

Міністерство освіти і науки України
Державний заклад
«Луганський національний університет імені Тараса Шевченка»

Факультет технологій та інформаційних систем

Кафедра інформаційних технологій та систем

Науменко Олександр Вадимович

**МОБІЛЬНИЙ ДОДАТОК ДЛЯ ФОРМУВАННЯ СТАРТАП -
КОМАНДИ**

кваліфікаційна робота

**здобувача вищої освіти першого (бакалаврського) рівня
освітньої програми «Інженерія програмного забезпечення»
за спеціальністю 121 Інженерія програмного забезпечення**

Особистий підпис _____



Олександр НАУМЕНКО

Науковий керівник _



Володимир ДОНЧЕНКО,
старший викладач
кафедри інформаційних технологій
та систем

Завідувач кафедри _____



Микола СЕМЕНОВ,
кандидат педагогічних наук, доцент
кафедри інформаційних технологій
та систем

Міністерство освіти і науки України
Державний заклад „Луганський національний університет
імені Тараса Шевченка”

Інститут

Факультет технологій та інформаційних
систем

Кафедра, циклова комісія

Інформаційних технологій та систем

Рівень освіти

перший (бакалаврський)

Напрямок підготовки (спеціальність)

121 «Інженерія програмного забезпечення»

(код, назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри ІТС

М.А. Семенов

(підпис)

(ініціали, прізвище)

“ ”

2026 р.

**ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Науменку Олександру Вадимовичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) **Мобільний додаток для формування
стартап - команди**

Керівник кваліфікаційної роботи

Донченко В.Ю.

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

затверджена наказом по університету

Від“ ” 2025 року №_

2. Строк подання студентом проекту (роботи)

3. Вихідні дані до роботи (проекту) у результаті виконання роботи

повинно бути розроблено мобільний додаток для формування
стартап-команди

(визначаються кількісні або (та) якісні показники, яким повинен відповідати об'єкт розробки)

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно
розробити) **АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ**

ПРОЄКТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ SINERGY HUB

РЕАЛІЗАЦІЯ МОБІЛЬНОГО ДОДАТКА

(визначаються назви розділів або (та) перелік питань, які повинні увійти до тексту ПЗ)

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання „_____” _____ 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
	Вибір теми роботи, вивчення наукової літератури, затвердження теми та керівника.	До 15 жовтня	
	Аналіз літературних джерел за темою роботи. Розробка та апробація методики дослідно-експериментальної роботи. Подання структури теоретичної частини роботи та плану експериментальних досліджень.	Другий тиждень листопада (10 листопада)	
	Робота над теоретичною частиною. Подання теоретичної частини роботи для першого читання науковим керівником.	До 15 грудня	
	Усунення зауважень, урахування рекомендацій наукового керівника. Подання теоретичної частини роботи на друге читання.	До 28 січня	
	Проведення експериментальної роботи. Поетапний аналіз та обговорення її результатів. Перевірка стану виконання роботи.	Перший тиждень березня	
	Урахування рекомендацій наукового керівника, усунення недоліків, підготовка варіанта роботи до передзахисту. Розробка презентації.	До 31 березня	
	Попередній захист роботи на кафедрі	квітень	
	Доопрацювання роботи з урахуванням рекомендацій після передзахисту. Подання роботи науковому керівникові та рецензентові на підготовку відгуку та рецензії	За 10 днів до державної атестації	
	Подання на кафедру остаточного варіанта роботи, переплетеного та підписаного автором, науковим керівником і рецензентом.	За 5 днів до державної атестації	

Студент

підпис

О.В. Науменко

(ініціали, прізвище)

Керівник проекту (роботи)

підпис

В.Ю. Донченко

АНОТАЦІЯ

Науменко О. В.

Тема: Мобільний додаток для формування стартап - команди

Спеціальність: 121 "Інженерія програмного забезпечення"

Установа: ЛНУ імені Тараса Шевченка, 2026 р.

Бакалаврська робота містить: 69 с., 16 рис., 4 табл., 3 додат., 36 джерел.

Об'єкт дослідження – процес формування та організації стартап-команд у цифровому середовищі.

Предмет дослідження – методи, засоби та програмні технології розробки мобільних застосунків для пошуку та взаємодії учасників стартап-проектів.

Мета роботи – розробка кросплатформенного мобільного застосунку Synergy Hub для формування стартап-команд з використанням мови програмування Dart та фреймворку Flutter.

Результати роботи. У дипломній роботі досліджено предметну область формування стартап-команд, проаналізовано існуючі програмні рішення та їх недоліки. Спроектовано та реалізовано мобільний застосунок Synergy Hub з використанням фреймворку Flutter та мови Dart. Застосовано архітектурний підхід Clean Architecture з розподілом на рівні Domain, Data та Presentation. Для керування станом використано патерн BLoC. Серверну частину реалізовано на основі Firebase Cloud Functions (TypeScript). Впроваджено систему свайп-взаємодії, алгоритм матчингу з тривимірною оцінкою сумісності, систему реального часу чатів та push-сповіщень.

Висновки. У результаті розробки створено працездатний мобільний застосунок Synergy Hub, який забезпечує зручний пошук однодумців, формування стартап-команд та підтримку комунікації між учасниками .

Ключові слова: МОБІЛЬНИЙ ЗАСТОСУНОК, СТАРТАП-КОМАНДА, FLUTTER, DART, FIREBASE, CLEAN ARCHITECTURE, BLOC, СВАЙП-СИСТЕМА, МАТЧИНГ, PUSH-СПОВІЩЕННЯ , CLOUD FUNCTIONS.

ABSTRACT

Naumenko O. V.

Theme: Mobile application for startup team formation

Speciality: 121 "Software Engineering"

Institution: Luhansk Taras Shevchenko National University, 2026.

Bachelor thesis contains: 69 pages, 16 figures, 4 tables, 3 appendices, 36 references.

Research object – the process of forming and organizing startup teams in a digital environment.

Research subject – methods, tools and software technologies for developing mobile applications for searching and interacting with startup project participants.

Purpose – development of a cross-platform mobile application Synergy Hub for startup team formation using Dart programming language and Flutter framework.

