєкт з відкритим кодом [19].

Cassandra базується на колонковій моделі зберігання, яка відрізняється від традиційних таблиць у реляційних СУБД. Дані організовані у вигляді ключів і значень, де значення представлені як набори колонок, що дозволяє ефективно працювати з великими масивами інформації, орієнтованими на читання/запис по ключу. Така модель є оптимальною для аналітичних задач, роботи з часовими рядами та великими потоками телеметрії.

Однією з основних переваг Cassandra є її "masterless" архітектура. На відміну від більшості СУБД, у Cassandra всі вузли кластера рівноправні - відсутній центральний координаційний сервер. Завдяки цьому забезпечується висока відмовостійкість, автоматична балансування навантаження та легке масштабування без зупинки системи.

Ключові особливості Cassandra:

- підтримка реплікації даних між датацентрами - з можливістю налаштування
- стратегій реплікації для забезпечення доступності;
- консистентність "на вибір" (eventual consistency), що дозволяє балансувати між продуктивністю та точністю;
- розподілена обробка запитів, що забезпечує рівномірне навантаження;
- мова CQL (Cassandra Query Language) - SQL-подібний синтаксис для зручної роботи з даними;
- висока швидкодія при масовому записі/читанні - ідеально підходить для Big Data-сценаріїв.

Попри численні переваги, Cassandra має і певні обмеження. Система не призначена для обробки складних об'єднань (joins), підзапитів або транзакцій на рівні кількох таблиць. Також вона вимагає ретельного проєктування моделі даних, оскільки запити мають бути оптимізовані під конкретну структуру зберігання. Без цього ефективність системи може суттєво знижуватись.

Cassandra найчастіше використовується в проєктах, пов'язаних із:

- аналітикою в реальному часі;
- обробкою логів та телеметрії (наприклад, IoT);
- георозподіленими системами;
- фінансовими потоками з великим обсягом подій.

У контексті мобільних застосунків Cassandra є виправданим вибором у випадках, коли потрібно забезпечити безперервний збір великого потоку подій з багатьох пристроїв (наприклад, фітнес-трекери, датчики, поведінкові логи). Проте, для класичного CRUD-застосунку з помірним обсягом структурованих даних та

аналітичними запитами - більш практичним буде використання реляційних СУБД, таких як PostgreSQL.

1.4 Висновки щодо вибору СУБД

У процесі створення мобільного застосунку для пошуку друзів за інтересами було сформовано базові технічні вимоги: швидка розробка MVP, надійне зберігання даних, простота супроводу та можливість масштабування в майбутньому. Враховуючи ці цілі, було прийнято рішення реалізувати як клієнтську, так і серверну частину у вигляді монолітної архітектури, що значно спрощує логіку взаємодії компонентів на ранніх етапах розробки.

Клієнтська частина реалізована з використанням мови Kotlin та Jetpack Compose - сучасного UI-фреймворку для декларативної розробки інтерфейсів. Цей вибір забезпечує не лише високу швидкість створення візуального функціоналу, але й простоту подальшої підтримки та адаптації під сучасні стандарти Android-розробки.

Бекенд-система побудована на платформі Node.js у поєднанні з Nest.js - прогресивним фреймворком, що реалізує модульну архітектуру, підтримку TypeScript, інтегровану валідацію, логування та маршрутизацію REST-запитів. Це дозволяє створювати чисту та стабільну кодову базу.

У якості системи управління базами даних для серверної частини обрано PostgreSQL, яка підтримує складні запити, транзакції, JSON-поля, повнотекстовий пошук та розширення. Завдяки сумісності з ORM-бібліотекою TypeORM, забезпечено зручну роботу з даними на рівні сутностей, а також ефективне управління схемою через міграції.

2 РОЗРОБКА ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Визначення та аналіз вимог до програмного забезпечення

На даному етапі основні питання – це визначення ступеня деталізації вимог та способів їх представлення.

Вимоги користувачів, які мають бути задоволені програмним забезпеченням:

- зручність пошуку користувачів за інтересами: можливість швидко знайти людей з подібними інтересами через застосування розширених фільтрів;
- групи за інтересами: створення та участь у групах на основі спільних хобі;
- приватність та безпека: можливість налаштовувати видимість профілю та персональних даних, блокування небажаних користувачів;
- легка навігація та UX/UI: додаток має бути інтуїтивно зрозумілим, з приємним інтерфейсом.

Для забезпечення безпеки користувачів та ефективної обробки даних, система використовує захищені протоколи обміну інформацією та сучасні методи шифрування. Це необхідно для того, щоб персональні дані користувачів залишалися конфіденційними та захищеними від зловживань або несанкціонованого доступу.

2.2 Розробка специфікації вимог SRS

2.2.1 Введення (Розділ 1 SRS)

2.2.1.1 Призначення (Підрозділ 1.1 SRS)

Специфікація вимог програмного забезпечення призначена для розробки мобільного додатка, який дозволяє користувачам знаходити друзів за спільними інтересами. Ця SRS визначає вимоги до додатка, який є повністю самостійним,

набір сценаріїв використання, що описують взаємодію користувача з програмним забезпеченням.

SRS призначена для розробників та тестувальників, які відповідають за створення та підтримку якісного програмного продукту.

2.2.1.2 Область дії (Підрозділ 1.2 SRS)

Програмний продукт має назву “MateUp”. Мобільний додаток повинен мати зручний інтерфейс, який дозволить швидко та ефективно шукати друзів за інтересами. Цільова аудиторія – користувачі мобільних пристроїв, які шукають можливості для соціалізації та нових знайомств

2.2.1.3 Визначення, акроніми та скорочення (Підрозділ 1.3 SRS)

ПЗ – програмне забезпечення, набір програм для виконання різних завдань на мобільному пристрої.

API (Application Programming Interface) – інтерфейс програмування додатків, що дозволяє взаємодіяти з іншими системами.

UI (User Interface) – користувацький інтерфейс, що забезпечує зручну взаємодію між людиною і додатком.

UX (User Experience) – досвід користувача при використанні продукту.

2.2.1.4 Публікації (Підрозділ 1.4 SRS)

Закон України "Закон України Про авторське право і суміжні права" охороняє особисті немайнові права і майнові права авторів та їх правонаступників, пов'язані із створенням та використанням творів науки, літератури і мистецтва - авторське право, і права виконавців, виробників фонограм і відеограм та організацій мовлення - суміжні права.

Стандарт ISO/IEC/IEEE 26514:2022 – містить вимоги до проектування та розробки документації користувача програмного забезпечення як частини процесів життєвого циклу, визначає процес документування з точки зору розробника документації, визначає структуру, зміст і формат документації користувача, а також надає інформативні вказівки щодо стилю документації користувача, не залежить від програмних засобів, які можна використовувати для створення документації, і застосовується як до друкованої документації, так і до документації на екрані.

Стандарт ISO/IEC/IEEE 24765:2017 – надає загальний словник, застосовний до всіх систем і програмної інженерії, був підготовлений для збору та стандартизації термінології, містить посилання на стандарти активних джерел для визначень.

2.2.1.5 Короткий огляд (Підрозділ 1.5 SRS)

Документ специфікації вимог до програмного забезпечення структуровано у три основні розділи, кожен з яких деталізується підрозділами з послідовною нумерацією. Перший розділ охоплює призначення ПЗ, ключові системні вимоги, термінологію, акроніми, скорочення та бібліографічні посилання. Другий розділ надає загальний опис, включаючи контекст продукту, його функціональність, обмеження, припущення та залежності. Третій розділ детально розкриває вимоги до програмного продукту, зокрема аспекти надійності, безпеки та інтерфейсів.

2.2.2 Загальний опис (Розділ 2 SRS)

2.2.2.1 Контекст програмного продукту (Підрозділ 2.1 SRS)

Даний програмний продукт має зручний інтерфейс, який є інтуїтивно зрозумілим для користувачів, а також адаптується до екрану будь-якого пристрою.

2.2.2.2 Функції програмного продукту (Підрозділ 2.2 SRS)

Програмний продукт MateUp виконує наступні функції:

- створення та редагування профілю користувача;
- пошук друзів за інтересами за допомогою фільтрів;
- блокування небажаних користувачів для забезпечення безпеки;
- додавання та видалення людей з друзів.

2.2.2.3 Характеристики користувача (Підрозділ 2.3 SRS)

Цільові користувачі мають бути знайомі з базовими функціями мобільних додатків. Додаток розрахований на широкий спектр користувачів різного віку та навичок.

2.2.2.4 Обмеження (Підрозділ 2.4 SRS)

У даному підрозділі визначено апаратні та програмні обмеження, яких має дотримуватися кінцевий користувач для забезпечення коректної роботи мобільного застосунку

Таблиця 2.1 - Мінімальні вимоги до користувача

Компонент	Характеристика
Операційна системаAndroid	8.0 (API рівень 26)
Оперативна пам'ять	2 ГБ
Вільне місце на накопичувачі	70 МБ
Процесор	Qualcomm Snapdragon 450, Samsung Exynos 7884, MediaTek Helio P22 або еквівалент
Підключення доІнтернету	Стабільне з'єднання зі швидкістю від 10 Мбіт/с
Дозволи	Доступ до камери, сховища

Крім основних вимог, користувачам також рекомендується мати оновлене програмне забезпечення та достатній заряд батареї для забезпечення сумісності та безпеки під час використання додатка. Оновлені версії операційної системи забезпечують кращу продуктивність, безпеку та підтримку сучасних веб-технологій, що є важливим для коректної роботи додатка.

Таблиця 2.2 - Рекомендовані вимоги до користувача

Компонент	Характеристика
Операційна системаAndroid	Android 12.0 (API рівень 31)
Оперативна пам'ять	4 ГБ або більше
Вільне місце на накопичувачі	150 МБ або більше
Процесор	Qualcomm Snapdragon 660, Samsung Exynos 9611, MediaTek Helio G70 або потужніше
Підключення до Інтернету	Стабільне з'єднання зі швидкістю від 25 Мбіт/с
Дозволи	Доступ до камери, сховища

2.2.2.5 Допущення і залежності (Підрозділ 2.5 SRS)

Фактори, що можуть вплинути на вимоги:

— додавання нових функцій для покращення досвіду користувачів (наприклад, розширені фільтри пошуку, підтримка нових мов);

- зміни в політиках конфіденційності та умовах користування платформами соціальних мереж, з якими інтегрується додаток;
- впровадження нових технологій для оптимізації швидкості пошуку друзів.

2.2.2.6 Розподілення вимог (Підрозділ 2.6 SRS)

Деякі вимоги, які можуть бути відкладені для майбутніх версій додатку для покращення користувацького досвіду в додатку для пошуку друзів за інтересами:

- можливість залишати голосові повідомлення: функція відправки голосових повідомлень для полегшення спілкування між користувачами;
- спільні календарі для планування подій: можливість створювати і ділитися календарями з друзями для координації зустрічей або спільних заходів.

Вимоги до інтерфейсу також можуть бути відкладені до появи майбутніх версій системи.

2.2.3 Специфічні вимоги (Розділ 3 SRS)

2.2.3.1 Функції інтерфейсу користувача

Після запуску мобільного застосунку користувач одразу потрапляє на стартовий екран, де пропонується обрати одну з двох дій: «Створити акаунт» або «Увійти».

Під час реєстрації нового користувача відкривається екран профілю, де він вводить особисту інформацію: ім'я, вік, країну, місто, інтереси та фото. Після заповнення профілю або після авторизації через кнопку «Увійти» користувач переходить на головний екран застосунку.

Основна взаємодія з додатком відбувається саме на головному екрані, де доступні такі можливості:

- стрічка користувачів – відображає картки користувачів із короткою інформацією. Звідси можна швидко додати в друзі, переглянути профіль, або розпочати чат;
- пошуковий рядок – дозволяє шукати людей за ім'ям, країною або містом безпосередньо з головного екрану;
- фільтри для пошуку – дають змогу сортувати користувачів за віком, інтересами чи локацією;

– кнопка профілю в меню навігації – забезпечує швидкий доступ до налаштувань акаунту та редагування особистих даних;

– розділ друзів – окремий екран для перегляду списку друзів, надсилання запрошень, прийняття чи відхилення заявок на дружбу.

2.2.3.2 Атрибути якості програмної системи

2.2.3.2.1 Надійність

Додаток має стабільно працювати на мобільних пристроях з різними технічними характеристиками та підтримувати резервне збереження даних.

2.2.3.2.2 Захист

Згідно закону України "Про авторське право і суміжні права", використовувати у власному користуванні, без права передачі третім особам.

2.2.3.2.3 Зручність супроводження

Програма реалізована за стандартними архітектурними шаблонами, які дозволять легко масштабувати проект, вводячи новий функціонал, або внести до нього зміни.

2.3 Діаграми варіантів використання

UML (Unified Modeling Language) – це універсальна графічна мова моделювання, призначена для візуалізації, специфікації, конструювання та документування програмних систем. Вона надає стандартизований інструмент для представлення систем з різних ракурсів, сприяючи ефективній співпраці між розробниками, аналітиками та іншими учасниками проєкту протягом усього життєвого циклу розробки.

UML поділяється на два основні типи діаграм: структурні та поведінкові. Структурні діаграми фокусуються на статичних аспектах системи, таких як класи, об'єкти та їхні взаємозв'язки. До них належать класові діаграми, що відображають структуру класів, їхні атрибути, методи та зв'язки; діаграми компонентів, які

описують фізичні складові системи; та діаграми об'єктів, що фіксують стан конкретних об'єктів у певний момент часу. На противагу цьому, поведінкові діаграми ілюструють динамічні аспекти системи, зокрема взаємодію між компонентами або зміни станів об'єктів. Серед них виділяються діаграми варіантів використання, що описують функції системи з точки зору користувача, а також діаграми послідовностей, активностей та станів, що демонструють динаміку роботи системи.

UML значно полегшує спільну роботу, допомагаючи розробникам чітко візуалізувати структуру та поведінку системи, зменшуючи кількість непорозумінь і забезпечуючи чітке документування. Це робить підтримку та подальшу еволюцію системи простішою та ефективнішою. Наприклад, діаграми варіантів використання моделюють вимоги програми, показуючи, які функції вона виконує для користувачів та які ролі ці користувачі відіграють. Завдяки своїй універсальності та стандартизації, UML ідеально підходить для моделювання систем будь-якої складності, від невеликих додатків до масштабних корпоративних платформ, що робить її невід'ємною частиною сучасного процесу розробки програмного забезпечення.

Діаграма варіантів використання додатку для пошуку друзів за інтересами зображена нарисунку 2.1.

Основний актор програми: користувач, що шукає друзів за інтересами та взаємодіє з програмою

В межах програми було визначено шість основних прецедентів, що на діаграмі позначені відповідними зв'язками включення. Стислий опис кожного прецедента:

- реєстрація – користувач створює профіль, вказуючи особисту інформацію та інтереси;
- авторизація – користувач входить у систему, використовуючи свої облікові дані;
- пошук за інтересами – функція пошуку інших користувачів із подібними інтересами;

- редагування профілю – редагування власних персональних даних інтересів для точнішого підбору;
- надсилання запитів на дружбу – користувач надсилає запити на дружбу іншим користувачам;
- відповідь запитів на дружбу – користувач приймає або відхилення запитів на дружбу від інших.

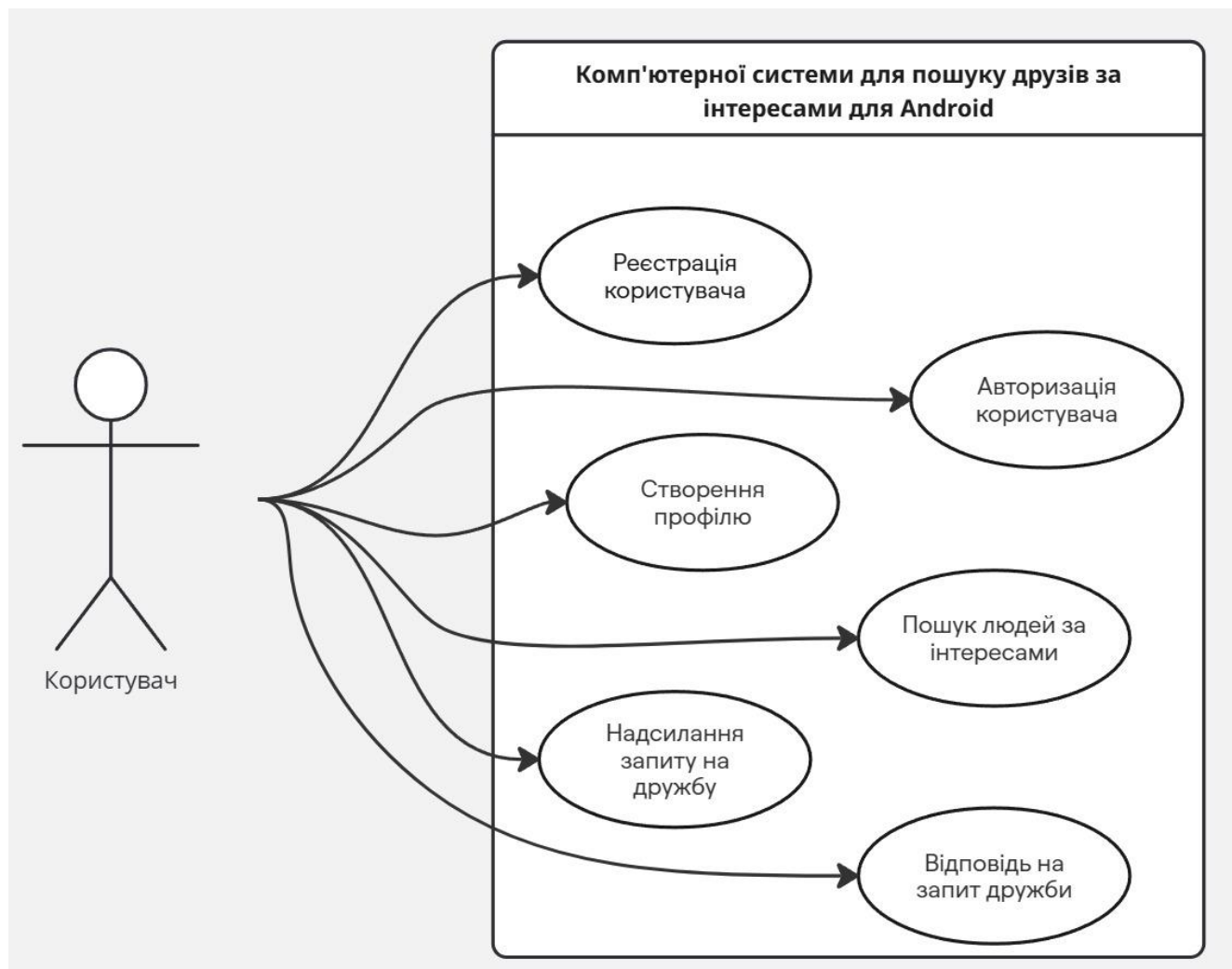


Рисунок 2.1 – Діаграма варіантів використання додатку для пошуку друзів за інтересами

2.4 Діаграми послідовності

Діаграми послідовності в додатку для пошуку друзів за інтересами є ключовими для наочного відображення взаємодій між користувачем та різними елементами системи, такими як база даних профілів або сервер пошуку. Їхня основна роль полягає у візуалізації послідовності обробки запитів користувача, що зрештою призводить до бажаного результату – знайдення або додавання нових друзів за спільними інтересами. Ці діаграми детально ілюструють обмін повідомленнями між об'єктами системи, розкриваючи часову залежність операцій. Завдяки їм розробники можуть легко ідентифікувати потенційні вузькі місця або неефективні процеси в роботі програми.

Ці діаграми фокусуються не стільки на змісті самих повідомлень, скільки на порядку їх слідування, що дозволяє чітко зрозуміти, як саме система обробляє дані для досягнення поставленої мети. Наприклад, діаграма детально показує покроковий процес: від надсилання запиту користувача до серверу, його подальшої передачі до бази даних, і, нарешті, повернення відповіді користувачеві. Вертикальний вимір діаграми відображає часову послідовність подій, рухаючись зверху вниз, тоді як горизонтальний вимір представляє різні компоненти системи, з якими взаємодіє користувач. Така візуалізація дозволяє чітко побачити взаємозв'язок між компонентами та глибоко зрозуміти логіку їхньої роботи в контексті виконання конкретного сценарію. Кожна стрілка на діаграмі символізує конкретну дію або передачу даних, що робить процес прозорим. Завдяки цьому інструменту, навіть складні сценарії взаємодії стають легко зрозумілими для всіх учасників розробки.

У даному проекті діаграми послідовності відносно прості, оскільки в них задіяний один основний актор – користувач, що шукає друзів за інтересами, та система, яка обробляє ці запити і повертає результати (рисунки 2.2 – 2.6).

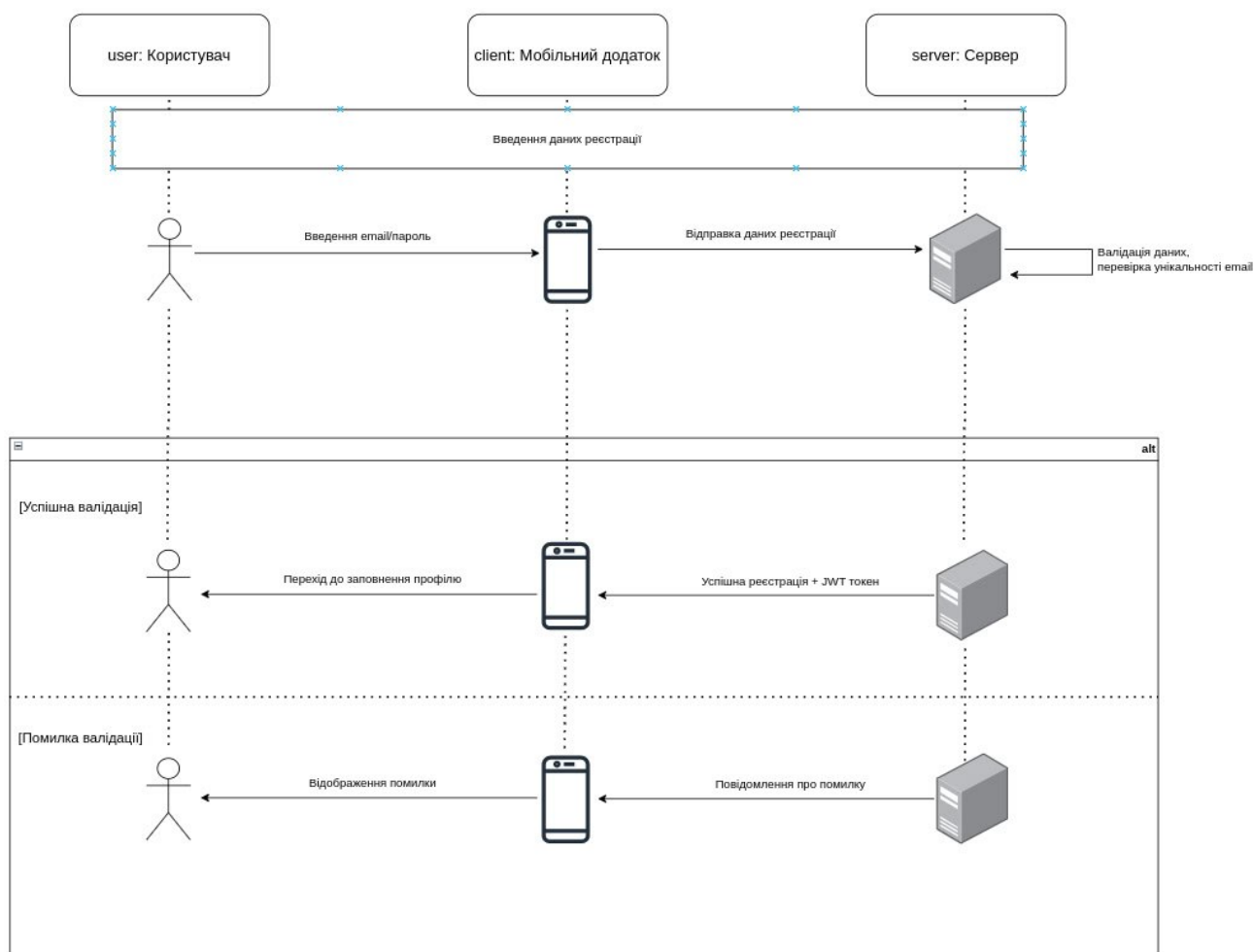


Рисунок 2.2 – Запит на реєстрацію користувача

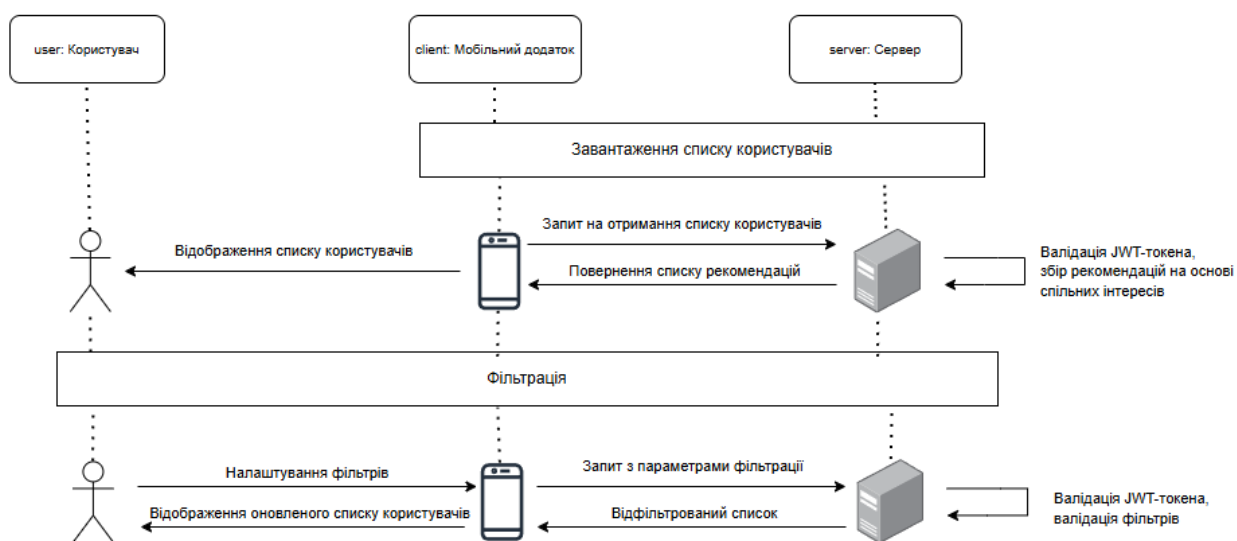


Рисунок 2.3 – Запит на отримання списку користувачів на основі спільних інтересів для пошуку друзів

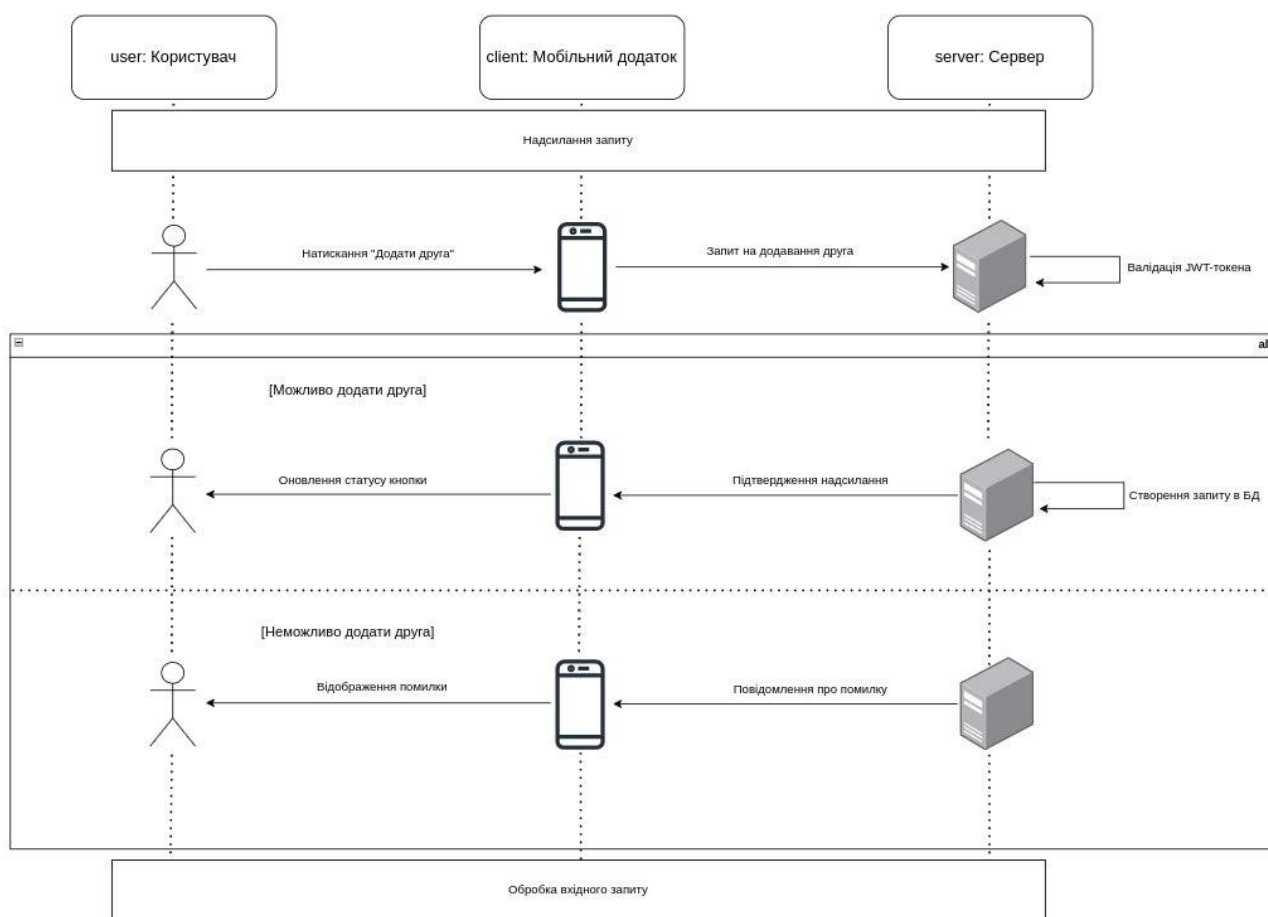


Рисунок 2.4 – Запит на дружбу (надсилання запиту)

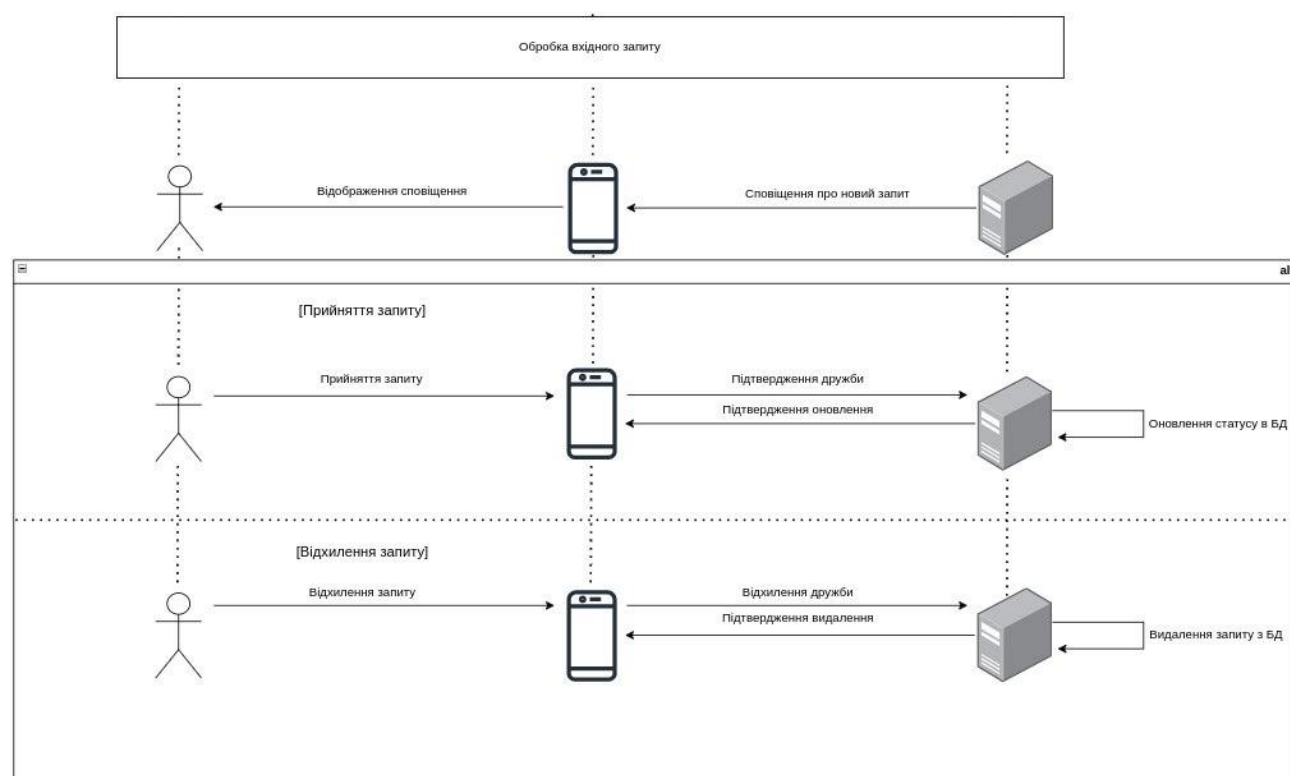


Рисунок 2.5 –Відповідь на запит дружби (обробка вхідного запиту)

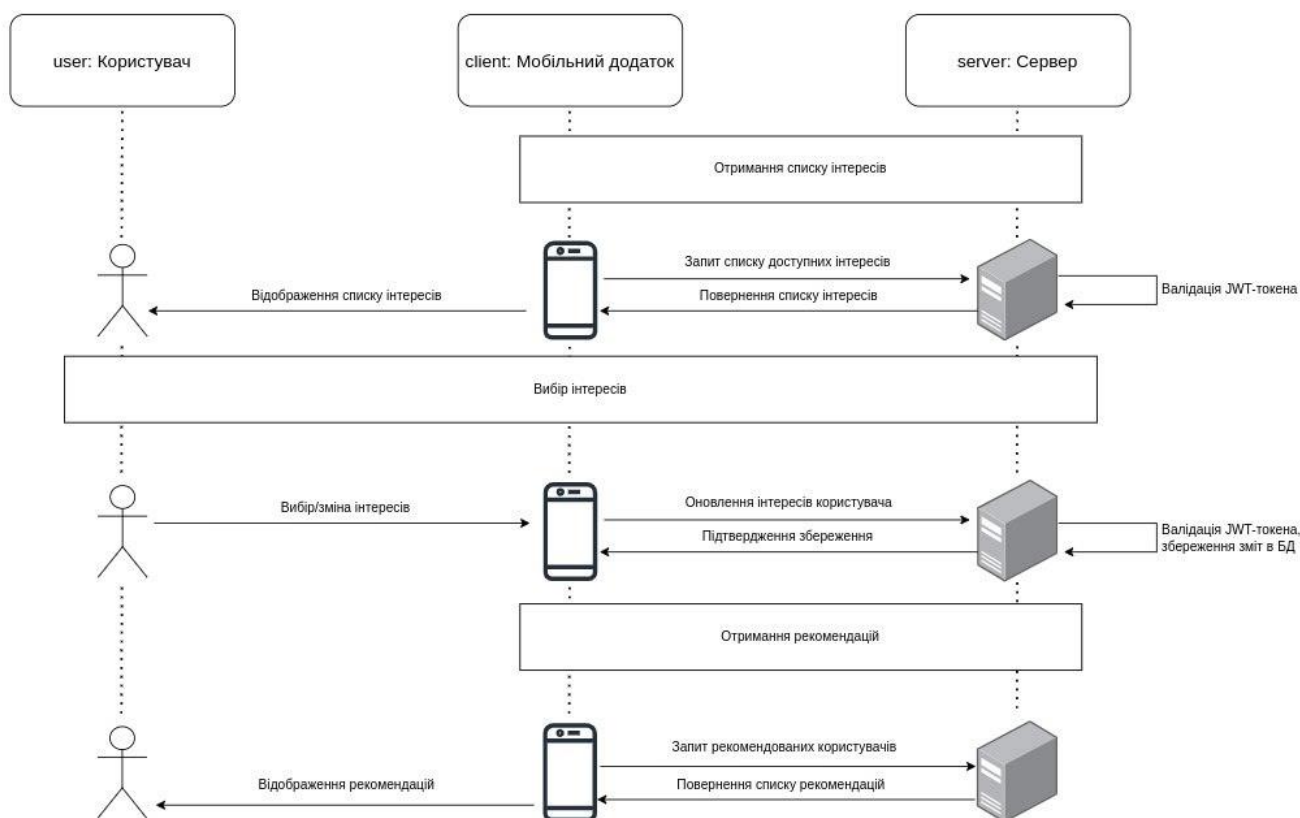


Рисунок 2.6 – Налаштування інтересів

2.5 Розгорнуті описи прецедентів

Розгорнуті описи прецедентів є важливим елементом деталізації функціональності додатка для пошуку друзів за інтересами. Вони уточнюють вимоги та описують ключові сценарії використання, як-от реєстрація користувача або пошук друзів. Це полегшує розробникам і тестувальникам розуміння очікуваної поведінки системи й формування тестових сценаріїв.

2.5.1 Прецедент П1. Реєстрація користувача в додатку.

Основний виконавець: Користувач.

Зацікавлені особи та їх потреби:

- Користувач: бажає створити обліковий запис, щоб отримати доступ до функцій пошуку друзів.

Передумови: Користувач повинен мати активне підключення до Інтернету. Для реєстрації потрібно надати дійсну електронну пошту або номер телефону.

Результати (Постумови): Після успішної реєстрації користувач отримує доступ до основних функцій додатка, включаючи можливість налаштувати власний профіль та переглядати профілі інших користувачів.

Основний успішний сценарій:

- користувач завантажує додаток і відкриває його;
- користувач натискає на кнопку «Зареєструватися» та заповнює форму з персональними даними (ім'я, електронна пошта, пароль тощо);
- програма надсилає підтверджувальний код на номер телефону;
- користувач вводить код для підтвердження своєї особистості;
- програма зберігає дані користувача і створює новий обліковий запис.

Розширення або альтернативні потоки:

- Якщо код підтвердження неправильний, користувач отримує відповідне повідомлення про помилку і можливість повторного введення коду.

Спеціальні вимоги: Висока безпека даних користувачів. Ім'я користувача та електронна пошта є унікальними (користувачів з однаковим ім'ям або електронною поштою не повинно бути)

Частота використання: Один раз на користувача.

Відкриті питання: Як обробляти ситуацію з втратою доступу до електронної пошти або номеру телефону?

2.5.2 Прецедент П2. Пошук користувачів за інтересами.

Основний виконавець: Користувач.

Зацікавлені особи та їх потреби:

- Користувач: хоче знайти нових друзів за спільними інтересами.

Передумови: Користувач повинен мати налаштований профіль з вказаними інтересами.

Результати (Постумови): Після виконання пошуку користувач отримує список потенційних друзів.

Основний успішний сценарій:

- користувач переходить до вкладки «Пошук друзів», так званий фід (від англ. feed);
- користувач обирає параметри пошуку (інтереси, місцезнаходження тощо);
- програма обробляє запит і виводить список профілів, які відповідають критеріям.

Розширення або альтернативні потоки:

- Якщо не знайдено відповідних користувачів, система пропонує розширити критерії пошуку.

Спеціальні вимоги: Висока швидкість роботи системи пошуку.

Частота використання: Постійно.

Відкриті питання: Які фільтри використовувати для покращення результатів пошуку?

2.5.3 Прецедент ПЗ. Відправка запиту на дружбу.

Основний виконавець: Користувач.

Зацікавлені особи та їх потреби:

- Користувач: хоче відправити запит на дружбу іншому користувачу.

Передумови: Користувач знайшов профіль іншого користувача.

Результати (Постумови): Після успішної відправки запиту інший користувач отримує сповіщення про запит на дружбу.

Основний успішний сценарій:

- користувач переглядає профіль іншого користувача;
- користувач натискає на кнопку «Додати в друзі»;
- програма надсилає запит і повідомляє користувача про очікування відповіді.

Спеціальні вимоги: Можливість відміни запиту на дружбу.

Частота використання: Регулярно.

Відкриті питання: Чи можна надсилати повторний запит після відхилення?

2.5.4 Прецедент П4. Редагування профілю користувача.

Основний виконавець: Користувач.

Зацікавлені особи та їх потреби:

– Користувач: бажає оновити власну інформацію (фото, ім'я, опис, контакти) для підтримки актуальності профілю.