Results. The thesis explores the domain of startup team formation, analyzes existing software solutions and their limitations. A cross-platform mobile application Synergy Hub has been designed and implemented using the Flutter framework and Dart language. Clean Architecture approach has been applied with Domain, Data, and Presentation layers. The BLoC pattern is used for state management. Server-side logic is implemented using Firebase Cloud Functions (TypeScript). A swipe interaction system, matching algorithm with three-dimensional compatibility scoring, real-time chat system, and push notifications have been implemented.

Conclusions. As a result, a functional mobile application Synergy Hub has been developed that provides convenient search for like-minded people, startup team formation, and communication support between project participants.

Keywords: MOBILE APPLICATION, STARTUP TEAM, FLUTTER, DART, FIREBASE, CLEAN ARCHITECTURE, BLOC, SWIPE SYSTEM, MATCHING, PUSH NOTIFICATIONS, CLOUD FUNCTIONS.

ІТС. ІПЗ4.0126-ВП
ВІДОМІСТЬ ПРОЄКТУ. МОБІЛЬНИЙ ДОДАТОК ДЛЯ
ФОРМУВАННЯ СТАРТАП-КОМАНДИ.

[illegible]

Міністерство освіти і науки України
Державний заклад «Луганський національний університет
імені Тараса Шевченка»

Факультет

Факультет технологій та інформаційних систем

(повна назва)

Кафедра

Інформаційних технологій та систем

(повна назва)

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання програмної розробки (ПР) :

"МОБІЛЬНИЙ ДОДАТОК ДЛЯ ФОРМУВАННЯ СТАРТАП-КОМАНДИ"

ІТС. ІПЗ4.0126-02-ТЗ

ПОГОДЖЕНО

Керівник кваліфікаційної роботи

Володимир ДОНЧЕНКО

“ _____ ” 2026р.

ВИКОНАВЕЦЬ

Студент групи 4 ІПЗ МС

Олександр НАУМЕНКО

“ _____ ” 2026р.

ЗМІСТ

1. ЗАГАЛЬНІ ПОЛОЖЕННЯ.....	3
1.1. Повне найменування системи та її умовне позначення	3
1.2. Найменування організації-замовника та організації-учасника робіт	3
1.3. Перелік документів, на підставі яких створюється система	3
1.4. Планові терміни початку і закінчення роботи зі створення системи	3
2. ПРИЗНАЧЕННЯ І ЦІЛІ СТВОРЕННЯ СИСТЕМИ	4
2.1. Призначення системи	4
2.2. Цілі створення системи	4
3. ХАРАКТЕРИСТИКА ОБ'ЄКТА АВТОМАТИЗАЦІЇ	4
4. ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	5
4.1. Вимоги до функціональних характеристик.....	5
4.2. Вимоги до надійності.....	5
4.3. Вимоги до складу і параметрів технічних засобів.....	6
5. СТАДІЇ І ЕТАПИ РОЗРОБКИ	6
6. ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ СИСТЕМИ	7
6.1. Види випробувань	7
6.2. Необхідні вимоги для впровадження ПР і завершення робіт.....	7
6.3. Перелік звітних документів, необхідних для прийняття етапів роботи.....	7
6.4. Загальний перелік до прийому звітних документів, макетів, експериментальних зразків	8
6.5. Тестування ПР	8
7. ПОРЯДОК ВНЕСЕННЯ ЗМІН ДО ТЕХНІЧНЕ ЗАВДАННЯ, ЩО ЗАТВЕРДЖЕНО	8

1. ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1. Повне найменування системи та її умовне позначення

Повна назва системи: Мобільний додаток для формування стартап-команди.

Коротке найменування системи: «Synergy Hub».

1.2. Найменування організації-замовника та організації-учасника робіт

Замовником є кафедра Інформаційних технологій та систем Державного закладу "Луганський національний університет імені Тараса Шевченка" (далі за текстом — Замовник). Адреса замовника: вул. Віктора Новікова, 2, м. Лубни, Полтавська область, 37500

Розробник сервісу — студент групи ПЗ-4 кафедри Інформаційних технологій та систем Державного закладу "Луганський національний університет імені Тараса Шевченка" Науменко Олександр Вадимович.

1.3. Перелік документів, на підставі яких створюється система

При розробці системи і створення проектно-експлуатаційної документації Виконавець повинен керуватися вимогами наступних нормативних документів:

- ДСТУ 19.201-78. Технічне завдання. Вимоги до змісту і оформлення;
- ДСТУ 34.601-90. Комплекс стандартів на автоматизовані системи. Автоматизовані системи. Стадії створення;
- ДСТУ 34.201-89. Інформаційні технології. Комплекс стандартів на автоматизовані системи. Види, комплексність і позначення документів при створенні автоматизованих систем.

1.4. Планові терміни початку і закінчення роботи зі створення системи

Плановий термін початку роботи над створенням мобільного додатку для формування стартап-команди – 1 грудня 2025 року.

Плановий термін по закінченню роботи над створенням мобільного додатку для формування стартап-команди – 22 травня 2026 року.

2. ПРИЗНАЧЕННЯ І ЦІЛІ СТВОРЕННЯ СИСТЕМИ

2.1. Призначення системи

Мобільний додаток призначений для підтримки процесу формування стартап-команд та комунікації між їхніми учасниками.

2.2. Цілі створення системи

Цілями розробки даної системи є:

- полегшення процесу пошуку партнерів та формування стартап-команд за рахунок свайп-перегляду профілів користувачів, оцінки сумісності за спеціалізацією та досвідом, а також автоматичного створення чату між учасниками після взаємного матчу.

Для досягнення поставлених цілей необхідно розв'язати наступні задачі:

- реєстрація та автентифікація користувачів (email/password, Google OAuth) із багатоетапним онбордингом для збору інформації про спеціалізацію, досвід, навички та роль у команді;
- перегляд профілів інших користувачів у форматі свайп-карток з фільтрацією за спеціалізацією та досвідом;
- автоматичне створення чату після взаємного матчу та обмін текстовими повідомленнями між учасниками стартап-команди у реальному часі.

3. ХАРАКТЕРИСТИКА ОБ'ЄКТА АВТОМАТИЗАЦІЇ

Для того, щоб користуватися системою, користувач повинен бути зареєстрованим та авторизованим. Для кожного користувача створюється єдиний тип акаунта учасника стартап-команди, у якому під час багатоетапного онбордингу заповнюються спеціалізація, рівень досвіду, ключові навички та бажана роль у проєкті.