Передумови: Користувач повинен бути авторизованим у додатку та мати активне підключення до Інтернету.

Результати (Постумови): Збережені зміни одразу відображаються в особистому профілі користувача та стають доступними для перегляду іншими користувачами.

Основний успішний сценарій:

- користувач відкриває додаток і заходить у розділ «Профіль»;
- користувач натискає кнопку «Редагувати профіль»;
- відкривається форма редагування з полями (ім'я, опис, фото, контакти тощо);
- користувач вносить зміни в потрібні поля;
- користувач натискає кнопку «Зберегти»;
- програма перевіряє правильність введених даних;
- у разі успішної перевірки дані зберігаються в системі;
- оновлений профіль відображається користувачу.

Розширення або альтернативні потоки:

– Якщо дані не пройшли перевірку (наприклад, ім'я порожнє або неправильний формат електронної пошти) — користувач отримує повідомлення про помилку з підказкою та можливість виправити.

Спеціальні вимоги: Валідація унікальності ім'я користувача (якщо потрібно) або перевірка допустимих символів.

Частота використання: За потребою користувача.

Відкриті питання: Чи дозволяти редагувати лише певні поля (наприклад, не змінювати дату народження після першого введення)?

2.5.5 Прецедент П5. Відповідь на запит дружби

Основний виконавець: Користувач.

Зацікавлені особи та їх потреби:

– Користувач: бажає прийняти або відхилити запрошення в друзі для керування власним колом спілкування.

Передумови: Користувач має активний обліковий запис, авторизований у додатку, і отримав одне чи кілька запрошень в друзі.

Результати (Постумови): Після прийняття – користувач і той, хто надіслав запрошення. Після відхилення – запит видаляється.

Основний успішний сценарій:

- користувач відкриває додаток;
- користувач заходить у розділ «Запити в друзі»;
- додаток відображає список отриманих запрошень;
- користувач вибирає конкретне запрошення;
- користувач натискає «Прийняти» або «Відхилити»;
- якщо «Прийняти» – система додає обох користувачів у список друзів;
- якщо «Відхилити» – запит видаляється з системи;
- відображається повідомлення про успішне виконання дії.

Розширення або альтернативні потоки:

– Якщо під час відповіді на запит сталася помилка мережі — виводиться повідомлення з можливістю повтору.

Спеціальні вимоги: Нотифікація іншому користувачу про прийняття чи відхилення.

Частота використання: Залежить від активності користувача.

Відкриті питання: Чи дозволяти відправляти новий запит після відхилення (з яким інтервалом)?

3 РОЗРОБКА СИСТЕМИ

3.1 Розробка клієнтської частини системи

3.1.1 Налаштування середовища розробки

Розробка мобільного додатку "MateUp" для платформи Android розпочинається з налаштування середовища розробки через конфігураційний файл `build.gradle.kts` (поданий у додатку А), який визначає плагіни, залежності, параметри збірки та конфігурацію проєкту.

Файл `build.gradle.kts` визначає конфігурацію проєкту "MateUp" для Android. У блоці `plugins` підключаються плагіни для Android-додатків (`android.application`), підтримки Kotlin (`kotlin.android`), Jetpack Compose (`kotlin.compose`), серіалізації (`kotlinx.serialization`), ін'єкції залежностей через Hilt (`com.google.dagger.hilt.android`) та Parcelable (`kotlin-parcelize`).

Параметри `android` встановлюють простір назв (`com.example.mateup`), версію компіляції (`compileSdk = 35`), мінімальну (`minSdk = 26`) і цільову (`targetSdk = 35`) версії Android, а також версію додатку (`versionCode = 1`, `versionName = "1.0"`). Ресурси `MATEUP_BASE_URL` і `MATEUP_WEBSOCKET_URL` конфігуруються через `gradleLocalProperties` для забезпечення гнучкості API-запитів. Увімкнено підтримку Jetpack Compose через `buildFeatures { compose = true }`, а компіляція налаштована для Java 1.8.

Блок `dependencies` містить бібліотеки, необхідні для розробки додатку. `retrofit` (2.11.0) і `converter-gson` (2.11.0) забезпечують HTTP-запити та обробку JSON. `logging-interceptor` допомагає налагоджувати мережеві запити. `kotlinx-coroutines` (`core` та `android`) підтримує асинхронне програмування. `material3` і `androidx.material.icons.extended` (1.7.7) забезпечують сучасний дизайн інтерфейсу, сумісний з Material Design 3. `coil3-coil-compose` і `coil-network-okhttp` відповідають за завантаження та відображення зображень. `jwtdecode` використовується для обробки JWT-токенів під час аутентифікації.

Для керування станом і навігацією додатку використовуються `androidx.lifecycle.runtime.compose`, `androidx.lifecycle.viewmodel.compose` та `androidx.navigation.compose`. `hilt.android (2.51.1)` і `hilt-navigation` ~

`System: compose` забезпечують ін'єкцію залежностей і навігацію в `Compose.accompanist.permissions` спрощує обробку дозволів `Android.androidx.datastore.preferences` (1.1.4)` використовується для зберігання налаштувань користувача.

Для забезпечення якості коду та тестування підключено `junit` для модульних тестів, `androidx.junit` та `androidx.espresso.core` для інструментальних тестів, а також `androidx.ui.test.junit4` для тестування інтерфейсу `Compose`. Бібліотеки `androidx.ui`, `androidx.ui.graphics`, `androidx.ui.tooling.preview` і `androidx.ui.tooling` забезпечують розробку та налагодження UI, а `androidx.core.ktx` і `androidx.activity.compose` додають базові утиліти для `Android` і `Compose`.

Ці інструменти та бібліотеки забезпечують створення масштабованого, безпечного та зручного `Android`-додатку "MateUp" із сучасним інтерфейсом, підтримкою мережових запитів, аутентифікацією та ефективним керуванням станом. Ретельна конфігурація `build.gradle.kts` є фундаментом для стабільної та функціональної збірки застосунку. Використання актуальних версій бібліотек гарантує доступ до новітніх можливостей та покращень безпеки. Інтеграція `Hilt` спрощує керування залежностями, роблячи кодову базу більш підтримуваною. Наявність бібліотек для тестування на різних рівнях підкреслює важливість забезпечення високої якості продукту.

3.1.2 Створення структури проєкту

Розробка сучасного `Android`-додатку вимагає ретельного підходу до організації архітектури, що забезпечує зручність розробки, підтримку коду та можливість масштабування. Правильно спроектована структура проєкту є ключовим фактором, який впливає на ефективність командної роботи, спрощення інтеграції нових функцій і зменшення технічних складнощів у процесі розвитку продукту. При створенні додатку "MateUp" для `Android` основна увага приділялася формуванню логічної та гнучкої архітектури, яка відповідає сучасним стандартам

мобільної розробки та підтримує принципи модульності й ізоляції функціональних компонентів. Структура проєкту "MateUp" представлена на рисунку 3.1.

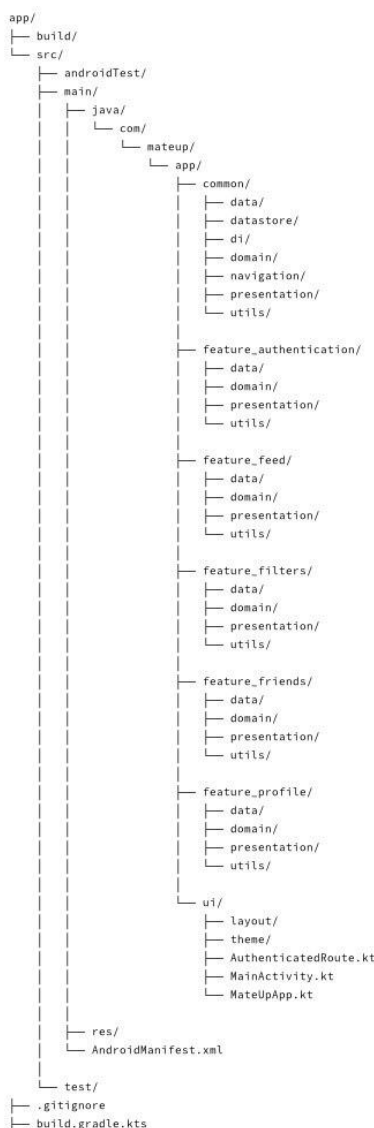


Рисунок 3.1 - Структура проєкту "MateUp"

Основою проєкту є коренева папка `app`, яка містить увесь вихідний код і ресурси додатку. У ній виділено кілька ключових піддиректорій, що відповідають за різні аспекти функціональності. Директорія `src/` є основною та поділяється на підпапки для модульної організації коду, а саме: `androidTest/`, що містить інструментальні тести для перевірки функціональності додатку на пристроях Android (у підпапці `main` розміщені тестові класи); `main/`, де знаходиться основний код додатку, розподілений за пакетами в структурі `java/com/mateup`, включаючи `app/`

(основний пакет додатку, що включає `MateUpApp.kt`, `MainActivity.kt` та `AuthenticatedRoute.kt`), `common/` (утиліти та загальні функції, що використовуються в усьому додатку, з підпапками `data/` і `datastore/`, `di/`, `domain/`, `navigation/`, `presentation/` та `utils/`), та `feature/` (модулі, що відповідають за окремі функціональні блоки додатку, такі як `feature_authentication/`, `feature_feed/`, `feature_filters/`, `feature_friends/` та `feature_profile/`, кожен з яких має підпакети `data/`, `domain/`, `presentation/` і `utils/`); а також `ui/`, що містить глобальні UI-компоненти та стилі (`layout/` та `theme/`). Окрім цього, є директорія `res/`, яка містить ресурси додатку, включаючи макети, іконки, строки та інші статичні активи, а файл `AndroidManifest.xml` визначає основні налаштування додатку, такі як дозволи та компоненти. На верхньому рівні проєкту розташовані конфігураційні файли: `build.gradle.kts` (налаштування збірки), `proguard-rules.pro` (правила оптимізації для ProGuard), а також `.gitignore` для виключення тимчасових файлів із системи контролю версій.

Така модульна структура проєкту "MateUp" відповідає принципам чистої архітектури, розділяючи шари даних (`data/`), бізнес-логіки (`domain/`) та інтерфейсу користувача (`presentation/`). Це сприяє ізоляції відповідальностей, спрощує тестування, підвищує гнучкість коду та забезпечує зручність масштабування додатку в майбутньому.

3.1.3 Побудова глобального макета додатка

У розробці Android-додатку "MateUp" забезпечення послідовного інтерфейсу користувача є ключовим аспектом. Глобальний макет у контексті Jetpack Compose реалізується через ініціалізацію основної активності (`MainActivity`) та систему навігації (`AppNavigation`), які разом визначають структуру додатка, забезпечують єдиний стиль через тему (`MateUpTheme`) та керують переходами між екранами. Такий підхід дозволяє уникнути дублювання коду та забезпечує зручний користувацький досвід.

Основна точка входу додатка - це `MainActivity`, яка ініціалізує глобальну тему та навігацію. Код файлу `MainActivity.kt` наведено у лістингу 4.1.

Лістинг 3.1 – Код файлу MainActivity.kt

```
@AndroidEntryPoint
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            MateUpTheme {
                AppNavigation()
            }
        }
    }
}
```

У MainActivity метод onCreate викликає enableEdgeToEdge() для повноекранного режиму та встановлює базовий Compose-компонент через setContent. Всередині setContent застосовується глобальна тема MateUpTheme, яка забезпечує єдиний стиль (кольори, типографію) для всіх екранів додатка. Компонент AppNavigation відповідає за маршрутизацію та відображення екранів.

Для керування навігацією між екранами використовується AppNavigation, який реалізує маршрутизацію та визначає логіку переходів між основними екранами додатка. Код файлу AppNavigation.kt наведено у Додатку А.

AppNavigation використовує Jetpack Navigation для створення маршрутів між екранами додатка. Початковий маршрут (startDestination) залежить від стану автентифікації: якщо користувач авторизований і його профіль заповнений, відкривається FeedScreen, інакше - ProfileScreen або LoginScreen. Кожен екран обгортається у MainAppScaffold, який забезпечує єдиний макет із заголовком, кнопкою повернення (за потреби) та нижньою навігаційною панеллю. AuthenticatedRoute захищає доступ до екранів, доступних лише авторизованим користувачам.

Основні екрани включають:

- LoginScreen і SignUpScreen (автентифікація та реєстрація);
- FeedScreen (стрічка контенту з можливістю переходу до профілю іншого користувача);
- FriendsScreen (список друзів);

- ProfileScreen (профіль користувача);
- FiltersScreen (налаштування фільтрів для стрічки).

Такий підхід забезпечує модульність, ізоляцію екранів та єдиний стиль інтерфейсу, що полегшує розробку та подальшу підтримку додатка "MateUp".

3.1.4 Реалізація сторінок автентифікації

Функціонал автентифікації є основою сучасних мобільних додатків, забезпечуючи безпечний доступ користувачів до персоналізованих даних. У проєкті "MateUp" для Android цей блок реалізовано через два взаємопов'язані екрани: LoginScreen і SignUpScreen, які побудовані з використанням Jetpack Compose для модульності та зручності розробки. Кожен екран інкапсулює специфічну логіку входу або реєстрації, використовуючи композицію компонентів і відокремлюючи візуальний дизайн від бізнес-логіки.

Екрани автентифікації базуються на компоненті AuthenticationLayout, який задає єдиний стиль із фоновим градієнтом та контейнером для вмісту. LoginScreen (код наведено у додатку А) відображає форму LoginFormFields у межах AuthenticationLayout, а SignUpScreen (код наведено у додатку А) - форму SignUpFormFields. Такий підхід забезпечує узгоджений дизайн і розподіл відповідальностей: екрани керують макетом, тоді як форми обробляють введення даних і валідацію.

LoginScreen відображає форму з полями для email і пароля, кнопкою "Forgot Password?" та соціальними входами (наприклад, Google). Логіка валідації та автентифікації реалізовано у LoginViewModel, який використовує LoginUseCase для обробки запитів до сервера та зберігає токени через TokenManager. SignUpScreen додає поле для імені користувача та опцію прийняття умов, з логікою в SignUpViewModel і UserSignUpUseCase. Обидва екрани відображають індикатор завантаження (CircularProgressIndicator) під час обробки запитів і підтримують перехід між собою через відповідні колбеки (onNavigateToSignUp, onNavigateToLogin).

Такий підхід забезпечує модульність, узгоджений дизайн і безпечну обробку даних автентифікації в додатку "MateUp".

3.1.5 Реалізація сторінки керування профілем користувача

Сторінка профілю в додатку знайомств "MateUp" для Android є складною системою для створення та редагування персональної інформації користувача. Архітектура побудована з використанням Jetpack Compose та ViewModel, з валідацією даних через утиліти (UserValidator, DateValidator) та асинхронною обробкою API-запитів. Система автоматично визначає режим роботи: при першому відвідуванні відображається форма для заповнення базових даних, а при повторних - дані завантажуються з сервера через GetUserProfileUseCase. Інтерфейс розроблено з урахуванням сучасних принципів UX/UI, використовуючи Material Design 3 компоненти для створення елегантного та зручного досвіду.

3.1.5.1 Екран ProfileScreen

Екран ProfileScreen (код наведено у додатку А) реалізує повний цикл управління профілем із валідацією та збереженням даних. Він використовує ProfileViewModel для обробки стану, CountryViewModel для вибору країни та InterestsViewModel для інтересів. Екран підтримує завантаження фото через pickImageLauncher, відображення прогресу (CircularProgressIndicator) та сповіщення через SnackbarHostState. Форма включає поля для імені, прізвища, дати народження, країни, міста, біографії, статі та інтересів, із умовним рендерингом залежно від стану завантаження чи збереження.

ProfileViewModel (код частково наведено у додатку А) керує станом профілю (ProfileUiState), завантажуючи дані через GetUserProfileUseCase і зберігаючи їх через CreateUserProfileUseCase або UpdateUserProfileUseCase. Він підтримує валідацію полів (наприклад, UserValidator.validateName) та обробку фото через UploadPhotoUseCase. Логіка включає асинхронне завантаження профілю, вибір дати народження через DatePickerState та оновлення інтересів.

Така реалізація забезпечує гнучкість, безпеку та зручність редагування профілю, адаптуючись до стану користувача та підтримуючи сучасний дизайн.

3.1.6 Реалізація головної сторінки

Головна сторінка додатка "MateUp" для Android представлена екраном FeedScreen, який є інтерактивним каталогом профілів користувачів із

можливостями фільтрації та персоналізації. Користувачі можуть переглядати профілі у вигляді карток, застосовувати фільтри (реалізовані на окремому екрані `FiltersScreen`) та оновлювати список через механізм "pull-to-refresh". Дизайн виконано з використанням `Material Design 3`, із градієнтним фоном карток для створення романтичної атмосфери та білого тла для контенту, що забезпечує елегантну рамку.

3.1.6.1 Екран `FeedScreen`

Основний компонент `FeedScreen` (код наведено у лістингу 3.2) є центральним елементом системи знайомств, реалізуючи управління станом через `FeedViewModel`. Компонент відображає список профілів, отриманих із сервера, із підтримкою асинхронного завантаження (`CircularProgressIndicator` під час `isLoading`). Використовується `FeedPullToRefreshBox` для оновлення даних через жест "pull-to-refresh" та передачі колбеку `onUserCardClick` для переходу до профілю іншого користувача.

Лістинг 3.2 – Код екрану `FeedScreen`

```
@Composable
fun FeedScreen(
    modifier: Modifier = Modifier,
    feedViewModel: FeedViewModel = hiltViewModel(),
    onUserCardClick: (user: FeedUser) -> Unit,
) {
    val feedUiState by
    feedViewModel.uiState.collectAsStateWithLifecycle()

    if (feedUiState.isLoading) {
        Box(modifier = Modifier.fillMaxSize(), contentAlignment =
        Alignment.Center) {
            CircularProgressIndicator(color = Coral)
        }
    } else {
        FeedPullToRefreshBox(
            users = feedUiState.users,
            onRefresh = { feedViewModel.fetchFeed() },
            modifier = modifier,
            onUserCardClick = onUserCardClick,
        )
    }
}

@OptIn(ExperimentalMaterial3Api::class)
```

```

@Composable
fun FeedPullToRefreshBox(
    users: List<FeedUser>,
    isRefreshing: Boolean = false,
    onRefresh: () -> Unit,
    onUserCardClick: (user: FeedUser) -> Unit,
    modifier: Modifier = Modifier,
) {
    PullToRefreshBox(
        isRefreshing = isRefreshing,
        onRefresh = onRefresh,
        modifier = modifier,
    ) {
        Column(modifier = Modifier.fillMaxSize()) {
            FeedList(users = users, onUserCardClick =
onUserCardClick)
        }
    }
}

```

3.1.6.2 Список профілів FeedList

Компонент FeedList (код наведено у лістингу 3.3) відповідає за відображення профілів у вигляді сітки. Використовується LazyVerticalGrid із двома стовпцями (gridCellsSize = 2) та відступами 8.dp між картками. Кожна картка генерується через items, із унікальним ключем на основі user.id, що забезпечує ефективне оновлення при змінах даних.

Лістинг 3.3 – Код компонента FeedList

```

@Composable
fun FeedList(
    users: List<FeedUser>,
    contentPadding: PaddingValues = PaddingValues(16.dp),
    onUserCardClick: (user: FeedUser) -> Unit,
    modifier: Modifier = Modifier,
) {
    val gridCellsSize = 2
    val gridItemsSpacing = 8.dp

    LazyVerticalGrid(
        columns = GridCells.Fixed(gridCellsSize),
        horizontalArrangement =
Arrangement.spacedBy(gridItemsSpacing),
        verticalArrangement =
Arrangement.spacedBy(gridItemsSpacing),
        contentPadding = contentPadding,
        modifier = modifier,
    ) {
        items(
            users.size,
            key = { index, user -> user.id },
        ) { index, user ->
            FeedUserCard(
                user = user,
                onUserCardClick = onUserCardClick,
            )
        }
    }
}

```

```

        content = {
            items(users, key = { user -> user.id }) { user ->
                FeedCard(user = user, onUserCardClick =
onUserCardClick)
            }
        }
    )
}

```

3.1.6.3 Компонент картки FeedCard

Компонент FeedCard (код наведено у додатку А) відображає профіль користувача у вигляді картки. Картка побудована з використанням Material 3 Card, із співвідношенням сторін 1:1.5 та заокругленими кутами (RoundedCornerShape(8.dp)). Фото профілю завантажується через AsyncImage із підтримкою кросфейду. Градієнтний оверлей (Brush.getVerticalGradient) створює напівпрозорий ефект у нижній частині, де відображаються ім'я, вік (обчислений через DateUtils.calculateUserAgeWithDate), країна та інтереси користувача (FeedInterestsList).

3.1.6.4 Екран фільтрації FiltersScreen

Екран FiltersScreen (код наведено у додатку А) реалізує механізм персоналізації пошуку профілів, дозволяючи користувачам налаштовувати віковий діапазон через слайдер (FilterAgeRangeSlider) та обирати інтереси за допомогою FilterInterestsRow. Дані фільтрів отримуються з сервера через GetFiltersUseCase і зберігаються через UpdateFiltersUseCase, із станом, керованим FiltersViewModel (код наведено у додатку А). Екран підтримує асинхронне завантаження (CircularProgressIndicator) та відображає помилки через filtersError. Користувач може застосувати налаштування (onApplySuccess) або скасувати їх (onCancelClick) за допомогою FilterActionButtonRow.

3.1.7 Реалізація сторінки профілю інших користувачів

Екран профілю інших користувачів у додатку "MateUp" слугує центральним хабом для перегляду детальної інформації про потенційних співрозмовників чи друзів, а також для ініціювання соціальної взаємодії. Користувачі можуть ознайомитися з біографією, інтересами та особистими даними, оцінити сумісність

на основі спільних захоплень і розпочати спілкування чи додати користувача до друзів. Інтеграція кнопок для чату та додавання в друзі забезпечує безперервний користувацький досвід, дозволяючи миттєво переходити від перегляду профілю до активної взаємодії, що підвищує залученість і соціальну конверсію.

3.1.7.1 Екран FeedUserProfileScreen

Екран FeedUserProfileScreen (код наведено у лістингу 3.4) є основним компонентом для відображення профілю іншого користувача, отриманого з фіду. Він передає дані профілю до дочірнього компонента FeedUserProfileCard і викликає FeedUserProfileViewModel для обробки запиту на додавання в друзі через requestFriendship. Екран підтримує прокрутку через verticalScroll та використовує біле тло (OffWhite) для створення елегантного дизайну.

Лістинг 3.4 – Код екрану FeedUserProfileScreen

```
@Composable
fun FeedUserProfileScreen(
    user: FeedUser,
    feedUserProfileViewModel: FeedUserProfileViewModel =
hiltViewModel(),
    modifier: Modifier = Modifier,
) {
    Column(
        modifier = modifier
            .fillMaxSize()
            .background(OffWhite)
            .verticalScroll(rememberScrollState())
    ) {
        FeedUserProfileCard(
            profile = user.profile,
            onChatClick = {},
            onAddFriendClick = {
                feedUserProfileViewModel.requestFriendship(user.id)
            }
        )
    }
}
```

3.1.7.2 Компонент профілю FeedUserProfileCard

Компонент FeedUserProfileCard (код наведено у додатку A) відповідає за детальне відображення профілю користувача. Він включає фото профілю (завантажене через AsyncImage) або ініціал імені як заглушку, із градієнтним

оверлеєм (Brush.getVerticalGradient) для створення візуального ефекту. Нижня частина картки має заокруглені кути (RoundedCornerShape) і відображає біографію (bio) та інтереси (FeedInterestsList). Кнопки для чату та додавання в друзі (ProfileActionButton) дозволяють ініціювати взаємодію, із підтримкою асинхронної обробки через FeedUserProfileViewModel.

3.1.7.3 ViewModel для обробки взаємодій

FeedUserProfileViewModel (код наведено у додатку A) відповідає за обробку запиту на додавання в друзі через RequestFriendshipUseCase. ViewModel керує станом через FeedUserProfileUiState, із підтримкою асинхронного виконання (viewModelScope) та обробкою помилок (feedUserProfileError). Дані передаються до сервера з використанням Bearer-токена, отриманого через TokenManager.

3.1.8 Реалізація сторінки друзів

Екран друзів у додатку "MateUp" створений для централізованого управління соціальними зв'язками користувача, надаючи можливість переглядати список поточних друзів, надіслані та отримані запити на дружбу в єдиному інтерфейсі. Ця функціональність дозволяє користувачам підтримувати свою соціальну мережу, контролювати доступ до контенту та ефективно керувати новими знайомствами через систему запитів на дружбу з можливістю їх прийняття або відхилення.

3.1.8.1 Екран FriendsScreen

Екран FriendsScreen (код наведено у додатку A) є основним компонентом для управління друзями та запитами на дружбу. Він реалізує табову навігацію через TabRowComponent із трьома вкладками: "Friends", "Sent" та "Received". Компонент підтримує пошук через SearchTextField, із фільтрацією за введеним текстом (searchValue), та асинхронне завантаження даних (DataLoadingProgressIndicator). Дані про друзів і запити завантажуються через FriendsViewModel, який викликає відповідні UseCase.

3.1.8.2 Компонент для відображення друзів UserFriendsScreen

Компонент UserFriendsScreen (код наведено у лістингу 3.5) відображає список поточних друзів у вигляді карток (FriendCard). Кожна картка містить кнопки для швидкого переходу до чату та контекстне меню (MoreActionMenu), яке


```

onDismissMenuRequest,
                                onDismissRequest =
                                ) } } } } } } } } } }

```

3.1.8.3 Компонент для відображення надісланих запитів SentRequestsScreen

Компонент `SentRequestsScreen` (код наведено у лістингу 3.6) відображає список надісланих запитів на дружбу. Кожна картка (`FriendCard`) містить кнопку для скасування запиту (`ActionIconButton` із іконкою `DeleteForever`). Логіка скасування ще не реалізована, але передбачається виклик відповідного `UseCase` для видалення запиту.

Лістинг 3.6 – Код компонента `SentRequestsScreen`

```

@Composable
fun SentRequestsScreen(
    sentRequests: List<SentFriendshipRequestEntity>,
    modifier: Modifier = Modifier
) {
    Column(
        modifier = modifier
            .fillMaxSize()
            .verticalScroll(rememberScrollState()),
    ) {
        sentRequests.forEachIndexed { index, request ->
            FriendCard(
                friendProfile = request.profile,
                trailingContent = {
                    Row(horizontalArrangement =
Arrangement.spacedBy(8.dp)) {
                        ActionIconButton(
                            onClick = {},
                            imageVector =
Icons.Default.DeleteForever,
                            contentDescription = "Delete Request
Icon"
                        )
                    }
                }
            )
        }
    }
}

```

3.1.8.4 Компонент для відображення отриманих запитів ReceivedRequestsScreen

Компонент ReceivedRequestsScreen (код наведено у лістингу 3.7) відображає список отриманих запитів на дружбу. Кожна картка (FriendCard) містить кнопки для прийняття (CheckCircle) або відхилення (Cancel) запиту. Логіка обробки запитів ще не реалізована, але передбачається виклик UseCase для асинхронного оновлення статусу запиту.

Лістинг 3.7 – Код компонента ReceivedRequestsScreen

```
@Composable
fun ReceivedRequestsScreen(
    receivedRequests: List<ReceivedFriendshipRequestEntity>,
    modifier: Modifier = Modifier
) {
    Column(
        modifier = modifier
            .fillMaxSize()
            .verticalScroll(rememberScrollState()),
    ) {
        receivedRequests.forEachIndexed { index, request ->
            FriendCard(
                friendProfile = request.profile,
                trailingContent = {
                    Row(horizontalArrangement =
Arrangement.spacedBy(8.dp)) {
                        ActionIconButton(
                            onClick = {},
                            imageVector = Icons.Default.Cancel,
                            contentDescription = "Cancel Icon"
                        )
                        ActionIconButton(
                            onClick = {},
                            imageVector = Icons.Default.CheckCircle,
                            contentDescription = "Check Circle Icon"
                        )
                    }
                }
            )
        }
    }
}
```

3.1.8.5 ViewModel для управління друзями

FriendsViewModel (код частково наведено у додатку A) керує станом екрану через FriendsUiState, завантажуючи дані про друзів (GetFriendsUseCase), надіслані

(GetSentFriendshipRequestsUseCase) та отримані запити (GetReceivedFriendshipRequestsUseCase). Він підтримує табову навігацію (onSelectTabIndex), пошук (onSearchValueChange) та контекстне меню (onOpenMenuRequest, onDismissMenuRequest). Дані завантажуються асинхронно з сервера з використанням Bearer-токена.

3.2 Розробка серверної частини системи

3.2.1 Моделювання бази даних

Для забезпечення функціональності комп'ютерної системи "MateUp", розроблена база даних для зберігання даних користувачів, їх профілю, друзів, відправлених запитів на дружбу та налаштування фільтрів для пошуку друзів, які отримуються з серверу системи. Структура бази даних реалізована за допомогою TypeORM із підтримкою реляційних зв'язків між таблицями.

Діаграма структури бази даних наведена на рисунку 3.2.

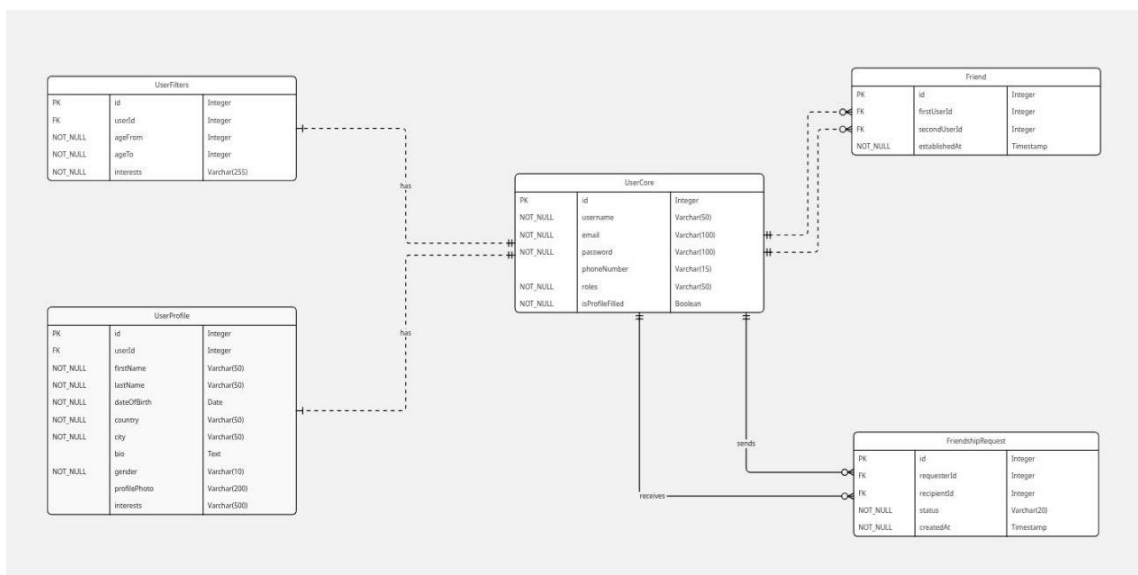


Рисунок 3.2 – ER-модель бази даних

ER-модель бази даних включає наступні таблиці:

- користувачів (UserCore);
- фільтри користувачів (UserFilters);
- профіль користувача (UserProfile);
- запити на дружбу (FriendshipRequest);
- друзі (Friends).

Структура кожної таблиці зображена в таблицях 3.1-3.5.

Таблиця 3.1 - Склад таблиці UserCore

Атрибут	Тип даних	Допустимі значення	Значення за замовчуванням	Обов'язковість	Примітки
id	UUID	Унікальний UUID	-	Так	Первинний ключ, автоінкремент
username	varchar(255)	Латиниця	-	Так	Унікальне значення
email	varchar	Латиниця	-	Так	Унікальне значення
password	varchar	Латиниця, кирилиця	-	Так	-
phoneNumber	varchar(15)	Латиниця	-	Ні	-
roles	array(enum: Role)	Значення з Role	[Role.USER]	Так	-
isProfileFilled	boolean	true/false	false	Так	-
filters	OneToOne	UserFilters	-	Ні	Каскадне видалення
profile	OneToOne	UserProfile	-	Ні	Каскадне видалення
sentFriendshipRequests	OneToMany	FriendshipRequest[]	-	Ні	Каскадне видалення
receivedFriendshipRequests	OneToMany	FriendshipRequest[]	-	Ні	Каскадне видалення
friends	ManyToMany	Friend[]	-	Ні	Через таблицю зв'язку

Таблиця users_core (код наведено у лістингу 3.8) зберігає основну інформацію про користувачів. Вона містить унікальні поля username та email, пароль, додаткові

дані, такі як номер телефону та ролі (roles), а також зв'язки з профілем (UserProfile) та фільтрами (UserFilters). Первинний ключ id генерується як UUID.

Лістинг 3.8 – Код сутності UserCore

```
@Entity("users_core")
@Unique(["username", "email"])
export class UserCore extends BaseEntity {
  @Index("users_id_idx")
  @PrimaryGeneratedColumn("uuid")
  id: string;

  @Column({ type: "varchar", nullable: false, length: 255 })
  username: string;

  @Column({ type: "varchar", nullable: false })
  email: string;

  @Column({ type: "varchar", nullable: false })
  password: string;

  @Column({ type: "varchar", nullable: true, length: 15 })
  phoneNumber: string;

  @Column({ type: "enum", enum: Role, array: true, nullable: false,
default: [Role.USER] })
  roles: Role[];

  @Column({ type: "boolean", nullable: false, default: false })
  isProfileFilled: boolean;

  @OneToOne(() => UserFilters, (filters: UserFilters) =>
filters.user, { cascade: true, onDelete: "CASCADE" })
  filters: UserFilters;

  @OneToOne(() => UserProfile, (profile) => profile.user, { cascade:
true, onDelete: "CASCADE" })
  profile: UserProfile;

  @OneToMany(() => FriendshipRequest, (friendship) =>
friendship.requester, { onDelete: "CASCADE" })
  sentFriendshipRequests: FriendshipRequest[];

  @OneToMany(() => FriendshipRequest, (friendship) =>
friendship.recipient, { onDelete: "CASCADE" })
  receivedFriendshipRequests: FriendshipRequest[];

  @ManyToMany(() => Friend, (friend: Friend): UserCore[] =>
friend.users)
  @JoinTable()
  friends: Friend[];}
```

Таблиця 3.2 - Склад таблиці UserFilters

Атрибут	Тип даних	Допустимі значення	Значення за замовчуванням	Обов'язковість	Примітки
id	UUID	Унікальний UUID	-	Так	Первинний ключ, автоінкремент
user	OneToOne	UserCore	-	Так	Зовнішній ключ, каскадне видалення
ageFrom	int	Натуральне число	-	Так	-
ageTo	int	Натуральне число	-	Так	-
interests	array(enum: Interest)	Значення з Interest	-	Ні	-

Сутність UserFilters (код наведено у лістингу 3.9) зберігає налаштування фільтрів користувача для персоналізації фіду в додатку "MateUp". Вона пов'язана з UserCore через відношення один до одного.

Лістинг 3.9 – Код сутності UserFilters

```

@Entity("user_filters")
export class UserFilters extends BaseEntity {
    @PrimaryGeneratedColumn("uuid")
    id: string;

    @OneToOne(() => UserCore, (user: UserCore): string => user.id, {
        onDelete: "CASCADE",
    })
    @JoinColumn({ name: "userId" })
    user: UserCore;

    @Column({ type: "int", nullable: false })
    ageFrom: number;

    @Column({ type: "int", nullable: false })
    ageTo: number;

    @Column({ type: "enum", array: true, enum: Interest, nullable:
false })
    interests: Interest[];
}

```

Таблиця 3.3 - Склад таблиці UserProfile

Атрибут	Тип даних	Допустимі значення	Значення за замовчуванням	Обов'язковість	Примітки
id	UUID	Унікальний UUID	-	Так	Первинний ключ, автоінкремент
user	OneToOne	UserCore	-	Так	Зовнішній ключ, каскадне видалення
firstName	varchar(255)	Латиниця, кирилиця	-	Так	-
lastName	varchar(255)	Латиниця, кирилиця	-	Так	-
dateOfBirth	date	-	-	Ні	-
country	varchar(255)	Латиниця, кирилиця	-	Ні	-
city	varchar(255)	Латиниця, кирилиця	-	Ні	-
bio	varchar(300)	Латиниця, кирилиця	-	Так	-
gender	enum: Gender	Значення з Gender	-	Так	-
profilePhoto	varchar(255)	URL (латиниця)	-	Ні	URL фото
interests	array(enum: Interest)	Значення з Interest	-	Ні	-

Сутність UserProfile (код наведено у лістингу 3.10) зберігає додаткову інформацію про профіль користувача, включаючи особисті дані та інтереси.

Лістинг 3.10 – Код сутності UserProfile

```

@Entity("users_profile")
export class UserProfile extends BaseEntity {
    @PrimaryGeneratedColumn("uuid")
    id: string;

    @OneToOne(() => UserCore, (user) => user.id, {
        onDelete: "CASCADE",
    })
    @JoinColumn({ name: "userId" })
    user: UserCore;

    @Column({
        type: "varchar",
        length: 255,
        nullable: false,
    })

```

```

    })
    firstName: string;

    @Column({
        type: "varchar",
        length: 255,
        nullable: false,
    })
    lastName: string;

    @Column({
        type: "date",
        nullable: true,
    })
    dateOfBirth: Date;

    @Column({
        type: "varchar",
        length: 255,
        nullable: true,
    })
    country: string;

    @Column({
        type: "varchar",
        length: 255,
        nullable: true,
    })
    city: string;

    @Column({
        type: "varchar",
        length: 300,
        nullable: false,
    })
    bio: string;

    @Column({
        type: "enum",
        enum: Gender,
        nullable: false,
    })
    gender: Gender;

    @Column({ type: "varchar", length: 255, nullable: true })
    profilePhoto?: string;

    @Column({ type: "enum", enum: Interest, array: true, nullable:
true })
    interests?: Interest[];
}

```

Таблиця 3.4 - Склад таблиці FriendshipRequest

Атрибут	Тип даних	Допустимі значення	Значення за замовчуванням	Обов'язковість	Примітки
id	UUID	Унікальний UUID	-	Так	Первинний ключ, автоінкремент
requesterId	UUID	Унікальний UUID	-	Так	Зовнішній ключ, індексований
requester	ManyToOne	UserCore	-	Так	Каскадне видалення
recipientId	UUID	Унікальний UUID	-	Так	Зовнішній ключ, індексований
recipient	ManyToOne	UserCore	-	Так	Каскадне видалення
status	enum: FriendshipStatus	Значення з FriendshipStatus	PENDING	Так	-
createdAt	timestamp	-	Поточна дата/час	Так	Автоматично генерується

Сутність FriendshipRequest (код наведено у лістингу 3.11) зберігає інформацію про запити на дружбу між користувачами.

Лістинг 3.11 – Код сутності FriendshipRequest

```

@Entity("friendship_requests")
export class FriendshipRequest extends BaseEntity {
    @PrimaryGeneratedColumn("uuid")
    id: string;

    @Index("friendship_request_requester_id_idx")
    @Column({ type: "uuid", nullable: false })
    requesterId: string;

    @ManyToOne(() => UserCore, (userCore) =>
        userCore.sentFriendshipRequests, {
            onDelete: "CASCADE",
        })
    @JoinColumn({ name: "requesterId" })
    requester: UserCore;

    @Index("friendship_request_recipient_id_idx")
    @Column({ type: "uuid", nullable: false })
    recipientId: string;

    @ManyToOne(
        () => UserCore,

```

```

    (userCore) => userCore.receivedFriendshipRequests,
    {
        onDelete: "CASCADE",
    },
)
@JoinColumn({ name: "recipientId" })
recipient: UserCore;

@Column({
    type: "enum",
    enum: FriendshipStatus,
    default: FriendshipStatus.PENDING,
})
status: FriendshipStatus;

@CreateDateColumn()
createdAt: Date;
}

```

Таблиця 3.5 - Склад таблиці Friend

Атрибут	Тип даних	Допустимі значення	Значення за замовчуванням	Обов'язковість	Примітки
id	UUID	Унікальний UUID	-	Так	Первинний ключ, автоінкремент
firstUserId	UUID	Унікальний UUID	-	Так	Зовнішній ключ, індексований
firstUser	ManyToOne	→ UserCore	-	Так	Каскадне видалення
secondUserId	UUID	Унікальний UUID	-	Так	Зовнішній ключ, індексований
secondUser	ManyToOne	→ UserCore	-	Так	Каскадне видалення
users	ManyToMany	→ UserCore[]	-	Ні	Через таблицю зв'язку
establishedAt	timestamp	-	Поточна дата/час	Так	Автоматично генерується

Сутність Friend (код наведено у лістингу 3.12) зберігає інформацію про встановлені дружні зв'язки між користувачами.