Об'єктом автоматизації є процес пошуку учасників стартап-команди, фіксації взаємних рішень (лайк/дизлайк) та обміну повідомленнями між користувачами після матчу.

4. ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Вимоги до функціональних характеристик

Мобільний додаток повинен виконувати роль комунікатора між учасниками стартап-проектів — від моменту пошуку партнера до обміну повідомленнями всередині команди.

Система має виконувати наступні функції для всіх типів користувачів:

- система повинна надавати користувачу можливість створити персональний акаунт, який дозволить переглядати профілі інших учасників, фіксувати рішення та спілкуватися у чатах після матчу.

Для типу користувача «Учасник стартап-команди» система повинна виконувати наступні функції:

- 1) багатоетапний онбординг для збору даних про спеціалізацію, досвід, навички та роль у команді;
- 2) перегляд профілів інших користувачів у форматі свайп-карток;
- 3) фіксація рішень користувача (лайк/дизлайк) та збереження історії взаємодій;
- 4) автоматичне створення чату між учасниками після взаємного матчу;
- 5) обмін текстовими повідомленнями у реальному часі;
- 6) отримання push-сповіщень про нові матчі та повідомлення;
- 7) редагування власного профілю користувача;
- 8) фільтрація користувачів за спеціалізацією та досвідом.

4.2. Вимоги до надійності

Система повинна функціонувати безвідмовно незважаючи на наявність ймовірних дефектів, які можуть проявлятися під час експлуатації. Виправлення таких дефектів повинно виконуватися на етапах розробки, бета-тестування та в процесі введення в дію і в межах промислової експлуатації.

Відмова програмного забезпечення сервісу не повинна призводити до руйнувань даних в інформаційних сховищах.

Будь-які аварійні ситуації повинні бути негайно задокументовані і надіслані розробникам задля усунення причин збою в роботі системи.

4.3. Вимоги до складу і параметрів технічних засобів

Коректне функціонування даного програмного продукту на стороні користувача може бути гарантоване тільки за наступних умов:

- наявність мобільного пристрою з наступними технічними характеристиками:
 1. операційна система Android 8.0 (API 26) або новіша, чи iOS 13.0 або новіша;
 2. об'єм оперативної пам'яті не менше 2 ГБ;
 3. не менше 200 МБ вільного місця у внутрішній пам'яті пристрою;
 4. підключення до мережі Інтернет (Wi-Fi або мобільний інтернет) для роботи з серверною частиною та отримання push-сповіщень.
- сенсорний екран з діагоналлю не менше 4,7 дюйма та роздільною здатністю не нижче 720 × 1280 пікселів.

5. СТАДІЇ І ЕТАПИ РОЗРОБКИ

Основні етапи виконання робіт з розробки мобільного додатку для формування стартап-команди:

№ з/п	Назва етапу роботи	Термін виконання
1.	Підготовка технічного завдання на розробку програмного продукту	
2.	Розробка сценарію роботи	
3.	Технічне проектування – функціональність, модулі, задачі, цілі тощо.	
4.	Узгодження з керівником інтерфейсу користувача	
5.	Розробка інформаційного забезпечення	
6.	Розробка програмного забезпечення	
7.	Налагодження програми	
8.	Тестування програми	
9.	Здача готового програмного продукту замовнику	

6. ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ СИСТЕМИ

6.1. Види випробувань

Задля перевірки правильності роботи програмного продукту буде проведено функціональне тестування. В ході тестування буде виконано перевірку всіх функціональних характеристик мобільного-застосунку. Також, система буде перевірена на відмовостійкість шляхом виконання некоректних дій користувачем. Окремими тестами буде перевірена безпека системи в цілому.

6.2. Необхідні вимоги для впровадження ПР і завершення робіт

Оцінка результатів розробки і доцільність її продовження здійснюється замовником за поданням наступних матеріалів:

- встановлено програмний комплекс на ЕОМ замовника;
- перелік файлів на резервному носії;
- короткий опис роботи ПР і опис всіх файлів, які необхідні для роботи ПР.
- перелік документів:
 - Технічне завдання
 - Пояснювальна записка
 - Програма і методика тестування
 - Керівництво користувача

6.3. Перелік звітних документів, необхідних для прийняття етапів роботи

- короткий опис результатів етапу у вигляді анотованого звіту (для 1 та 2 етапів);
- частковий програмний комплекс на ЕОМ замовника згідно календарного плану робіт;
- акт приймання продукції.

Звітні матеріали подаються у вигляді звітів на папері по ГОСТ 7.32-91

6.4. Загальний перелік до прийому звітних документів, макетів, експериментальних зразків

До прийому пред'являються: акт здачі-приймання продукції, акт впровадження ПР.

6.5. Тестування ПР

Тестування виконується в "Програми і методики тестування", яка розробляється виконавцем і затверджується замовником

7. ПОРЯДОК ВНЕСЕННЯ ЗМІН ДО ТЕХНІЧНЕ ЗАВДАННЯ, ЩО ЗАТВЕРДЖЕНО

Дане технічне завдання може уточнюватися в процесі розробки ПР при узгодженні сторін з оформленням доповнень до ТЗ.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЗ «ЛУГАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА»

Факультет технологій та інформаційних систем

(назва факультету, інституту)

Інформаційних технологій та систем

(назва кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

БАКАЛАВРА

(освітньо-кваліфікаційний рівень)

на тему: МОБІЛЬНИЙ ДОДАТОК ДЛЯ ФОРМУВАННЯ СТАРТАП-КОМАНДИ

Виконав: студент 4 курсу

Спеціальності 121 «Інженерія програмного
забезпечення»

(шифр і назва напрямку підготовки, спеціальності)

Науменко О.В.

(прізвище та ініціали)

Керівник

Донченко В.Ю.

(прізвище та ініціали)

Рецензент

Козуб Ю.Г.