Лістинг 3.12 – Код сутності Friend

```

@Entity("friends")
@Check(`"firstUserId" < "secondUserId"`)
export class Friend extends BaseEntity {
    @PrimaryGeneratedColumn("uuid")
    readonly id: string;
}

```

```

@Index("first_user_id_friend_idx")
@Column({
    type: "uuid",
    nullable: false,
})
firstUserId: string;

@ManyToOne(() => UserCore, {
    onDelete: "CASCADE",
    nullable: false,
})
@JoinColumn({ name: "firstUserId", referencedColumnName: "id" })
readonly firstUser: UserCore;

@Index("second_user_id_friend_idx")
@Column({
    type: "uuid",
    nullable: false,
})
secondUserId: string;

@ManyToOne(() => UserCore, {
    onDelete: "CASCADE",
    nullable: false,
})
@JoinColumn({ name: "secondUserId", referencedColumnName: "id" })
readonly secondUser: UserCore;

@ManyToMany(() => UserCore, (user: UserCore) => user.friends)
readonly users: UserCore[];

@CreateDateColumn()
readonly establishedAt: Date;

@BeforeInsert()
ensureCanonicalOrder(): void {
    if (this.firstUserId > this.secondUserId) {
        const temp: string = this.firstUserId;
        this.firstUserId = this.secondUserId;
        this.secondUserId = temp;
    }
}
}

```

3.2.2 Реалізація модуля автентифікації

Модуль автентифікації забезпечує базові операції входу, реєстрації, отримання даних користувача, виходу та оновлення токенів для додатка "MateUp". Він реалізовано через контролер AuthController, який використовує сервіс

AuthService для обробки логіки бізнесу. Автентифікація базується на JWT (JSON Web Token) із підтримкою оновлення tokenів через RefreshAuthGuard, а доступ до ендпоінтів регулюється за допомогою анотацій @Public() та @UseGuards().

3.2.2.1 Контролер автентифікації (AuthController)

Контролер AuthController (код наведено у додатку А) визначений із тегом @ApiTags("auth") і маршрутом /auth. Він включає п'ять основних ендпоінтів для управління автентифікацією:

- логін (/login);
- реєстрація (/sign-up);
- отримання даних користувача (/user);
- вихід (/logout);
- оновлення tokenів (/refresh).

/login має публічний ендпоінт (@Public()) приймає дані користувача через UserLoginDto і повертає LoginResponse із токенами. Використовує код стану HTTP 200. /sign-up має публічний ендпоінт із гвардом DoesUserExistGuard для перевірки унікальності користувача. Приймає UserSignUpDto і повертає SignUpResponse. /user має захищений ендпоінт із JwtAuthGuard, повертає об'єкт UserCore для авторизованого користувача за його userId. /logout має захищений ендпоінт із JwtAuthGuard, виконує вихід користувача за userId і повертає повідомлення про успішний вихід. /refresh має публічний ендпоінт із RefreshAuthGuard, приймає refreshToken і повертає нові Tokens на основі JwtPayload

3.2.3 Реалізація модуля профілю користувача

Модуль профілю користувача забезпечує операції створення, отримання та оновлення профілю авторизованого користувача в додатку "MateUp". Він реалізовано через контролер ProfileController, який використовує сервіс ProfileService для обробки бізнес-логіки. Усі ендпоінти захищені авторизацією через JwtAuthGuard, що гарантує доступ лише для авторизованих користувачів.

3.2.3.1 Контролер профілю (ProfileController)

Контролер ProfileController (код наведено у лістингу 3.13) визначений із тегом

@ApiTags("Profile") і маршрутом /profile. Він включає три основні ендпоінти для управління профілем користувача:

- створення профілю (/create-profile);
- отримання профілю (/get-profile);
- оновлення профілю (/update-profile).

/create-profile має ендпоінт із методом POST приймає дані профілю через CreateProfileDto та ідентифікатор користувача (userId), повертаючи створений об'єкт UserProfile. Використовує код стану HTTP 201.

/get-profile має ендпоінт із методом GET повертає профіль користувача (UserProfile) за його userId. Використовує код стану HTTP 200.

/update-profile має ендпоінт із методом PUT оновлює профіль користувача через UpdateProfileDto і повертає оновлений об'єкт UserProfile. Використовує код стану HTTP 200.

Лістинг 3.13 – Код контролера ProfileController

```
@ApiTags("Profile")
@Controller("profile")
@UseGuards(JwtAuthGuard)
export class ProfileController {
  constructor(private readonly profileService: ProfileService) {}

  @ApiBearerAuth()
  @Post("/create-profile")
  @HttpCode(HttpStatus.CREATED)
  async createProfile(
    @GetCurrentUserId() userId: string,
    @Body() createProfileDto: CreateProfileDto,
  ): Promise<UserProfile> {
    return await this.profileService.createProfile(userId,
createProfileDto);
  }

  @ApiBearerAuth()
  @Get("/get-profile")
  @HttpCode(HttpStatus.OK)
  async getProfile(@GetCurrentUserId() userId: string):
Promise<UserProfile> {
    return await this.profileService.getProfile(userId);
  }

  @ApiBearerAuth()
```

```

@Put("/update-profile")
@HttpCode(HttpStatus.OK)
async updateProfile(
    @GetCurrentUserId() userId: string,
    @Body() updateProfileDto: UpdateProfileDto,
): Promise<UserProfile> {
    return await this.profileService.updateProfile(userId,
updateProfileDto);
}
}

```

3.2.4 Реалізація модуля управління файлами

Модуль управління файлами забезпечує функціонал завантаження, отримання, перегляду списку та видалення фотографій профілю користувачів у додатку "MateUp". Він реалізовано через контролер `FileController`, який використовує сервіс `FileService` для обробки бізнес-логіки, а також `WinstonService` для логування помилок. Усі ендпоінти потребують авторизації через JWT, що забезпечується через `@ApiBearerAuth()`.

3.2.4.1 Контролер файлів (`FileController`)

Контролер `FileController` (код наведено у додатку A) визначений із тегом `@ApiTags("Files")` і маршрутом `/file`. Він включає чотири основні ендпоінти для роботи з файлами:

- завантаження фотографії (POST `/file`);
- отримання фотографії GET `/file/:filename`;
- отримання списку фотографій GET `/file`;
- видалення фотографії DELETE `/file/:filename`.

POST `/file` має ендпоінт із методом POST дозволяє завантажувати фото профілю. Використовує `FileInterceptor` для обробки файлу та `ParseFilePipeBuilder` для валідації типу файлу (`VALID_UPLOADS_MIME_TYPES`) та розміру (`MAX_PICTURE_SIZE_IN_BYTES`). Повертає результат із кодом стану HTTP 201.

GET `/file/:filename` має ендпоінт із методом GET повертає потік даних (stream) фотографії за її ім'ям (filename) для авторизованого користувача. Використовує код стану HTTP 200.

GET /file має ендпоінт із методом GET повертає список метаданих (FileDetails[]) усіх фотографій користувача. Використовує код стану HTTP 200.

DELETE /file/:filename має ендпоінт із методом DELETE видаляє фотографію за її ім'ям (filename). Повертає результат із кодом стану HTTP 200.

3.2.5 Реалізація модуля управління головної сторінки

Модуль управління головної сторінки забезпечує функціонал отримання персоналізованого списку користувачів для авторизованих користувачів у додатку "MateUp". Він реалізовано через контролер FeedController, який використовує сервіс FeedService для обробки бізнес-логіки. Ендпоінт потребує авторизації через JWT, що забезпечується через декоратор @UseGuards(JwtAuthGuard) та @ApiBearerAuth().

3.2.5.1 Контролер FeedController

Контролер FeedController (код наведено у лістингу 3.14) визначений із тегом @ApiTags("feed") і маршрутом /feed. Він включає один основний ендпоінт для роботи з фідом: отримання рек (POST /feed) має ендпоінт із методом POST повертає список користувачів (UserCore[]) для авторизованого користувача на основі його ідентифікатора (userId). Використовує код стану HTTP 200.

Лістинг 3.14 – Код контролера FeedController

```
@ApiTags("feed")
@Controller("feed")
@UseGuards(JwtAuthGuard)
export class FeedController {
  constructor(private readonly feedService: FeedService) {}

  @ApiBearerAuth()
  @HttpCode(HttpStatus.OK)
  @Post()
  async getFeed(@GetCurrentUserId() userId: string):
    Promise<UserCore[]> {
    try {
      const response = await this.feedService.getFeed(userId);
      return response;
    } catch (error) {
      throw new HttpException(
        {
          type: ResponseType.ERROR,
```

```

        message: error.message || "Error occurred while retrieving
feed",
        data: null,
    },
    HttpStatus.INTERNAL_SERVER_ERROR,
);
}
}
}

```

3.2.6 Реалізація модуля управління фільтрами

Модуль управління фільтрами забезпечує функціонал отримання та оновлення фільтрів для персоналізації фіду користувачів у додатку "MateUp". Він реалізовано через контролер `FiltersController`, який використовує сервіс `FiltersService` для обробки бізнес-логіки та `WinstonService` для логування помилок. Усі ендпоінти потребують авторизації через JWT, що забезпечується через декоратор `@UseGuards(JwtAuthGuard)` та `@ApiBearerAuth()`.

3.2.6.1 Контролер фільтрів (`FiltersController`)

Контролер `FiltersController` (код наведено у додатку А) визначений із тегом `@ApiTags("filters")` і маршрутом `/filters`. Він включає два основні ендпоінти для роботи з фільтрами:

- отримання фільтрів (GET `/filters/get-filters`);
- оновлення фільтрів (PUT `/filters/update-filters`).

GET `/filters/get-filters` має ендпоінт із методом GET повертає налаштування фільтрів користувача (`UserFilters`) на основі його ідентифікатора (`userId`). Використовує код стану HTTP 200.

PUT `/filters/update-filters` має ендпоінт із методом PUT дозволяє оновлювати налаштування фільтрів користувача на основі даних, переданих у тілі запиту (`UpdateFiltersDto`). Повертає оновлені фільтри (`UserFilters`) із кодом стану HTTP 200.

3.2.7 Реалізація модуля управління друзями

Модуль управління друзями забезпечує функціонал управління дружніми зв'язками в додатку "MateUp", включаючи отримання списку друзів, надсилання та обробку запитів на дружбу, а також видалення друзів. Він реалізовано через

контролер `FriendsController`, який використовує сервіси `FriendsService` та `FriendshipRequestService` для обробки бізнес-логіки. Усі ендпоінти потребують авторизації через JWT, що забезпечується через декоратор `@UseGuards(JwtAuthGuard)` та `@ApiBearerAuth()`.

3.2.7.1 Контролер друзів (`FriendsController`)

Контролер `FriendsController` (код наведено у додатку А) визначений із тегом `@ApiTags("friends")` і маршрутом `/friends`. Він включає вісім основних ендпоінтів для роботи з дружніми зв'язками:

- отримання списку друзів (GET `/friends`);
- отримання надісланих запитів на дружбу (GET `/friends/sent-requests`);
- отримання отриманих запитів на дружбу (GET `/friends/received-requests`);
- надсилання запиту на дружбу (POST `/friends/request-friendship/:recipientId`);
- прийняття запиту на дружбу (POST `/friends/accept-friendship/:recipientId`);
- відхилення запиту на дружбу (POST `/friends/reject-friendship/:recipientId`);
- видалення друга (DELETE `/friends/:friendId`);
- видалення всіх друзів (DELETE `/friends`).

GET `/friends` повертає список друзів користувача (`UserCore[]`) на основі його ідентифікатора (`userId`). Використовує код стану HTTP 200.

GET `/friends/sent-requests` повертає список користувачів (`UserCore[]`), яким поточний користувач надіслав запити на дружбу. Використовує код стану HTTP 200.

GET `/friends/received-requests` повертає список користувачів (`UserCore[]`), від яких поточний користувач отримав запити на дружбу. Використовує код стану HTTP 200.

POST `/friends/request-friendship/:recipientId` дозволяє надіслати запит на дружбу іншому користувачу за його ідентифікатором (`recipientId`). Повертає результат (`RequestFriendshipResponse`) із кодом стану HTTP 200.

POST `/friends/accept-friendship/:recipientId` дозволяє прийняти запит на дружбу від іншого користувача за його ідентифікатором (`recipientId`). Повертає результат (`AcceptFriendshipResponse`) із кодом стану HTTP 200.

POST /friends/reject-friendship/:recipientId дозволяє відхилити запит на дружбу від іншого користувача за його ідентифікатором (recipientId). Повертає результат (AcceptFriendshipResponse) із кодом стану HTTP 200.

DELETE /friends/:friendId видаляє дружній зв'язок із користувачем за його ідентифікатором (friendId). Повертає результат (DeleteFriendResponse) із кодом стану HTTP 200.

DELETE /friends видаляє всі дружні зв'язки поточного користувача. Повертає результат (DeleteFriendResponse) із кодом стану HTTP 200.

4 ВИПРОБУВАННЯ КОМП'ЮТЕРНОЇ СИСТЕМИ

4.1 Випробування користувацького інтерфейсу

Головною метою при побудові інтерфейсу мобільного застосунку було забезпечення зручної, зрозумілої та стабільної взаємодії користувача з системою на сучасних мобільних пристроях з різними розмірами екранів. Особлива увага приділялася коректному відображенню елементів інтерфейсу на Android-пристроях, оскільки додаток розроблявся винятково для мобільної платформи.

На рисунку 4.1 зображено головний екран мобільного застосунку, де реалізовано механізм вертикального скролу для перегляду потенційних друзів за інтересами.

Головний екран мобільного застосунку складається з двох основних частин: верхньої панелі із заголовком сторінки та іконкою налаштувань, а також основного блоку з картками користувачів. Кожна картка містить ім'я, вік, країну та головний інтерес користувача у вигляді кольорової мітки.

У нижній частині екрана розташована панель навігації, яка забезпечує швидкий доступ до основних розділів застосунку: домашньої сторінки, списку друзів, сповіщень та профілю. Інтерфейс оптимізований для зручного взаємодії

однією рукою та має візуальну ієрархію, яка дозволяє користувачу легко зчитувати інформацію.

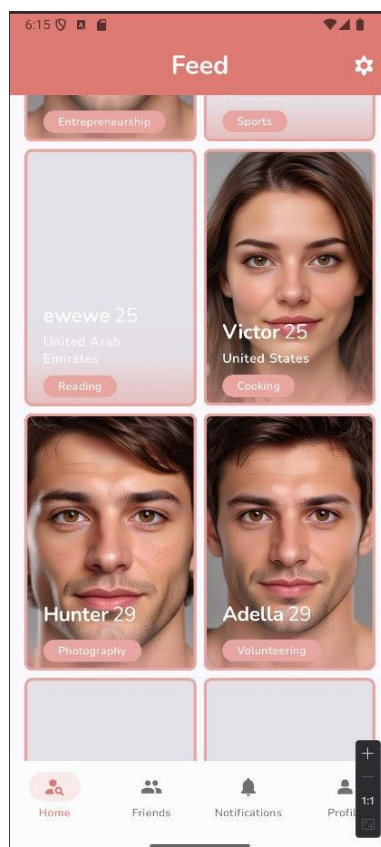


Рисунок 4.1 – Головна сторінка мобільного додатку (екран “Feed”)

Головний екран мобільного застосунку складається з двох основних частин: верхньої панелі із заголовком сторінки та іконкою налаштувань, а також основного блоку з картками користувачів. Кожна картка містить ім'я, вік, країну та головний інтерес користувача у вигляді кольорової мітки.

У нижній частині екрана розташована панель навігації, яка забезпечує швидкий доступ до основних розділів застосунку: домашньої сторінки, списку друзів, сповіщень та профілю. Інтерфейс оптимізований для зручного взаємодії однією рукою та має візуальну ієрархію, яка дозволяє користувачу легко зчитувати інформацію.

Ще одним важливим екраном застосунку є сторінка фільтрації, яка надає користувачу можливість гнучко налаштовувати параметри пошуку для персоналізації результатів. Цей екран відіграє ключову роль у покращенні

релевантності знайдених співрозмовників, дозволяючи враховувати вік та інтереси потенційних друзів.

На рисунку 4.2 зображено інтерфейс екрану фільтрів, який включає дві основні секції: вибір вікової категорії та набір інтересів.



Рисунок 4.2 – Екран фільтрів користувача

У верхній частині екрана розташовано повзунок для вибору вікового діапазону, який доповнений швидкими кнопками (Quick Selections) для швидкого встановлення типових вікових груп (18–24, 25–34 тощо). Обраний діапазон відображається у верхньому правому куті для зручності користувача.

Секція "Interests" представлена у вигляді інтерактивних міток, які дозволяють обирати теми, що цікавлять користувача. Вибрані інтереси автоматично підсвічуються кольором, що візуально відрізняє їх від неактивних. Це спрощує процес взаємодії та сприяє швидкому орієнтуванню у великому списку можливостей.

У нижній частині інтерфейсу реалізовано дві кнопки: “Cancel” та “Apply”, які дозволяють скасувати або застосувати вибрані фільтри відповідно.

Для підтримки соціальної взаємодії користувачів, важливим елементом застосунку є екран "Друзі" (рис.4.3) , який надає інтуїтивно зрозумілий інтерфейс для керування соціальними зв'язками.



Рисунок 4.3 – Екран друзів користувача

Екран друзів складається з верхньої панелі із заголовком “Friends” та кнопкою повернення, а також пошукового поля для швидкого знаходження контактів. Нижче розташовано три вкладки: “Friends”, “Sent” та “Received”, які дозволяють користувачу переглядати список поточних друзів, надіслані запити на дружбу та отримані запити відповідно. Кожна вкладка має чітке візуальне виділення, що полегшує навігацію.

Основна частина екрана відведена під список контактів, де кожна картка друга містить аватар, ім'я користувача та короткий опис, наприклад, його статус або інтереси. У нижній частині екрана розташована панель навігації, яка забезпечує

швидкий доступ до основних розділів застосунку: домашньої сторінки, списку друзів, сповіщень та профілю. Дизайн інтерфейсу створений з урахуванням зручності використання однією рукою, а також має чітку візуальну ієрархію для легкого сприйняття інформації.

Переглядаючи профіль іншого користувача, ви отримуєте вичерпну інформацію про нього, яка відображається на екрані профілю користувача (рис.4.4).

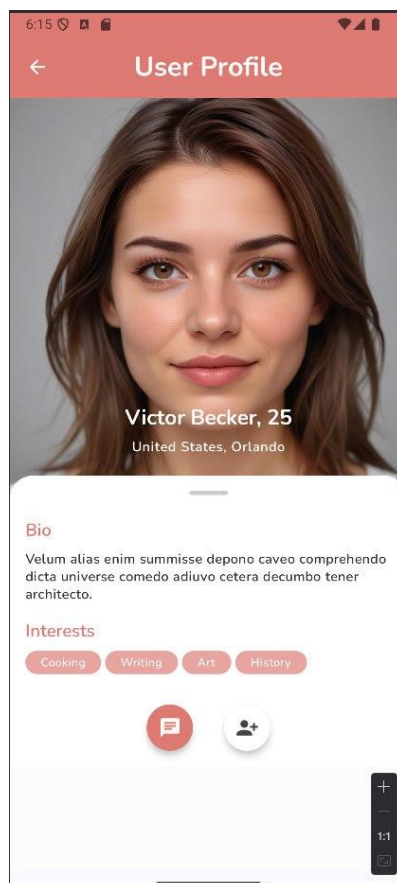


Рисунок 4.4 – Екран профілю користувача

Екран профілю користувача складається з кількох ключових секцій, що забезпечують повноцінне ознайомлення з інформацією. Верхня панель містить заголовок “User Profile” та кнопку повернення для зручної навігації. Основна частина екрана включає фотографію користувача, його ім’я, вік та місце проживання. Нижче розташований блок “Bio” з коротким описом, який дає змогу дізнатися більше про особистість користувача, наприклад, його хобі чи погляди.

Далі представлено секцію “Interests”, де у вигляді кольорових міток відображені основні інтереси користувача, такі як “Cooking”, “Writing”, “Art” та “History”. Це дозволяє швидко зрозуміти, чи є спільні теми для спілкування. У нижній частині екрана розміщені кнопки для взаємодії: відправлення повідомлення або додавання до друзів, що сприяє легкому встановленню контакту.

Екран власного профілю користувача (рис.4.5) в додатку "MateUp" призначений для перегляду та редагування персональної інформації автентифікованого користувача.