(прізвище та ініціали)

Лубни 2026

ЗМІСТ

ВСТУП.....	11
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА	
ЗАДАЧІ.....	13
1.1. Поняття стартап-команд та особливості їх формування	13
1.2. Аналіз існуючих мобільних сервісів для пошуку партнерів	15
1.2.1. Сервіси знайомств.....	15
1.2.2. Професійні соціальні мережі	15
1.2.3. Платформи для спільнот та подій	16
1.2.4. Узагальнений аналіз існуючих рішень	16
1.3. Проблеми та недоліки наявних рішень.....	17
1.4. Обґрунтування вибору концепції застосунку Synergy Hub	19
1.5. Формування вимог до програмного продукту	20
Висновки до розділу	22
РОЗДІЛ 2. ПРОЄКТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ	
SYNERGY HUB.....	23
2.1. Вибір мови програмування та технологій	23
2.2. Загальна архітектура програмного забезпечення	25
2.3. Використання Clean Architecture (Domain, Data, Presentation).....	27
2.4. Проєктування бази даних на основі Firebase Firestore	29
2.5. Моделювання взаємодії користувачів у системі	31
2.6. Проєктування інтерфейсу користувача	34
2.7. Розробка онбордингу та збору інформації про користувача.....	35
2.8. Проєктування механізму пошуку та свайп-системи	37
2.9. Проєктування системи чатів та повідомлень	40
Висновки до розділу	42
РОЗДІЛ 3. РЕАЛІЗАЦІЯ МОБІЛЬНОГО ДОДАТКА	44
3.1. Архітектура реалізації клієнтської частини на Flutter та Dart.....	44
3.2. Використання патерну Repository, Unit of Work та Dependency	
Injection.....	46

3.3. Реалізація анімованого фону з використанням GLSL-шейдерів.....	48
3.4. Реалізація системи автентифікації користувачів	49
3.5. Реалізація Discover-екрану та механізму свайпів	51
3.6. Реалізація чатів та збереження повідомлень	53
3.7. Реалізація бізнес-логіки на основі патерну Use Case	55
3.8. Інтеграція з Firebase Firestore та Firestore Security Rules	56
3.9. Реалізація серверної логіки на основі Firebase Cloud Functions (TypeScript)	58
3.9.1. Функція матчингу користувачів у транзакції (matchUser)	58
3.9.2. Обробка нових повідомлень та розсилка push-сповіщень (onNewMessage).....	60
3.9.3. Синхронізація профілю користувача в чатах (onUserUpdate)....	61
3.10. Інтеграція Firebase Cloud Messaging для push-сповіщень.....	61
Висновки до розділу	63
ВИСНОВКИ	65
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	68
ДОДАТОК А.....	71
ДОДАТОК Б.....	73
ДОДАТОК В.....	74

ВСТУП

У сучасних умовах розвитку цифрових технологій та глобалізації економіки стартап-проекти відіграють важливу роль у формуванні інноваційного середовища та створенні конкурентоспроможних продуктів. Успішність стартапу значною мірою залежить не лише від якості ідеї, але й від професійного рівня, мотивації та узгодженості дій членів команди.

Процес формування стартап-команди є складним та багатофакторним завданням, яке передбачає пошук фахівців з необхідними компетенціями, оцінку їх професійних і особистісних якостей, а також забезпечення ефективної комунікації між учасниками проекту. Традиційні методи пошуку партнерів, зокрема через особисті знайомства, соціальні мережі або професійні платформи, не завжди забезпечують швидкий та якісний підбір команди.

З розвитком мобільних технологій з'являється можливість створення спеціалізованих програмних засобів, орієнтованих на підтримку процесу формування стартап-команд. Мобільні застосунки дозволяють забезпечити доступність сервісу, зручність взаємодії користувачів та оперативний обмін інформацією незалежно від їх місцезнаходження.

Даний дипломний проєкт націлений на розробку ефективного інструменту для пошуку однодумців, співзасновників та учасників стартап-проектів із використанням сучасних інформаційних технологій. Створення мобільного застосунку для формування стартап-команд сприятиме розвитку підприємницької діяльності, підвищенню рівня інноваційності та оптимізації процесів командної взаємодії.

Об'єкт дослідження: мобільний додаток для формування стартап-команди.

Предмет дослідження: Методи, засоби та програмні технології розробки мобільних застосунків для пошуку та взаємодії учасників стартап-проектів.

Мета дослідження: проєктування та реалізація мобільного застосунку для формування стартап-команд

Методи дослідження: порівняння та аналіз.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

- проаналізувати предметну область та існуючі програмні рішення;
- визначити вимоги до функціональних можливостей застосунку;
- спроектувати архітектуру програмного забезпечення;
- розробити клієнтську частину мобільного застосунку;
- реалізувати механізми автентифікації та зберігання даних;
- створити систему пошуку та взаємодії користувачів;
- реалізувати серверну логіку на основі Firebase Cloud Functions;

Практичною цінністю роботи є розробка мобільного застосунку для формування стартап-команд.

В першому розділі проаналізовано предметну область формування стартап-команд та виконано порівняльний аналіз існуючих мобільних сервісів (Tinder, Bumble, LinkedIn, Meetup). Обґрунтовано концепцію застосунку Synergy Hub та сформовано функціональні й нефункціональні вимоги до програмного продукту.

В другому розділі виконано проєктування мобільного застосунку. Обґрунтовано вибір технологічного стеку, описано загальну архітектуру, спроектовано базу даних та змодельовано взаємодію користувачів у системі матчингу. Розроблено проєкти інтерфейсу, онбордингу, свайп-системи та чатів.

У третьому розділі детально описано реалізацію основних модулів застосунку клієнтської частини на Flutter із застосуванням патернів; серверної логіки; системи автентифікації; push-сповіщень та GLSL-шейдерів для анімованого фону. Наведено специфікацію функцій `matchUser`, `onNewMessage` та `onUserUpdate`.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Поняття стартап-команд та особливості їх формування

Стартап-команда — це група осіб, об'єднаних спільною ідеєю створення інноваційного продукту або сервісу з високим потенціалом розвитку та масштабування. Основною метою такої команди є реалізація бізнес-ідеї в умовах обмежених ресурсів, високої конкуренції та невизначеності.

Формування стартап-команди є одним із ключових факторів успішності проєкту, оскільки саме людський капітал визначає якість прийняття рішень, швидкість розвитку та адаптацію до змін ринку.