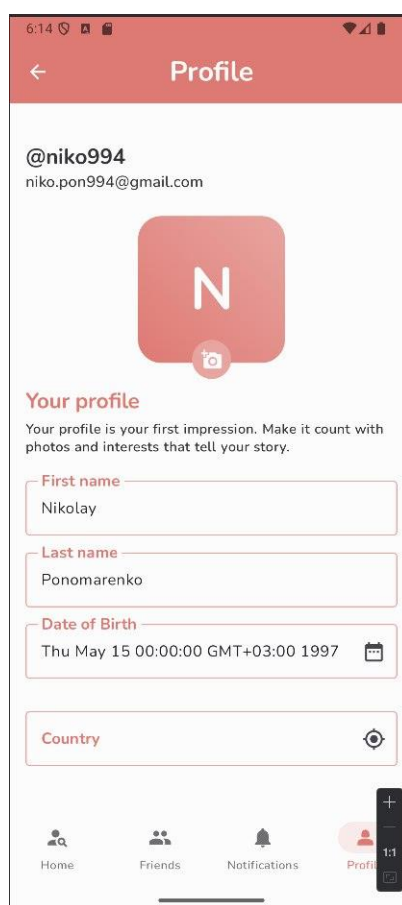


Рисунок 4.5 – Екран власного профілю користувача

Екран складається з верхньої панелі із заголовком "My Profile" та кнопкою налаштувань для швидкого доступу до додаткових опцій, таких як редагування профілю. Нижче розташовано блок із фотографією профілю, яка відображається через AsyncImage або ініціал імені як заглушка, із можливістю завантаження нового зображення через інтеграцію з модулем файлів.

Основна частина екрана поділена на секції: перша відображає основну інформацію, таку як ім'я, вік, країну та місто, а друга - біографію та інтереси користувача. Кнопки для редагування профілю та перегляду друзів розташовані нижче, забезпечуючи швидкий доступ до пов'язаних функцій. У нижній частині екрана розміщена панель навігації, яка дозволяє переходити до основних розділів додатка: домашньої сторінки, списку друзів, сповіщень та профілю.

4.2 Функціональне випробування

Під час перевірки інтерфейсу головної сторінки, екрану профілю та екрану друзів було також перевірено їх базову функціональність. У даному підрозділі буде опущено перевірку навігації між екранами та відображення інтерфейсу, зосередившись на функціональному тестуванні. Перевірка стосуватиметься наступних функціональностей:

- реєстрація та автентифікація;
- налаштування фільтрів;
- налаштування профілю.

Процес перевірки реєстрації та автентифікації розпочато з введення даних у форму реєстрації на екрані AuthScreen. Введено ім'я користувача "@niko994", електронну пошту "niko.pon994@gmail.com" та пароль "Kknr1769!23456". Надіслано запит на реєстрацію через ендпоінт /auth/sign-up. Після успішного надсилання запиту перевірено наявність нового запису в базі даних у таблиці users_core (рис.4.6). Запис містив коректні дані, а пароль було збережено у шифрованому вигляді.

Data Output Messages Notifications							
Showing rows: 1 to 53 Page No: 1 of 1							
id [PK] uuid	username character varying (255)	email character varying	roles users_core_roles_enum[]	password character varying	isProfileFilled boolean	phoneNum character varying	
14	3cc68ef8-fb09-4c58-a0eb-5fa77e8a550e	@kfwefkew	nikolay.dasda@gmail.com	(USER)	\$2b\$10\$VlQy7Phd84bMudT/qgoeGt2IPnc0eYmKXhK68UTVMUKTH5.ms	false	[null]
15	47c6b9a8-eae2-46b2-9829-8d39075a6d...	@gkgerkerwewqewq	fkwekew@gmail.com	(USER)	\$2b\$10\$9XjSUo4kvdNAGlqxVwRuBMyWdGMHdi8IoGeLRpNQKA.PAE3oA6	true	[null]
16	497ad4b7-8dea-41d4-9ae9-a664ba53cc...	@wewqewwq	efewdw@fgmail.com	(USER)	\$2b\$10\$dZFn0EZ_5crqTVeZle7fedWuWeZjzzVkcSg6sUSg234V6RuE1DW	true	[null]
17	4b4e4256-fe3b-48b5-a586-09f8c052c21f	@wwewdsa	dfffewk@gmail.com	(USER)	\$2b\$10\$J4MNowN6kapE3fVZ.JpJJe/pNN1NJMVpPsTDM3V9FcxaswmvDAKX6	false	[null]
18	4df6c1db-d98c-48aa-a823-d93723d8d3...	@ewqewqe	fewfew@gmail.com	(USER)	\$2b\$10\$MuRPTsYTIM.GmNk7_UA2y01k0aUC28mfTypOdiPNi75FkDy78kOca	false	[null]
19	4e267923-5f0f-4fc5-80a4-af6e012a2f3e	@niko994	niko.pon994@gmail.com	(USER)	\$2b\$10\$8iAvbhfZ9v/ggNbPCbwFOiREnW/LRY/FpibQjXLgDfQfIKm7.y	true	[null]
20	50bc4c71-86b7-4542-bd09-9e1dccc27c...	@rewrwe	gergerg@gmail.com	(USER)	\$2b\$10\$QWibYl0wiPIIGxyTo5VqbuMdqM7uixWbnQvLkKkAF5yflj.nX5ewi	false	[null]
21	526cf2bf-e88b-4fbd-afcb-cbe891e558a2	@kko03eee	gegereg@gmail.com	(USER)	\$2b\$10\$KDbg3n5yxJxr3RivXf.zLOP8S2N0tVb82ITOhZqZBRroxDUlzaYM.	false	[null]
22	6be70853-ede1-46a3-9d4f-242ec6f94abb	@jgijgjier	krtgkretk@gmail.com	(USER)	\$2b\$10\$WCQp9gm6b9GiStf1x3iojedAfonIE0y/8AKWkYhUe.JZ.tnJyB-HKq	false	[null]
23	6c87a731-e0ee-4cd8-b7cc-dcbfd2b586a7	@admin_user_soon	admin@example.com	(ADMIN,USER)	\$2b\$10\$kyXLz0224TLHITJgAf6cGeTci9jC1a8ZYpYyOulC0DhGQp3kCa5C	true	+12368128
24	712e5d4e-e0da-463e-8671-1a42aeb81...	@hrthtrtwe	rwerewrew@gmail.com	(USER)	\$2b\$10\$byeOALavCUPaW.PS2g.eyekONOKSLR9dAXIHfVwJWYxmo/BHxkq	false	[null]

Рисунок 4.6 – Створений запис користувача у базі даних

Далі виконано вхід через ендпоінт `/auth/login` із тими ж даними, після чого отримано токени доступу (`accessToken` та `refreshToken`), що підтверджує успішну автентифікацію.

Для перевірки функціональності фільтрів використано екран `FiltersScreen`. Після автентифікації користувача "`@niko994`" перейдено до екрану фільтрів через головну сторінку. Встановлено віковий діапазон від 25 до 34 років та обрано інтереси "Art", "Cooking", "Entrepreneurship" та інші. Натиснуто кнопку "Apply", після чого надіслано запит через `UpdateFiltersUseCase` до ендпоінта `/filters/update-filters`, де маємо запис у базі даних для користувача "`@niko994`", що підтверджує коректність роботи фільтрів (рис.4.7).

Data Output Messages Notifications							
Showing rows: 1 to 36 Page No: 1 of 1							
id [PK] uuid	userid uuid	ageFrom integer	ageTo integer	interests user_filters_interests_enum[]			
25	7bb5f7ed-e798-4854-ada0-d333334b8...	712e5d4e-e0da-463e-8671-1a42aeb8...	18	100	{Art,Cooking,Entrepreneurship}		
26	875b2b5f-e44a-4ed8-b202-8543cda4b1...	d2eae6e0-e6bb-4193-bbc3-7f95b0c6e...	18	100	{Art,Cooking,Entrepreneurship}		
27	9d78e7d5-9d37-44f9-be52-63f0e768d9...	4df6c1db-d98c-48aa-a823-d93723d8d...	18	100	{Art,Cooking,Entrepreneurship}		
28	a2a2fe00-4f44-48eb-91b8-c3b368c771...	b367637b-782a-44c9-955f-65ebee1aff...	18	100	{Art,Cooking,Entrepreneurship}		
29	a5c03e7b-b6dc-4702-89fa-666a05c472...	c0b80034-3878-4276-85bb-bb0fbacc2...	18	100	{Art,Cooking,Entrepreneurship}		
30	ba000220-9c61-4623-8350-cade5841f4...	d61a1e6a-2348-4184-8ecd-532760cb1...	18	100	{Art,Cooking,Entrepreneurship}		
31	be6cd588-8312-49d9-9389-19366aa74...	4e267923-5f0f-4fc5-80a4-af6e012a2f3e	25	34	{Art,Cooking,Entrepreneurship,Sports,Gaming,Programming,Reading,Music,Nature,Fitness}		
32	d3014bfb-44bf-44e4-90f9-71525e0db6...	ba934f99-34a6-4b77-90a2-a74b21905...	18	100	{Art,Cooking,Entrepreneurship}		
33	dd36532e-49e3-448c-828b-030a28be2...	02ecde34-80d1-4c32-9e70-e1acb666d...	18	100	{Art,Cooking,Entrepreneurship}		
34	e1612741-806c-468b-97c8-51c4aec2a...	0fd1e56f-cc1d-4cde-930e-dd90b2d814...	35	44	{Art,Cooking,Entrepreneurship,Reading,Sports,Nature}		
35	efb48327-d972-4674-bdde-1a5e5f4879...	b89ad48-557b-4761-afb0-1e96da428...	18	100	{Art,Cooking,Entrepreneurship}		
36	ff45d5ca-e1e3-456d-b366-a17191b4e0f9	526cf2bf-e88b-4fbd-afcb-cbe891e558a2	18	100	{Art,Cooking,Entrepreneurship}		

Рисунок 4.7 – Записи таблиці "user_filters" бази даних

Для тестування оновлення профілю використано екран `FeedUserProfileScreen` після автентифікації користувача "`@niko994`". Натиснуто кнопку редагування профілю, після чого введено нові дані: змінено ім'я на "Nikolay", прізвище на

"Ponomarenko", дату народження на "15 May 1997" та країну на "Ireland". Дані відправлено через ендпоінт /profile/update-profile. Перевірено таблицю users_profile у базі даних: для користувача "@niko994" оновлено поля firstName, lastName, dateOfBirth і country відповідно до введених значень (рис.4.8). На екрані профілю відображено оновлені дані, що підтверджує коректність роботи функціоналу оновлення.

Data Output Messages Notifications						
Showing rows: 1 to 24 Page No: 1 of 1						
	id [PK] uuid	userid uuid	firstName character varying (255)	lastName character varying (255)	gender users_profile_gender_enum	bio character varying (300)
11	65d189c1-43f5-4951-8d45-2f8c57df6d...	144bb29b-230b-4960-b9e0-297ce6a0d8...	mijjio	ghuihuihui	MALE	kikiojkioj
12	84c5e190-cf8f-46a8-b3c9-0f92127233...	237140b9-9e39-44e1-b5a7-e899165b84...	eewqewq	ewqewqe	MALE	rwerewrew
13	8c5d163c-875d-4e9a-9233-a40670fe9...	815d2d7b-563e-4b5e-ab8a-542a8ee00b...	Giovanni	Lesch	FEMALE	Compectus demitto caecus adinventitas vulpes audacia architecto confido tollo t
14	94c312b8-d4f7-4678-afa9-3bc9ce7836...	47c6b9a8-eae2-46b2-9829-8d39075a6d...	Nikolay	rewrew	FEMALE	qweqeq
15	9dc96d1b-f1d2-4217-8cd9-9c95071efa...	3882df86-205c-4453-a4ab-9119ae3511...	Cortney	Botsford	MALE	Denuo degenero tardus arca clam victus vis sol dapifer demens thorax cotidie abst
16	a1d282bf-92c1-48a5-9b44-737e227ff4...	4e267923-5f0f-4fc5-80a4-afe012a2f3e...	Nikolay	Ponomarenko	MALE	30 y.o Backend Node.js Developer from Dublin. Adore cooking, hiking and reading
17	b73ba429-3d7c-45a3-967b-c94e23ea5...	85844a2c-be05-48c0-bd58-8417bd0491...	Victor	Becker	FEMALE	Velum alias enim summis depono caveo comprehendo dicta universe comedo a
18	cec20c09-5519-4bee-8e96-0cd5c1edc...	92f4999a-3e36-4721-b346-a9b650e643...	Jaunita	Effertz	FEMALE	Spectaculum ter cauda sustineo facere sumo commemoro xiphias comparo testir
19	d55f3d3c-1480-4fed-87ff-344e1e8d9ae7	9233b612-92ef-4b4f-86b7-b8d11b86c1cb	Nikolay	Pon	MALE	rwerewrew
20	de73054c-10be-4f77-853a-677982f2d3...	ec8afcd4-dac4-42ba-8d36-5ed962a1f25d	Orrin	Kilback	MALE	Caries soluta creator curvo auctus supra non voro bibo solio utrum aggero recusa
21	e983266a-d25a-49e7-ab96-17f27eed9...	0bd1924d-3dda-4a50-b9ea-9c532ad536...	Nikolay	Ponomarenko	MALE	kofewfw

Рисунок 4.8 – Записи таблиці "users_profile" бази даних

6:14

Profile

@niko994
niko.pon994@gmail.com

N

Your profile

Your profile is your first impression. Make it count with photos and interests that tell your story.

First name
Nikolay

Last name
Ponomarenko

Date of Birth
Thu May 15 00:00:00 GMT+03:00 1997

Country

Home Friends Notifications Profile

Рисунок 4.10 – Екран профілю користувача після перевірки

ВИСНОВКИ

Розробка комп'ютерної системи, описаної в даній дипломній роботі, є надзвичайно актуальною в умовах стрімкого зростання потреби у створенні ефективних, масштабованих, безпечних та користувацьки зручних мобільних і вебзастосунків. В умовах активного розвитку цифрових технологій, особливого значення набуває обґрунтований вибір архітектури, технологій та методів проєктування, адже саме від цих рішень залежить якість, надійність, продуктивність і подальші можливості розвитку програмного продукту.

У першому розділі було детально проаналізовано сучасні підходи до побудови інформаційних систем, проведено огляд актуальних технологій та архітектурних рішень. З огляду на поставлені завдання проєкту та специфіку предметної області, обрано клієнт-серверну архітектуру, яка забезпечує гнучкість і масштабованість системи. Для реалізації клієнтської частини мобільного додатку обґрунтовано використання Kotlin разом із Jetpack Compose як сучасного фреймворку для побудови адаптивного і швидкого UI. Для серверної частини обрано Node.js у поєднанні з Nest.js, що дозволяє створювати надійну серверну платформу із чистою структурою коду та зручною реалізацією REST API. Крім того, проведено порівняльний аналіз кількох систем управління базами даних, за результатами якого обґрунтовано вибір PostgreSQL як продуктивної, безпечної та функціонально розвиненої СУБД, що відповідає вимогам до проєкту.

У другому розділі детально сформульовано вимоги до системи, включаючи функціональні, нефункціональні, технічні та експлуатаційні характеристики. Розроблено повноцінну специфікацію вимог SRS (Software Requirements Specification), що є основою для подальшого проєктування та розробки системи. Створено діаграми варіантів використання, діаграми послідовності, а також сформовано детальні описи прецедентів, які наочно демонструють логіку роботи системи та порядок взаємодії користувачів з її функціоналом. Це дозволило чітко

структурувати сценарії використання і визначити пріоритетність функцій, що реалізуються.

У третьому розділі виконано безпосередню розробку системи. Реалізовано клієнтську частину мобільного додатку, яка забезпечує зручний та інтуїтивно зрозумілий інтерфейс для користувачів. Особливу увагу приділено питанням UX/UI дизайну, адаптації до різних типів пристроїв і забезпеченню коректного відображення елементів інтерфейсу. Серверна частина вебсистеми реалізована із дотриманням принципів модульності, безпечної обробки даних та оптимізованої взаємодії з клієнтськими застосунками через API. Передбачена структура дозволяє оперативно додавати нові функціональні модулі та інтегрувати систему з іншими сервісами, що значно підвищує її гнучкість та розширюваність.

У четвертому розділі виконано всебічне тестування створеної системи. Проведено функціональне тестування основних модулів, яке підтвердило їх працездатність відповідно до встановлених вимог. Окремо протестовано користувацький інтерфейс, результати чого засвідчили його зручність, інтуїтивність та відповідність сучасним стандартам UX/UI-дизайну. Додатково виконано перевірку продуктивності та стабільності роботи системи під різними навантаженнями, що дозволило переконатися у її надійності.

Таким чином, усі поставлені в дипломній роботі цілі та завдання були успішно досягнуті. Розроблена система відповідає сучасним технологічним стандартам, має продуману архітектуру та структуровану логіку роботи. Вона пройшла повний цикл тестування і демонструє стабільну та коректну роботу в різних середовищах. Завдяки обраним технічним рішенням система має значний потенціал для подальшого розвитку, інтеграції нових сервісів, розширення функціоналу та впровадження у реальні проєкти.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Software Architecture Guide. URL: <https://martinfowler.com/architecture/> (дата звернення: 23.04.2025);
2. HOW TO DECIDE THE ARCHITECTURE OF SOFTWARE. URL: <https://omerabay1.medium.com/how-to-decide-the-architecture-of-software-b531efa6a980> (дата звернення: 23.04.2025);
3. Microservices vs Monoliths: Benefits, Tradeoffs, and How to Choose Your App's Architectue. URL: <https://www.freecodecamp.org/news/microservices-vs-monoliths-explained/> (дата звернення: 23.04.2025);
4. Advantages of microservices and disadvantages to know. URL: <https://www.atlassian.com/microservices/cloud-computing/advantages-of-microservices> (дата звернення: 23.04.2025);
5. Serverless Architecture Overview. URL: <https://www.datadoghq.com/knowledge-center/serverless-architecture/> (дата звернення: 23.04.2025);
6. Pros and Cons of Serverless Architecture. URL: <https://bejamas.com/hub/guides/serverless-architectures> (дата звернення: 24.04.2025);
7. Kotlin language specification. URL: <https://kotlinlang.org/spec/introduction.html> (дата звернення: 24.04.2025);
8. A comprehensive Guide to Jetpack Compose. URL: <https://medium.com/@ramadan123sayed/a-comprehensive-guide-to-jetpack-compose-b5add214692f> (дата звернення: 24.04.2025);
9. Introduction to React Native. URL: <https://www.geeksforgeeks.org/introduction-react-native/> (дата звернення: 25.04.2025);
10. Introduction to Node.js. URL: <https://nodejs.org/en/learn/getting-started/introduction-to-nodejs> (дата звернення: 25.04.2025);
11. Introduction to Nest.js. URL: <https://docs.nestjs.com/> (дата звернення: 25.04.2025);

12. Introduction to PHP. URL: <https://www.php.net/> (дата звернення: 25.04.2025);

13. Introduction to Laravel. URL: <https://laravel.com/> (дата звернення: 25.04.2025);

14. Relational database. URL: https://en.wikipedia.org/wiki/Relational_database (дата звернення: 30.04.2025);

15. Understanding NoSQL Databases. URL: <https://medium.com/@myrajarenga1234/understanding-nosql-databases-2214ce49e68c> (дата звернення: 30.04.2025);

16. MySQL. URL: <https://en.wikipedia.org/wiki/MySQL> (дата звернення: 30.04.2025);

17. What is PostgreSQL? URL: <https://aws.amazon.com/rds/postgresql/what-is-postgresql/> (дата звернення: 30.04.2025);

18. MongoDB Basics. URL: <https://www.mongodb.com/resources/products/fundamentals/basics> (дата звернення: 30.04.2025);

19. Apache Cassandra: Open Source NoSQL Database. URL: https://cassandra.apache.org/_/index.html (дата звернення: 30.04.2025);

ДОДАТОК А

Лістинг А.1 – Код файлу build.gradle.kts

```
import
com.android.build.gradle.internal.cxx.configure.gadleLocalProperties

plugins {
    alias(libs.plugins.android.application)
    alias(libs.plugins.kotlin.android)
    alias(libs.plugins.kotlin.compose)
    id("org.jetbrains.kotlin.plugin.serialization") version "1.9.23"
    id("com.google.dagger.hilt.android")
    kotlin("kapt")
    id("kotlin-parcelize")
}

android {
    namespace = "com.example.mateup"
    compileSdk = 35
    defaultConfig {
        applicationId = "com.example.mateup"
        minSdk = 26
        targetSdk = 35
        versionCode = 1
        versionName = "1.0"
        testInstrumentationRunner =
"androidx.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            isMinifyEnabled = false
            proguardFiles(
                getDefaultProguardFile("proguard-android-
optimize.txt"),
                "proguard-rules.pro"
            )
        }
    }
    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_1_8
        targetCompatibility = JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = "1.8"
    }
    buildFeatures {
        compose = true
    }
}
```

```
dependencies {
    (Опис залежностей, які використовуються в проекті - Retrofit,
    OkHttp та інші)
}
```