До основних характеристик ефективної стартап-команди належать:

- **комплементарність навичок** — поєднання технічних, управлінських, маркетингових та фінансових компетенцій;
- **спільність цінностей і бачення** — розуміння цілей проєкту та готовність працювати на результат;
- **гнучкість та адаптивність** — здатність швидко реагувати на зміни;
- **високий рівень мотивації** — зацікавленість у розвитку продукту;
- **комунікаційна ефективність** — вміння взаємодіяти та вирішувати конфлікти.

Процес формування стартап-команди зазвичай включає такі етапи:

- визначення ідеї та цілей проєкту;
- аналіз необхідних компетенцій;
- пошук потенційних учасників;
- оцінка професійних та особистісних якостей;
- формування ролей у команді;
- налагодження внутрішніх процесів взаємодії.

У сучасних умовах пошук учасників команди все частіше відбувається за допомогою цифрових платформ, соціальних мереж та мобільних

застосунків. Це дозволяє залучати фахівців з різних регіонів та галузей, проте ускладнює процес оцінювання їхньої сумісності.

Саме тому актуальним є створення спеціалізованих програмних рішень, орієнтованих не лише на пошук людей, а й на формування ефективних стартап-команд з урахуванням професійних і особистісних характеристик.

За даними досліджень, проведених CB Insights, серед основних причин невдач стартапів 23% припадають на невідповідний склад команди. Це підтверджує критичну важливість правильного підбору учасників на початкових етапах розвитку проєкту.

Стартап-команди відрізняються від традиційних робочих груп кількома суттєвими ознаками. По-перше, вони функціонують в умовах високої невизначеності, коли бізнес-модель ще не валідована, а ринок не до кінця визначений. По-друге, учасники стартап-команди часто поєднують кілька ролей, що вимагає широкого спектру компетенцій. По-третє, мотивація учасників базується не лише на фінансовій винагороді, але й на вірі в ідею та бажанні створити інноваційний продукт.

Типова стартап-команда включає такі ключові ролі: технічний співзасновник (СТО), який відповідає за технічну реалізацію продукту; продуктовий менеджер, який визначає стратегію розвитку; дизайнер, який забезпечує користувацький досвід; маркетолог, який відповідає за просування. Залежно від специфіки проєкту, склад команди може варіюватися.

Окремою проблемою є оцінка сумісності потенційних учасників. Традиційні методи, такі як співбесіди та рекомендації, мають обмежену ефективність у контексті стартапів, оскільки не враховують специфіку роботи в умовах невизначеності. Необхідні нові інструменти, які дозволять оцінювати не лише професійні навички, але й співпадіння бачення, цілей та робочого стилю.

1.2. Аналіз існуючих мобільних сервісів для пошуку партнерів

На сьогодні існує значна кількість мобільних сервісів, які опосередковано використовуються для пошуку партнерів, співзасновників та однодумців. Проте більшість із них не орієнтована безпосередньо на формування стартап-команд.

1.2.1. Сервіси знайомств

До цієї групи належать застосунки, спочатку створені для особистих знайомств, але які іноді використовуються з професійною метою. Прикладами є Tinder та Bumble.

Основні особливості:

- механізм свайпів для вибору користувачів;
- мінімальна інформація про професійні навички;
- орієнтація на особисті інтереси.

Недоліком таких сервісів є відсутність інструментів для оцінки професійної сумісності та командної взаємодії.

Варто зазначити, що сервіс Bumble має окремий режим Bumble Bizz, спрямований на професійні знайомства. Проте цей режим зберігає загальну концепцію додатку і не надає спеціалізованих інструментів для оцінки технічних навичок або формування проєктних команд. Профілі у Bumble Bizz містять лише загальну інформацію про посаду та компанію, без деталізації технологічного стеку чи рівня досвіду.

Окремо слід виділити додаток CoFoundersLab, який позиціонується як платформа для пошуку співзасновників. Проте цей сервіс доступний переважно у веб-форматі, має обмежену мобільну версію та не надає автоматизованого алгоритму матчингу, покладаючись на ручний пошук та фільтрацію.

1.2.2. Професійні соціальні мережі

Професійні платформи дозволяють знаходити фахівців різного профілю. Найбільш поширеним прикладом є LinkedIn.

Переваги:

- детальний опис досвіду та навичок;
- можливість налагодження ділових контактів;
- доступ до широкої аудиторії спеціалістів.

Недоліки:

- відсутність механізмів швидкого підбору команди;
- складність пошуку людей саме для стартап-проектів;
- орієнтація на класичне працевлаштування.

1.2.3. Платформи для спільнот та подій

До цієї категорії належать сервіси для організації зустрічей та заходів, наприклад Meetup.

Основні можливості:

- пошук спільнот за інтересами;
- участь у тематичних подіях;
- живе спілкування.

Недоліки:

- відсутність системи персонального підбору партнерів;
- обмежена підтримка онлайн-взаємодії;
- відсутність інтегрованих чатів для проектної роботи.

1.2.4. Узагальнений аналіз існуючих рішень

Аналіз наявних мобільних сервісів дозволяє зробити такі висновки:

- більшість платформ не спеціалізується на стартап-командах;
- відсутні алгоритми оцінки сумісності учасників;
- недостатньо інструментів для довготривалої співпраці;
- переважає орієнтація на знайомства або працевлаштування;
- обмежена підтримка командної комунікації.

Таким чином, наявні сервіси лише частково задовольняють потреби стартап-спільноти та не забезпечують комплексного підходу до формування команд. Це обґрунтовує доцільність розробки спеціалізованого мобільного

застосунку Synergy Hub, орієнтованого на професійний підбір учасників, підтримку комунікації та розвиток стартап-проектів.

Для більш наочного порівняння основних характеристик існуючих сервісів та запропонованого рішення Synergy Hub складено порівняльну таблицю (таблиця 1.1). Аналіз показує, що Synergy Hub є єдиним рішенням, яке поєднує всі ключові функціональні можливості, необхідні для формування стартап-команд: інтуїтивний свайп-механізм, детальний професійний профіль, автоматизований алгоритм оцінки сумісності та вбудовану систему комунікації.

Таблиця 1.1.

Порівняльний аналіз існуючих сервісів

Критерій	Tinder/Bumble	LinkedIn	Meetup	Synergy Hub
Свайп-механізм	Так	Ні	Ні	Так
Професійний профіль	Ні	Так	Частково	Так
Алгоритм сумісності	Ні	Ні	Ні	Так
Вбудований чат	Так	Так	Ні	Так
Орієнтація на стартапи	Ні	Частково	Частково	Так
Push-сповіщення	Так	Так	Так	Так

1.3. Проблеми та недоліки наявних рішень

На підставі проведеного аналізу можна виділити ключові проблеми існуючих програмних рішень, які ускладнюють процес формування стартап-команд:

Відсутність спеціалізації. Жоден із розглянутих сервісів не орієнтований безпосередньо на формування стартап-команд. Сервіси

знайомств не враховують професійні характеристики, а професійні мережі фокусуються на класичному працевлаштуванні.

Відсутність алгоритмів оцінки сумісності. Жодна з проаналізованих платформ не пропонує автоматизованої оцінки сумісності учасників за такими параметрами, як рівень досвіду, технічні навички та бачення проєкту. Підбір здійснюється виключно вручну.

Обмеженість комунікаційних інструментів. Більшість платформ не надає інтегрованих засобів для проєктної комунікації. Навіть за наявності чатів, вони не адаптовані для потреб командної співпраці.

Відсутність онбордингу. Існуючі сервіси не збирають детальну інформацію про професійний профіль користувача на етапі реєстрації, що ускладнює подальший підбір партнерів.

Обмежена масштабованість. Більшість рішень не передбачає гнучкої системи фільтрації та персоналізованого пошуку за спеціалізацією, роллю чи рівнем досвіду.

Відсутність автоматичного створення робочого простору. Після знаходження потенційного партнера користувачі змушені переходити на зовнішні платформи для комунікації (Telegram, Slack, Discord), що ускладнює процес та збільшує ймовірність втрати контакту.

Відсутність зворотного зв'язку щодо якості матчів. Жодна з платформ не надає інформації про ступінь сумісності між учасниками, що змушує користувачів самостійно оцінювати потенційних партнерів без будь-яких кількісних показників.

Таким чином, існуючі програмні рішення мають системні недоліки, які не дозволяють ефективно використовувати їх для формування стартап-команд. Це підтверджує необхідність створення спеціалізованого інструменту, орієнтованого саме на цю задачу.

1.4. Обґрунтування вибору концепції застосунку Synergy Hub

На основі проведеного аналізу існуючих рішень та визначених проблем було сформовано концепцію мобільного застосунку Synergy Hub, який поєднує переваги різних типів платформ та усуває їх ключові недоліки.

Основні принципи концепції:

- **свайп-механізм** для швидкого та інтуїтивного перегляду профілів потенційних партнерів, запозичений з сервісів знайомств, але адаптований під професійний контекст;
- **детальний професійний профіль** з інформацією про спеціалізацію, рівень досвіду, навички та бачення проєкту, подібно до LinkedIn, але зі спрощеним введенням;
- **алгоритм матчингу** з тривимірною оцінкою сумісності (досвід, навички, бачення), який автоматично визначає ступінь відповідності між користувачами;
- **інтегрована система чатів** для комунікації між учасниками, яка створюється автоматично після взаємного матчу;
- **багатоетапний онбординг** для збору інформації про користувача, що забезпечує якісний підбір команди.

Назва застосунку — Synergy Hub — відображає його основну місію: створення середовища, де синергія між учасниками стартап-команди досягається завдяки ефективному підбору та комунікації.

Вибір свайп-механізму як основного способу взаємодії обумовлений кількома факторами. По-перше, дослідження у сфері UX-дизайну демонструють, що свайп-жести є найбільш інтуїтивними для мобільних пристроїв. По-друге, бінарне рішення (лайк/дизлайк) знижує когнітивне навантаження на користувача, що особливо важливо при перегляді великої кількості профілів. По-третє, взаємне підтвердження інтересу створює більш якісну базу для початку комунікації, ніж одностороннє звернення.

Тривимірна модель оцінки сумісності є ключовою інновацією Synergy Hub. На відміну від існуючих рішень, які покладаються виключно на ручний

підбір, алгоритм автоматично розраховує три незалежні показники: співвідношення рівнів досвіду, перетин технічних навичок та співпадіння бачення проєкту. Кожен показник оцінюється за шкалою від 0 до 10, що дозволяє користувачам швидко оцінити потенціал співпраці.

Архітектурно застосунок розроблено з урахуванням можливості масштабування. Використання Firebase як Backend-as-a-Service забезпечує автоматичне масштабування серверної частини, а фреймворк Flutter дозволяє підтримувати множину платформ (Android, iOS, Web) з єдиної кодової бази. Це суттєво знижує витрати на розробку та підтримку.

Ключовими відмінностями Synergy Hub від існуючих рішень є:

- спеціалізація виключно на стартап-командах;
- автоматизований алгоритм оцінки сумісності;
- тривимірна модель матчингу (досвід, навички, бачення);
- інтеграція пошуку та комунікації в єдиному інтерфейсі;
- кросплатформність завдяки використанню Flutter.

1.5. Формування вимог до програмного продукту

На основі аналізу предметної області та обраної концепції було сформовано перелік функціональних та нефункціональних вимог до мобільного застосунку Synergy Hub.

Функціональні вимоги:

- реєстрація та автентифікація користувачів (email/password, Google OAuth);
- багатоетапний онбординг для збору інформації про спеціалізацію, досвід, навички та роль;
- перегляд профілів інших користувачів у форматі свайп-карток;
- фіксація рішення (лайк/дизлайк) та збереження історії взаємодій;
- автоматичне створення чату після взаємного матчу;
- обмін текстовими повідомленнями у реальному часі;
- push-сповіщення про нові матчі та повідомлення;
- редагування профілю користувача;

– фільтрація користувачів за спеціалізацією та досвідом.

Нефункціональні вимоги:

- кросплатформність (Android, iOS, Web);
- відповідність стандартам Material Design;
- час відгуку інтерфейсу не більше 2 секунд;
- масштабованість серверної частини;
- безпека зберігання та передачі даних;
- підтримка роботи в офлайн-режимі (кешування);
- автоматичний деплой через CI/CD.

Таблиця 1.2.