Лістинг А.2 – Код файлу AppNavigation.kt

```
@Composable
fun AppNavigation(
    navController: NavHostController = rememberNavController(),
    authViewModel: AuthViewModel = hiltViewModel(),
) {
    val authState by authViewModel.authState.collectAsState()
    val authUser = authState.user
    val startDestination = when {
        authState.state == UserAuthState.Authenticated -> {
            if (authUser?.isProfileFilled == true) Screen.Feed.route
        else Screen.Profile.route
        }
        else -> Screen.Login.route
    }
    (Компонент NavHost для опису екранів навігації - профілю,
    реєстрації, автентифікації та інші)
}
```

Лістинг А.3 – Код екрану LoginScreen

```
@Composable
fun LoginScreen(
    viewModel: LoginViewModel = hiltViewModel(),
    onLoginSuccess: (Boolean) -> Unit,
    onNavigateToSignUp: () -> Unit,
) {
    val uiState by viewModel.uiState.collectAsStateWithLifecycle()

    AuthenticationLayout {
        Column(
            modifier = Modifier
                .wrapContentSize()
                .clip(shape = RoundedCornerShape(20.dp))
                .background(OffWhite)
                .padding(horizontal = 16.dp, vertical = 24.dp)
        ) {
            AuthWelcomeMessage(
                firstTitle = "Welcome",
                secondTitle = "back!",
                description = "Please confirm you're our favorite
human and not the bot who keeps forgetting their password. Your
account is eagerly waiting for you."
            )
        }
    }
}
```

```

        (Компоненти для введення електронної пошти та паролю для
        автентифікації та інші)
    }
}

```

Лістинг А.4 – Код екрану SignUpScreen

```

@Composable
fun SignUpScreen(
    onSignUpSuccess: (Boolean) -> Unit,
    onNavigateToLogin: () -> Unit,
    viewModel: SignUpViewModel = hiltViewModel()
) {
    val uiState by viewModel.uiState.collectAsStateWithLifecycle()

    AuthenticationLayout {
        Column(
            modifier = Modifier
                .wrapContentSize()
                .clip(shape = RoundedCornerShape(20.dp))
                .background(OffWhite)
                .padding(horizontal = 16.dp, vertical = 24.dp),
            horizontalAlignment = Alignment.CenterHorizontally,
        ) {
            AuthWelcomeMessage(
                firstTitle = "What's up,",
                secondTitle = "mate!",
                description = "New faces, new possibilities. Create
your account and start connecting today"
            )
            (Компоненти для введення нікнейму, електронної пошти та
            паролю для реєстрації та інші)
        }
    }
}

```

Лістинг А.5 – Код файлу ProfileScreen.kt

```

@OptIn(ExperimentalMaterial3Api::class,
ExperimentalPermissionsApi::class)
@Composable
fun ProfileScreen(
    modifier: Modifier = Modifier,
    profileViewModel: ProfileViewModel = hiltViewModel(),
    countryViewModel: CountryViewModel = hiltViewModel(),
    interestsViewModel: InterestsViewModel = hiltViewModel(),
    onSaveProfileSuccess: () -> Unit,
) {
    val profileUiState by
profileViewModel.uiState.collectAsStateWithLifecycle()
    val countryUiState by
countryViewModel.uiState.collectAsStateWithLifecycle()
    val user = profileUiState.user
}

```

```

        val snackbarHostState = remember { SnackbarHostState() }
        val pickImageLauncher = rememberLauncherForActivityResult(
            contract = ActivityResultContracts.PickVisualMedia(),
            onResult = { uri -> uri?.let
{ profileViewModel.onProfilePhotoSelected(it) } };
profileViewModel.closePhotoSourceChooser() }
    )

    // Обробка сповіщень
    LaunchedEffect(profileUiState.photoUploadError)
{ profileUiState.photoUploadError?.let
{ snackbarHostState.showSnackbar(it, duration =
SnackbarDuration.Short); profileViewModel.dismissSaveStatus() } }

    // Відображення вибору фото
    if (profileUiState.showImageSourceChooser)
ProfileImagePickerClassBottomSheet(
    onDismissRequest =
{ profileViewModel.closePhotoSourceChooser() },
    pickImageLauncher = pickImageLauncher,
)

    (Компоненти для відображення інформації про користувача у
налаштуваннях профілю – фото, ім'я, прізвище та інші)
}
}

```

Лістинг А.6 – Код ProfileViewModel (фрагмент)

```

@HiltViewModel
class ProfileViewModel @Inject constructor(
    private val getUserUseCase: GetUserUseCase,
    private val getUserProfileUseCase: GetUserProfileUseCase,
    private val createUserProfileUseCase: CreateUserProfileUseCase,
    private val updateUserProfileUseCase: UpdateUserProfileUseCase,
    private val uploadPhotoUseCase: UploadPhotoUseCase,
    private val tokenManager: TokenManager,
    private val fileHelper: FileHelper,
) : ViewModel() {
    private val _uiState = MutableStateFlow(ProfileUiState())
    val uiState: StateFlow<ProfileUiState> = _uiState.asStateFlow()

    init {
        viewModelScope.launch {
            val user = loadAuthenticatedUser()
            if (user != null && user.isProfileFilled)
loadAuthenticatedUserProfile()
        }
    }

    private suspend fun loadAuthenticatedUser(): AuthUser? {

```

```

        val bearerToken =
TokenHelper.convertRawTokenToBearer(tokenManager.getAccessToken() ?:
return null)
        return when (val result =
getUserUseCase.execute(bearerToken)) {
            is ResponseResult.Success -> { _uiState.update
{ it.copy(user = result.data) }; result.data }
            is ResponseResult.Failure -> { _uiState.update
{ it.copy(profileError = result.error.displayMessage) }; null }
        }
    }
    // Інші методи (loadAuthenticatedUserProfile,
onSaveProfileClick, onFirstNameChange, onDateSelected тощо) опущено
для стислості
}

```

Лістинг А.7 – Код компонента FeedCard

```

@Composable
fun FeedCard(
    user: FeedUser,
    onUserCardClick: (user: FeedUser) -> Unit,
    modifier: Modifier = Modifier
) {
    val cardShape = RoundedCornerShape(8.dp)

    Card(
        modifier = modifier
            .aspectRatio(1f / 1.5f)
            .clip(cardShape)
            .clickable { onUserCardClick(user) },
        shape = cardShape,
        border = BorderStroke(3.dp, MediumPink),
        elevation = CardDefaults.cardElevation(defaultElevation =
6.dp)
    ) {
        Column(modifier = Modifier.fillMaxSize()) {
            Box(modifier = Modifier.fillMaxWidth().weight(1f)) {
                AsyncImage(
                    model =
ImageRequest.Builder(LocalContext.current)
                        .data(user.profile.profilePhoto)
                        .crossfade(true)
                        .build(),
                    contentDescription =
"$ {user.profile.firstName}'s profile picture",
                    contentScale = ContentScale.Crop,
                )
                (Компоненти для відображення інформації про користувача –
вік, ім'я, країна та інші)
            }
        }
    }
}

```

Лістинг А.8 – Код екрану FiltersScreen

```

@Composable
fun FiltersScreen(
    modifier: Modifier = Modifier,
    filtersViewModel: FiltersViewModel = hiltViewModel(),
    interestsViewModel: InterestsViewModel = hiltViewModel(),
    onCancelClick: () -> Unit,
    onApplySuccess: () -> Unit,
) {
    val filtersUiState by
filtersViewModel.uiState.collectAsStateWithLifecycle()
    var rangeSliderPosition =
filtersUiState.ageFrom.toFloat()..filtersUiState.ageTo.toFloat()

    if (filtersUiState.isLoading) {
        CircularProgressIndicator()
    } else {
        Box(
            modifier = modifier
                .fillMaxSize()
                .padding(16.dp)
        ) {
            Column(
                modifier = Modifier
                    .fillMaxSize()
                    .verticalScroll(rememberScrollState())
                    .padding(bottom = 80.dp)
                    .align(Alignment.TopStart),
            ) {
                (Компоненти для відображення слайдери для вибору
вікового діапазону та списку інтересів)
            }
        }
    }
}

```

Лістинг А.9 – Код FiltersViewModel (фрагмент)

```

@HiltViewModel
class FiltersViewModel @Inject constructor(
    private val tokenManager: TokenManager,
    private val getFiltersUseCase: GetFiltersUseCase,
    private val updateFiltersUseCase: UpdateFiltersUseCase,
) : ViewModel() {
    companion object {
        private const val GET_AUTHENTICATED_USER_TAG: String =
            "FiltersViewModel::loadAuthenticatedUser"
    }

    private val _uiState = MutableStateFlow(FiltersUiState())
    val uiState: StateFlow<FiltersUiState> = _uiState.asStateFlow()

    init {

```

```

viewModelScope.launch {
    try {
        val user = loadAuthenticatedUser()
        if (user != null) fetchFilters()
    } catch (exception: Exception) {
        _uiState.update {
            it.copy(
                isLoading = false,
                filtersError = exception.message ?: "Error"
            )
        }
    }
}

private suspend fun loadAuthenticatedUser(): AuthUser? {
    try {
        val user = tokenManager.getUser().firstOrNull()
        _uiState.update { it.copy(user = user) }
        return user
    } catch (exception: Exception) {
        Log.e(GET_AUTHENTICATED_USER_TAG, exception.message ?:
"Something went wrong")
        return null
    }
}

// Інші методи (fetchFilters, onAgeRangeChange,
handleInterestClick, onApplyFiltersClick) опущено для стислості

```

Лістинг А.10 – Код компонента FeedUserProfileCard

```

@Composable
fun FeedUserProfileCard(
    profile: UserProfile,
    onChatClick: () -> Unit,
    onAddFriendClick: () -> Unit,
    modifier: Modifier = Modifier
) {
    Column {
        Box(
            modifier = modifier
                .fillMaxSize()
                .background(
                    Brush.getVerticalGradient(
                        colors = listOf(
                            Color.Transparent,
                            LightPink.copy(alpha = 0.2f),
                            MediumPink.copy(alpha = 0.60f)
                        ),
                        VerticalBrush(startY = 300f, endY =
Float.POSITIVE_INFINITY)
                    ),

```

```

        )
    ) {
        if (profile.profilePhoto != null) {
            AsyncImage(
                model =
                ImageRequest.Builder(LocalContext.current)
                    .data(profile.profilePhoto)
                    .crossfade(true)
                    .build(),
                contentDescription = "${profile.firstName}'s
profile picture",
                contentScale = ContentScale.Crop,
                modifier = Modifier.fillMaxSize(),
            )
        } else if (profile.firstName != null) {
            profile.firstName.substring(0, 1).uppercase().let {
                Text(
                    text = it,
                    fontSize = (300 / 3).sp,
                    color = OffWhite,
                    fontWeight = FontWeight.Bold,
                    textAlign = TextAlign.Center,
                    modifier = Modifier.padding(4.dp)
                )
            }
        }
    }
    (Компоненти для опису даних профілю користувача -ім'я,
    країна, вік та інші)
}
}

```

Лістинг А.11 – Код FeedUserProfileViewModel

```

@HiltViewModel
class FeedUserProfileViewModel @Inject constructor(
    private val requestFriendshipUseCase: RequestFriendshipUseCase,
    private val tokenManager: TokenManager,
) : ViewModel() {
    private val _uiState =
MutableStateFlow(FeedUserProfileUiState())
    val uiState: StateFlow<FeedUserProfileUiState> =
_uiState.asStateFlow()

    fun requestFriendship(recipientId: String) {
        _uiState.update {
            it.copy(
                isLoading = true,
                feedUserProfileError = null,
            )
        }
        (Логіка по відправки запиту на дружбу)
    }
}

```

Лістинг А.12 – Код екрану FriendsScreen

```

@Composable
fun FriendsScreen(
    friendsViewModel: FriendsViewModel = hiltViewModel(),
    modifier: Modifier = Modifier,
) {
    val friendsUiState by
    friendsViewModel.uiState.collectAsStateWithLifecycle()

    if (friendsUiState.isLoading) {
        DataLoadingProgressIndicator()
    } else {
        Column(
            modifier = modifier
                .fillMaxSize()
                .background(OffWhite)
        ) {
            SearchTextField(
                searchValue = friendsUiState.searchValue,
                onSearchValueChange =
            { friendsViewModel.onSearchValueChange(it) }
            )
            HorizontalDivider(color = Coral)
            TabRowComponent(
                tabs = friendsUiState.tabs,
                selectedTabIndex = friendsUiState.selectedTabIndex,
                onSelectTabIndex = { index ->
                    friendsViewModel.onSelectTabIndex(index)
                },
            ) { selectedIndex ->
                when (selectedIndex) {
                    0 -> UserFriendsScreen(
                        expandedMenuFriendId =
friendsUiState.expandedMenuFriendId,
                        onMoreOptionsClick = {
                            friendsViewModel.onOpenMenuRequest(it)
                        },
                        menuItems = friendsViewModel.menuItems,
                        onDismissMenuRequest = {
                            friendsViewModel.onDismissMenuRequest()
                        },
                        friends = friendsUiState.friends,
                    )

                    1 -> SentRequestsScreen(sentRequests =
friendsUiState.sentRequests)
                    2 -> ReceivedRequestsScreen(receivedRequests =
friendsUiState.receivedRequests)
                }
            }
        }
    }
}

```

ЛІСТИНГ А.13 – Код FriendsViewModel (фрагмент)

```

@HiltViewModel
class FriendsViewModel @Inject constructor(
    private val getFriendsUseCase: GetFriendsUseCase,
    private val getSentFriendshipRequestsUseCase:
GetSentFriendshipRequestsUseCase,
    private val getReceivedFriendshipRequestsUseCase:
GetReceivedFriendshipRequestsUseCase,
    private val tokenManager: TokenManager,
) : ViewModel() {
    private val _uiState = MutableStateFlow(FriendsUiState())
    val uiState: StateFlow<FriendsUiState> = _uiState.asStateFlow()

    init {
        viewModelScope.launch {
            try {
                val user = loadAuthenticatedUser()
                if (user != null) {
                    fetchFriends()
                    fetchSentFriendshipRequests()
                    fetchReceivedFriendshipRequests()
                }
            } catch (exception: Exception) {
                _uiState.update {
                    it.copy(
                        isLoading = false,
                        friendsError = exception.message ?: "Error"
                    )
                }
            }
        }
    }

    private suspend fun loadAuthenticatedUser(): AuthUser? {
        try {
            val user = tokenManager.getUser().firstOrNull()
            _uiState.update { it.copy(user = user) }
            return user
        } catch (exception: Exception) {
            Log.e("FriendsViewModel::loadAuthenticatedUser",
exception.message ?: "Something went wrong")
            return null
        }
    }

    // Методи fetchFriends, fetchSentFriendshipRequests,
    fetchReceivedFriendshipRequests, onSelectTabIndex тощо опущено для
    стислості
}

```

Лістинг А.14 – Код контролера AuthController

```

@ApiTags("auth")
@Controller("auth")
export class AuthController {
    constructor(private readonly authService: AuthService) {}

    @ApiBody({ type: UserLoginDto })
    @Public()
    @Post("/login")
    @HttpCode(HttpStatus.OK)
    async login(@Body() userLoginDto: UserLoginDto):
    Promise<LoginResponse> {
        return await this.authService.login(userLoginDto);
    }

    @Public()
    @UseGuards(DoesUserExistGuard)
    @Post("/sign-up")
    @HttpCode(HttpStatus.OK)
    async signUp(@Body() userSignUpDto: UserSignUpDto):
    Promise<SignUpResponse> {
        return await this.authService.signUp(userSignUpDto);
    }

    @UseGuards(JwtAuthGuard)
    @Get("/user")
    @HttpCode(HttpStatus.OK)
    async getUser(@GetCurrentUserId() userId: string):
    Promise<UserCore> {
        return await this.authService.getUser(userId);
    }
    (Ендпоінти для виходу з аккаунту та оновлення access token і
    refresh token)
}

```

Лістинг А.15 – Фрагмент коду контролера FileController

```

@ApiTags("Files")
@ApiBearerAuth()
@Controller("file")
export class FileController {
    constructor(
        private readonly fileService: FileService,
        private readonly logger: WinstonService,
    ) {}

    @Post()
    @HttpCode(HttpStatus.CREATED)
    @UseInterceptors(FileInterceptor("photo"))
    async uploadPhoto(
        @UploadedFile()
        new ParseFilePipeBuilder()

```

```

        .addValidator(
            new UploadFileTypeValidator({
                fileType: VALID_UPLOADS_MIME_TYPES,
            }),
        )
        .addMaxSizeValidator({
            maxSize: MAX_PICTURE_SIZE_IN_BYTES,
            message: `Photo should be less than
${MAX_PICTURE_SIZE_IN_BYTES} bytes`,
        })
        .build({ errorHttpStatusCode:
HttpStatus.UNPROCESSABLE_ENTITY }),
    )
    file: Express.Multer.File,
    @Request() req: any,
    @Response() res: any,
) {
    const userId = req?.user?.id;

    if (!userId) {
        throw new UnauthorizedException("User ID is not found");
    }

    return await this.fileService.uploadFile(file, userId);
}
(Ендпоінти для отримання фотографій користувача, видалення
конкретної фотографії та видалення усіх фотографій)
}

```

Лістинг А.16 – Фрагмент коду контролера FiltersController

```

@Controller("filters")
@UseGuards(JwtAuthGuard)
export class FiltersController {
    constructor(
        private readonly filtersService: FiltersService,
    ) {}

    @ApiBearerAuth()
    @HttpCode(HttpStatus.OK)
    @Get("/get-filters")
    async getFilters(@GetCurrentUserId() userId: string):
Promise<UserFilters> {
        return await this.filtersService.getFilters(userId);
    }

    @ApiBearerAuth()
    @HttpCode(HttpStatus.OK)
    @Put("/update-filters")
    async updateFilters(
        @Body() updateFilterDto: UpdateFiltersDto,
        @GetCurrentUserId() userId: string,
    ): Promise<UserFilters> {

```

```

        return await
this.filtersService.updateFilters(updateFilterDto, userId);
    }
}

```

Лістинг 4.17 – Фрагмент коду контролера FriendsController

```

@Controller("friends")
@UseGuards(JwtAuthGuard)
export class FriendsController {
    constructor(
        private readonly friendsService: FriendsService,
        private readonly friendshipRequestService:
            FriendshipRequestService,
    ) {}

    @ApiBearerAuth()
    @Get()
    @HttpCode(HttpStatus.OK)
    async getFriends(
        @GetCurrentUserId() userId: string,
    ): Promise<UserCore[]> {
        return await this.friendsService.getFriends(userId);
    }

    @Get("/sent-requests")
    @HttpCode(HttpStatus.OK)
    async getSentFriendshipRequests(
        @GetCurrentUserId() userId: string,
    ): Promise<UserCore[]> {
        return await
            this.friendshipRequestService.getSentFriendshipRequests(userId);
    }

    @Get("/received-requests")
    @HttpCode(HttpStatus.OK)
    async getReceivedFriendshipRequests(
        @GetCurrentUserId() userId: string,
    ): Promise<UserCore[]> {
        return await
            this.friendshipRequestService.getReceivedFriendshipRequests(userId);
    }

    @ApiBearerAuth()
    @Post("/request-friendship/:recipientId")
    @HttpCode(HttpStatus.OK)
    async requestFriendship(
        @GetCurrentUserId() userId: string,
        @Param("recipientId") recipientId: string,
    ): Promise<RequestFriendshipResponse> {
        return await this.friendshipRequestService.requestFriendship(userId,
            recipientId);
    }

    @ApiBearerAuth()
    @Post("/accept-friendship/:recipientId")

```

```

@HttpCode(HttpStatus.OK)
async acceptFriendship(
  @GetCurrentUserId() userId: string,
  @Param("recipientId") recipientId: string,
): Promise<AcceptFriendshipResponse> {
  return await
    this.friendshipRequestService.acceptFriendshipRequest(userId,
      recipientId);

```

```

@ApiBearerAuth()
@Post("/reject-friendship/:recipientId")
@HttpCode(HttpStatus.OK)
async rejectFriendship(
  @GetCurrentUserId() userId: string,
  @Param("recipientId") recipientId: string,
): Promise<AcceptFriendshipResponse> {
  return await
    this.friendshipRequestService.rejectFriendshipRequest(userId,
      recipientId);

```

(Ендпоінти для отримання інформації про конкретного друга, видалення із друзів конкретного друга та видалення усіх друзів)

РОЗРОБКА КОМП'ЮТЕРНОЇ СИСТЕМИ ДЛЯ ПОШУКУ ДРУЗІВ ЗА ІНТЕРЕСАМИ ДЛЯ ANDROID

Національний університет "Запорізька політехніка"

Кафедра комп'ютерних систем та мереж

Виконавець: Студент групи КНТ-511 Пономаренко Микола Володимирович

Керівник: Скрупський Степан Юрійович

Рік захисту: 2025

Цілі та Завдання

Створення комп'ютерної системи

Розробити зручну комп'ютерну систему для пошуку друзів за спільними інтересами.