Функціональні вимоги до застосунку Synergy Hub

ID	Вимога	Пріоритет
FR-01	Реєстрація через email/password	Високий
FR-02	Автентифікація через Google OAuth	Високий
FR-03	Багатоетапний онбординг	Високий
FR-04	Свайп-перегляд профілів	Високий
FR-05	Фіксація лайку/дизлайку	Високий
FR-06	Автоматичне створення чату після матчу	Високий
FR-07	Обмін повідомленнями у реальному часі	Високий
FR-08	Push-сповіщення	Середній
FR-09	Редагування профілю	Середній
FR-10	Фільтрація за спеціалізацією	Середній

Додатково сформовано вимоги до інфраструктури та процесу розробки:

- використання системи контролю версій Git;
- автоматизація збірки та деплою через CI/CD;
- збереження секретних даних (API-ключі, токени) у захищеному середовищі;
- підтримка Firebase Emulator для локального тестування;
- покриття критичної бізнес-логіки юніт-тестами.

Сформовані вимоги є основою для подальшого проєктування архітектури та реалізації функціональних модулів застосунку, що описано у наступних розділах.

Висновки до розділу

У даному розділі було проведено аналіз предметної області формування стартап-команд та досліджено особливості підбору учасників для реалізації інноваційних проєктів. Встановлено, що успішність стартапу значною мірою залежить від правильного формування команди, рівня професійної сумісності учасників, їхніх навичок, досвіду та спільного бачення розвитку проєкту.

Було виконано аналіз існуючих програмних рішень для пошуку партнерів та командоутворення, зокрема сервісів Tinder, Bumble, LinkedIn, Meetup та інших платформ. Результати дослідження показали, що наявні рішення лише частково задовольняють потреби стартап-спільноти, оскільки не забезпечують комплексного підходу до формування команд, не містять ефективних механізмів оцінювання сумісності учасників та мають обмежені можливості для подальшої командної взаємодії.

На основі виявлених недоліків було обґрунтовано доцільність створення спеціалізованого мобільного застосунку Synergy Hub, орієнтованого на пошук потенційних учасників стартап-команд, оцінювання їхньої сумісності та підтримку комунікації між ними. Сформовано концепцію застосунку, визначено його основні функціональні можливості та переваги порівняно з існуючими аналогами.

Також було сформовано функціональні та нефункціональні вимоги до програмного продукту, які визначають основні характеристики майбутньої системи та слугують основою для її подальшого проєктування й реалізації. Отримані результати аналізу підтверджують актуальність розробки мобільного застосунку Synergy Hub та визначають напрямки подальшого проєктування архітектури, моделей даних і програмних компонентів системи.

РОЗДІЛ 2. ПРОЄКТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ SYNERGY HUB

2.1. Вибір мови програмування та технологій

Для реалізації мобільного застосунку Synergy Hub було обрано набір сучасних технологій та інструментів, що забезпечують ефективну розробку, тестування та розгортання продукту.

Flutter 3.9 та Dart — основний фреймворк та мова програмування для створення кросплатформного клієнтського застосунку. Flutter забезпечує компіляцію в нативний код для Android, iOS та Web з єдиної кодової бази.

TypeScript — мова програмування для реалізації серверної логіки Firebase Cloud Functions. TypeScript забезпечує строгу типізацію та покращує підтримуваність коду.

Firebase — платформа Backend-as-a-Service (BaaS) від Google, що включає:

- Firebase Auth — автентифікація користувачів;
- Firebase Firestore — хмарна NoSQL база даних;
- Firebase Cloud Functions — серверна логіка;
- Firebase Cloud Messaging — push-сповіщення;
- Firebase Storage — зберігання файлів;
- Firebase Hosting — хостинг веб-версії.

Інструменти кодогенерації. Для зменшення обсягу шаблонного коду (boilerplate) використовуються інструменти кодогенерації: Freezed для створення імутабельних data-класів з підтримкою `copyWith`, `equality` та `sealed class` `pattern matching`; `JsonSerializable` для автоматичної серіалізації/десеріалізації JSON; `go_router_builder` для типобезпечної генерації маршрутів; `flutter_gen` для генерації констант для ресурсів (зображення, шрифти).

Функціональне програмування. Бібліотека `dartx` надає функціональні типи `Either<Left, Right>` та `Option<T>`, що дозволяють явно моделювати

успішні та помилкові результати без використання exceptions. Це покращує надійність коду та робить обробку помилок більш передбачуваною.

Реактивне програмування. Бібліотека RxDart розширює стандартні Stream API Dart додатковими операторами (debounce, switchMap, combineLatest), що спрощує роботу з асинхронними потоками даних, особливо при реалізації пошуку та фільтрації.

Лінтинг та якість коду. Для забезпечення якості коду використовується файл analysis_options.yaml з правилами: avoid_print (забороняє debug-виведення), prefer_single_quotes (єдиний стиль лапок), require_trailing_commas (покращення git diff), prefer_relative_imports (чистіші імпорти). На сервері використовується ESLint з конфігурацією Google style guide.

Таблиця 2.1.

Основні залежності клієнтської частини

Бібліотека	Версія	Призначення
flutter_bloc	9.1.1	Керування станом (BLoC)
go_router	17.1.0	Типобезпечна маршрутизація
get_it	9.2.1	Впровадження залежностей
freezed	3.2.3	Генерація імутабельних моделей
firebase_core	4.1.1	Ініціалізація Firebase
cloud_firestore	6.1.1	База даних Firestore
firebase_auth	6.1.0	Автентифікація
firebase_messaging	16.0.2	Push-сповіщення
flutter_shaders	0.1.3	GLSL-шейдери
dartz	0.10.1	Функціональне програмування (Either)

Залежності серверної частини

Бібліотека	Версія	Призначення
firebase-admin	12.6.0	Серверний Firebase SDK
firebase-functions	6.0.1	Cloud Functions runtime
typescript	5.9.3	Мова програмування
jest	30.3.0	Фреймворк тестування
ts-jest	29.4.6	TypeScript для Jest

2.2. Загальна архітектура програмного забезпечення

Загальна архітектура програмного забезпечення мобільного застосунку Synergy Hub побудована з урахуванням принципів модульності, масштабованості та підтримованості. Під час проєктування системи було обрано підхід Clean Architecture, який передбачає чітке розмежування відповідальностей між окремими компонентами програми.

Основною метою використання такої архітектури є забезпечення незалежності бізнес-логіки від інтерфейсу користувача, зовнішніх сервісів та конкретних технологічних реалізацій. Це дозволяє спростити процес супроводу програмного продукту, його тестування та подальшого розвитку.