Пошук за уподобаннями

Забезпечити можливість знаходити нові знайомства на основі заданих уподобань, віку та ін.

Полегшення соціальної адаптації

Сприяти розширенню кола спілкування користувачів у цифровому просторі.

Підвищення зручності

Створити інтуїтивно зрозумілий інтерфейс для комфортного користування системою.

Актуальність Розробки

Епідемія самотності

Більше половини підлітків знаходять друзів онлайн, але майже половина людей відчуває дефіцит спілкування, а 30% регулярно переживають самотність щонайменше раз на тиждень. Самотність дедалі частіше визнається не лише емоційною проблемою, а й глобальною кризою громадського здоров'я.

Цифрова трансформація

Сучасні технології значно змінили спосіб, у який ми формуємо соціальні зв'язки. Все більше людей знаходять спільноти онлайн, де спільні інтереси та хобі стають основою для нових знайомств і дружби. Це відкриває можливості для створення інноваційних інструментів знайомства, заснованих не на зовнішності, а на змісті.

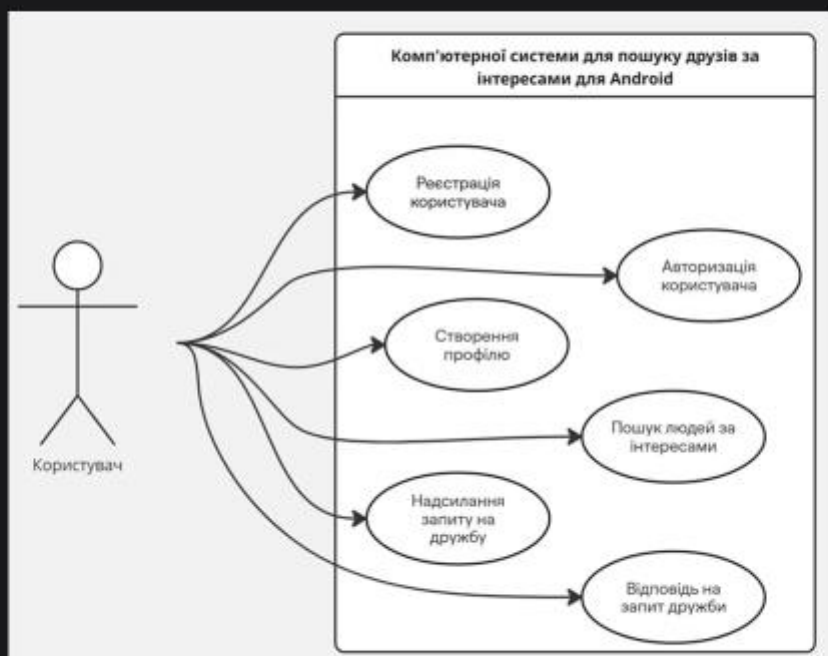
Алгоритмічна ізоляція

Попри доступність соціальних платформ, алгоритми часто формують "інформаційні бульбашки", які лише посилюють ізоляцію. Люди все рідше виходять за межі звичного кола контактів. Розробка застосунку, що дозволяє знаходити нових людей саме за інтересами, є відповіддю на цю цифрову ізоляцію. Такий підхід сприяє відкритому, здоровому спілкуванню та розширенню соціальних горизонтів.

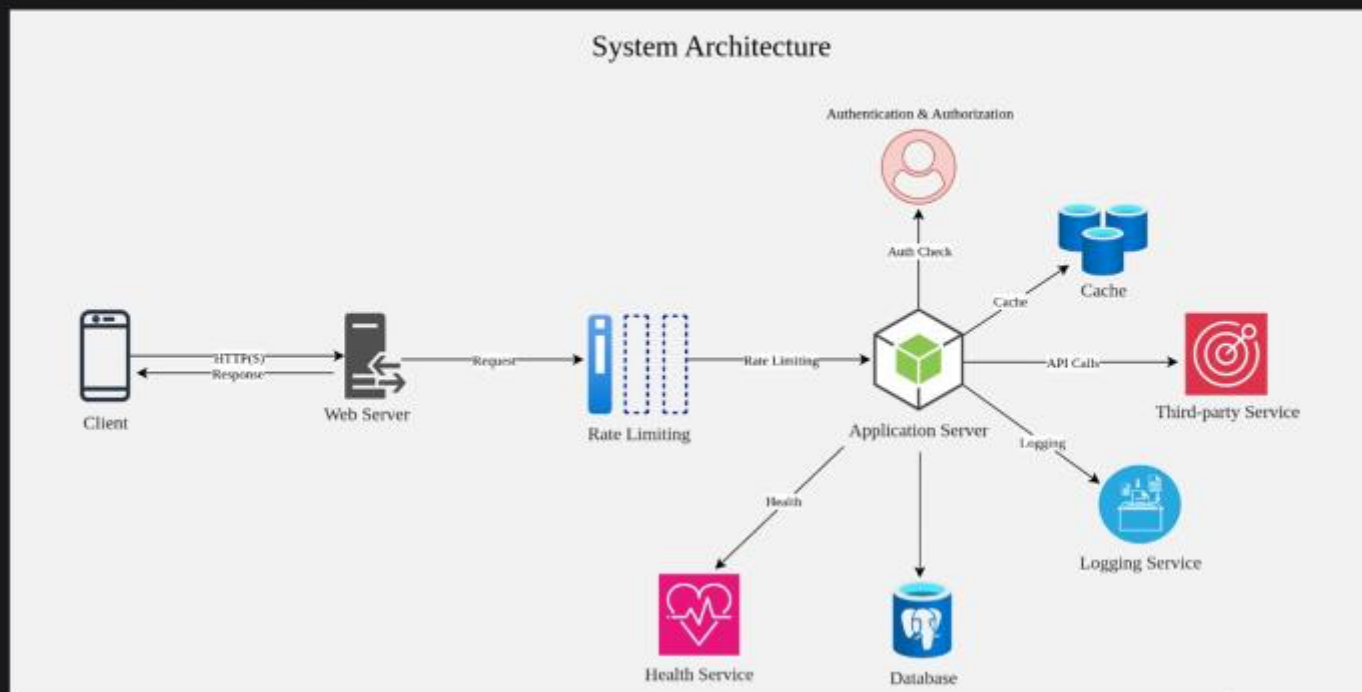
Психологічна підтримка через технології

Технічні рішення в соцмережах та комп'ютерних системах можуть не лише полегшувати знайомства, а й надавати психологічну підтримку, боротися зі стресом і тривогою, підвищувати емоційне благополуччя користувачів.

Діаграма варіантів використання (Use Case)



Архітектура Системи



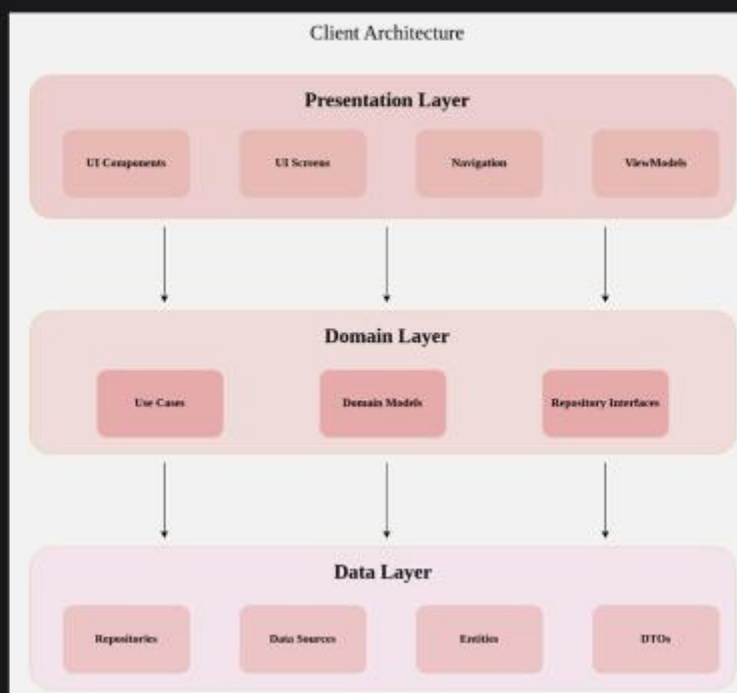
Мережева взаємодія клієнта та сервера

У поточній реалізації клієнт та сервер взаємодіють через HTTP-протокол — обмін даними здійснюється через REST API-запити. Обидві частини запуснені локально, тому шифрування з'єднання поки що не використовується.

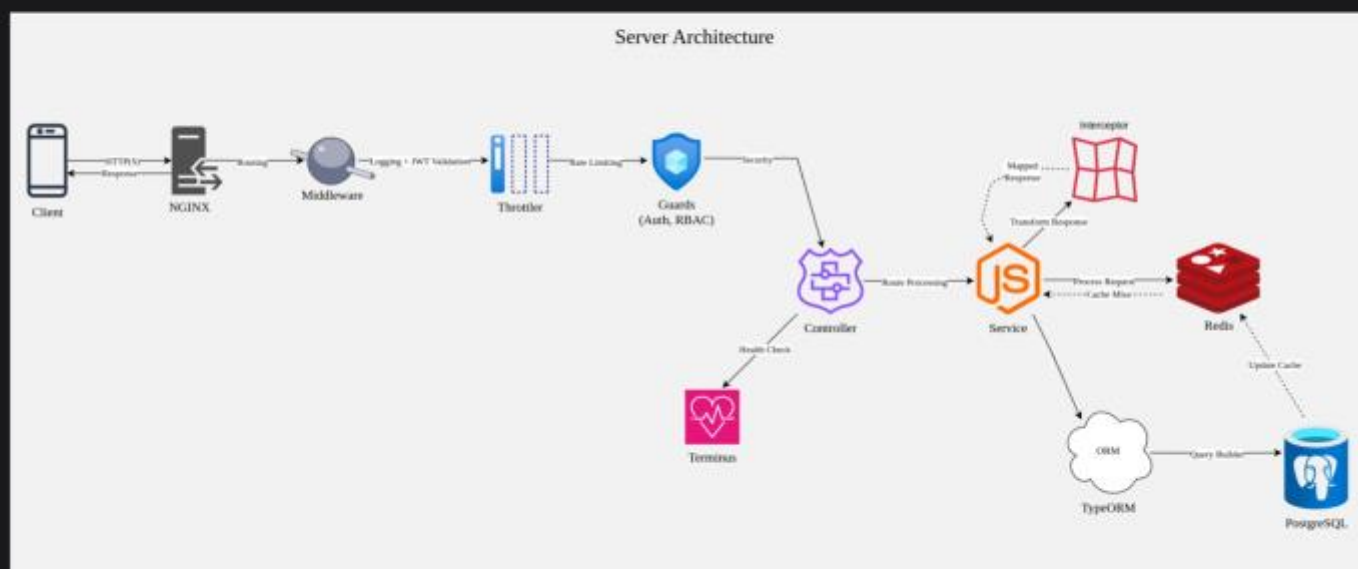
Однак у реальному (production) середовищі необхідно використовувати HTTPS (HTTP Secure) — це HTTP + шифрування через TLS (раніше SSL). Це критично важливо з кількох причин:

- Конфіденційність – запобігає перехопленню чутливих даних (логіни, токени тощо) у відкритих мережах (Wi-Fi, мобільний інтернет).
- Цілісність – гарантує, що дані не були змінені під час передачі (захист від "man-in-the-middle").
- Аутентифікація – клієнт впевнений, що підключений саме до справжнього сервера, а не до підробленого.
- SEO і довіра – пошуковики (Google тощо) дають пріоритет HTTPS-сайтам, а браузери помічають HTTP як "небезпечний".

Клієнтська Архітектура

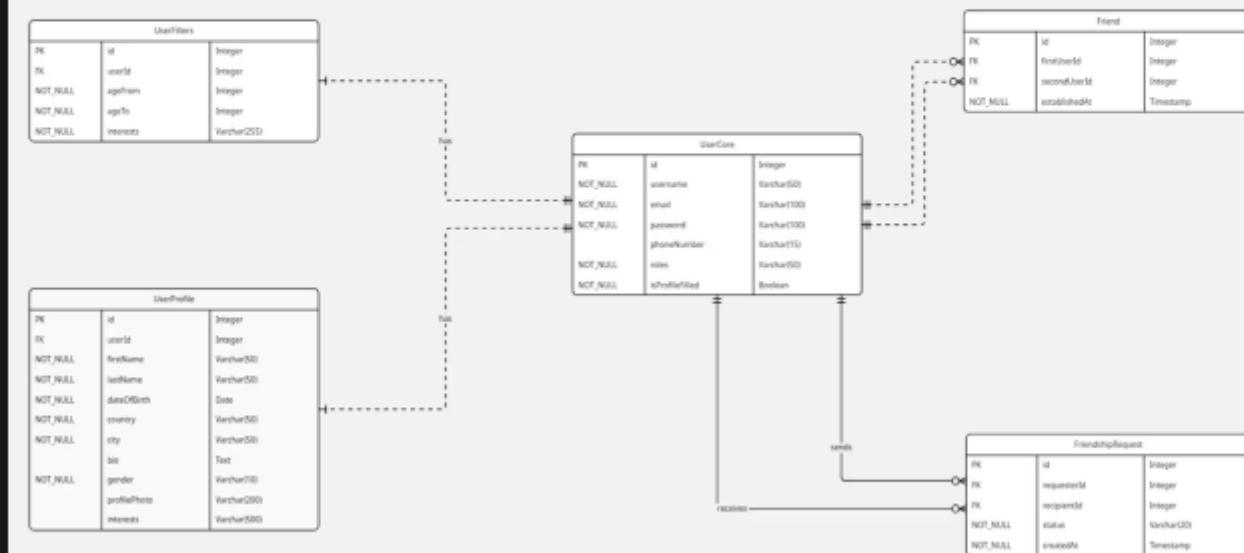


Серверна Архітектура (частина перша)



Серверна Архітектура (частина друга)

ER-diagram



Ключові Модулі

Авторизація (Auth)

Використання NestJS AuthModule та JWT-токенів для підтвердження сесії.

Стрічка (Feed)

Списки друзів і основна сторінка для користувачів.



Профілі користувачів

Обробка реєстрації, автентифікації та збереження даних профілю

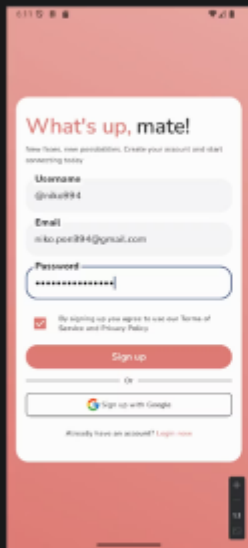
Фільтрація користувачів

Обробка запитів фільтрації за інтересами, віком та формування запитів до БД.

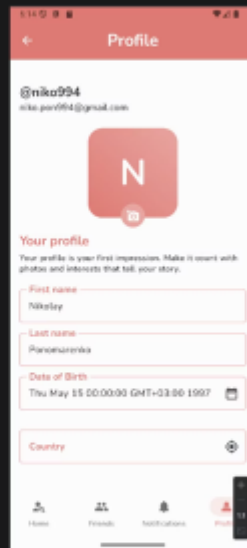
Система друзів

Керування дружніми зв'язками, надсилання/підтвердження запитів у друзі та можливість блокування користувачів

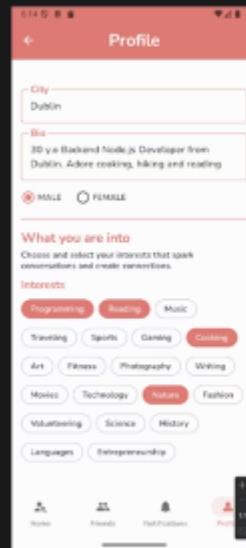
Результати Розробки (частина перша)



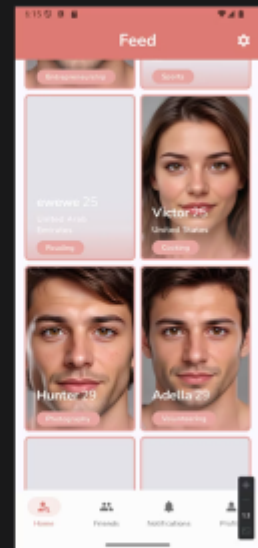
Екран реєстрації користувача для швидкого доступу до застосунку.



Інтерфейс профілю користувача з інформацією та налаштуваннями (частина перша)

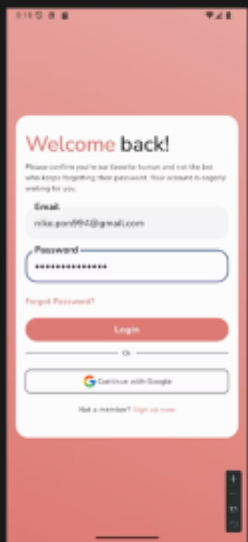


Інтерфейс профілю користувача з інформацією та налаштуваннями (частина друга)

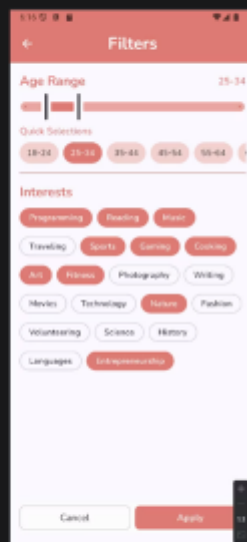


Стрічка — список користувачів, де можна шукати нових друзів.

Результати Розробки (частина друга)



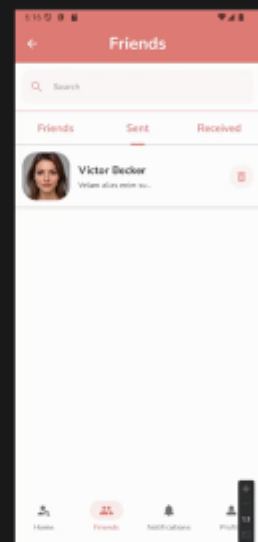
Екран входу до системи для користувача.



Екран фільтрації користувачів за різноманітними критеріями, включаючи інтереси та параметри пошуку.



Екран друзів з розділом списку друзів.



Екран друзів з розділом надісланих запитів на дружбу.

Системні Вимоги

Тип вимоги	Мінімальні вимоги	Рекомендовані вимоги
Версія ОС	Android 8.0 (API рівень 26)	Android 12.0 (API рівень 31)
Оперативна пам'ять	2 ГБ	4 ГБ або більше
Обсяг пам'яті	70 МБ	150 МБ або більше
Процесор	Qualcomm Snapdragon 450, Samsung Exynos 7884, MediaTek Helio P22 або еквівалент	Qualcomm Snapdragon 660, Samsung Exynos 9611, MediaTek Helio G70 або потужніше
Швидкість мережі	Стабільне з'єднання зі швидкістю від 10 Мбіт/с	Стабільне з'єднання зі швидкістю від 25 Мбіт/с
Дозволи	Доступ до камери, сховища	Доступ до камери, сховища

Висновки та Перспективи



Переваги рішення

Монолітна архітектура забезпечила швидку розробку та легке розгортання. Kotlin та Jetpack Compose гарантували високу продуктивність клієнта та швидку розробку UI. NestJS, TypeORM та PostgreSQL забезпечили гнучкість та надійність сервера.



Перспективи масштабування

Систему можна розширювати, виділяючи мікросервіси для авторизації, пошуку або рекомендацій. Можлива інтеграція чатів, мультиплатформенність (iOS-версія) та додавання геолокації.



Ефективність

Запропоноване рішення доводить свою ефективність у досягненні поставленої мети та готове до подальшого розвитку та масштабування.



Подальший розвиток

Планується впровадження машинного навчання для покращення алгоритмів підбору друзів та збільшення залученості користувачів.