Система складається з двох основних компонентів:

- **клієнтська частина** — мобільний застосунок на Flutter (Dart), що реалізує інтерфейс користувача, бізнес-логіку та взаємодію з Firebase;
- **серверна частина** — набір Firebase Cloud Functions (TypeScript), що забезпечує матчинг, сповіщення та синхронізацію даних.

Клієнтська частина побудована за принципом модульної архітектури, де кожна функціональна область (автентифікація, онбординг, матчинг, чати, профіль, сповіщення) є ізольованим модулем з власною структурою Domain-Data-Presentation. Така модульність забезпечує можливість незалежної розробки та тестування окремих функціональних областей.

Серверна частина реалізована у вигляді Firebase Cloud Functions — безсерверних (serverless) функцій, які виконуються у хмарному середовищі Google Cloud. Використання безсерверної архітектури забезпечує автоматичне масштабування при зростанні навантаження, оплату лише за фактичне використання ресурсів та відсутність необхідності управління серверною інфраструктурою.

Взаємодія між клієнтською та серверною частинами здійснюється двома способами:

- **пряме звернення до Firestore** — для операцій читання/запису даних (профілі, чати, повідомлення). Firestore SDK забезпечує кешування, офлайн-підтримку та оновлення в реальному часі;
- **виклик Cloud Functions** — для складних операцій, що вимагають серверної валідації або атомарності (матчинг, розрахунок сумісності). Callable Functions забезпечують автоматичну передачу токена автентифікації.

Третім каналом комунікації є Firebase Cloud Messaging (FCM), який забезпечує push-сповіщення. Cloud Functions відправляють сповіщення через Firebase Admin SDK, а клієнтський застосунок отримує їх через FCM SDK.

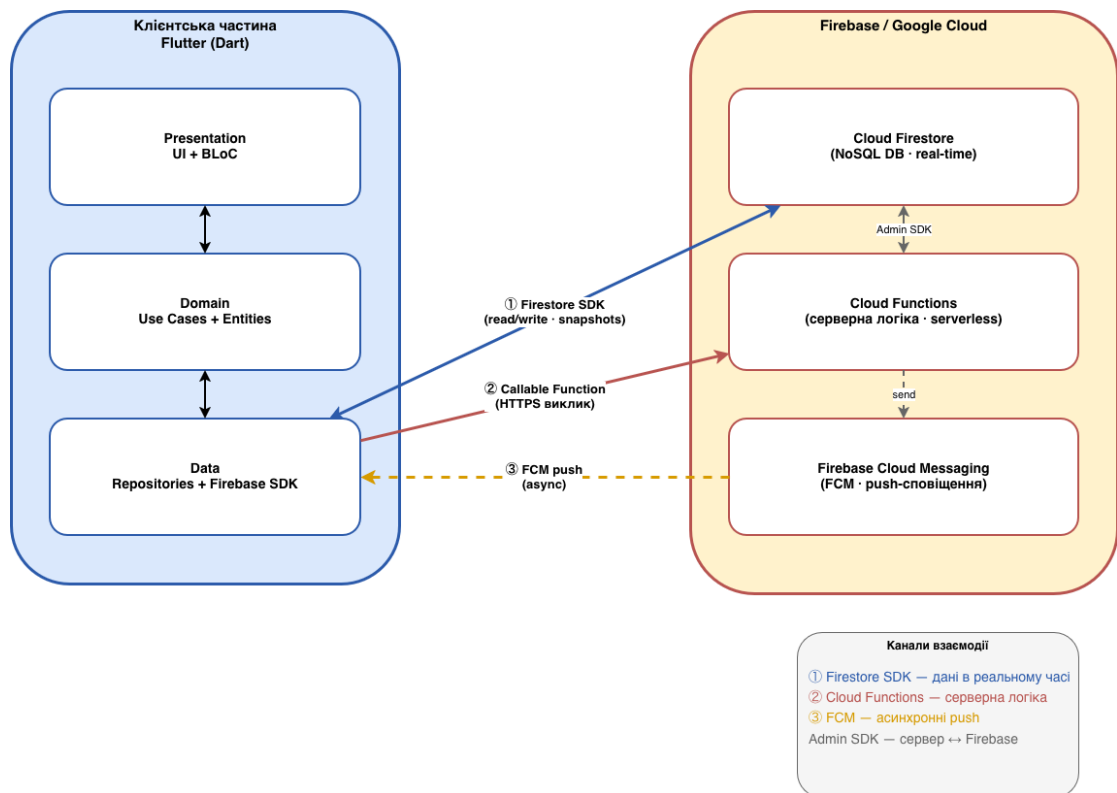


Рис. 2.1. Загальна архітектура системи Synergy Hub

2.3. Використання Clean Architecture (Domain, Data, Presentation)

Архітектура застосунку складається з трьох основних рівнів:

- Presentation Layer (рівень представлення);
- Domain Layer (рівень бізнес-логіки);
- Data Layer (рівень доступу до даних).

Кожен із зазначених рівнів виконує чітко визначені функції та взаємодіє з іншими рівнями через абстракції.

Рівень представлення (Presentation Layer) відповідає за взаємодію з користувачем та відображення інформації в інтерфейсі. До його основних функцій належать: формування графічного інтерфейсу за допомогою Flutter; обробка дій користувача (натискання, свайпи, введення даних); відображення станів завантаження, помилок та результатів операцій; взаємодія з бізнес-логікою через менеджер станів BLoC.

Рівень бізнес-логіки (Domain Layer) є центральною частиною архітектури та не залежить від зовнішніх бібліотек і платформ. До його складу

входять: бізнес-моделі (UserData, Chat, ChatMessage, Matches тощо); інтерфейси репозиторіїв; сценарії використання (Use Case); правила обробки даних.

Рівень доступу до даних (Data Layer) відповідає за отримання, збереження та обробку інформації з різних джерел. Основними компонентами даного рівня є: реалізації репозиторіїв; сервіси роботи з Firebase Firestore; Data Transfer Objects (DTO) для серіалізації/десеріалізації.

Взаємодія між рівнями відбувається відповідно до принципу інверсії залежностей: Presentation Layer звертається до Domain Layer через Use Case; Domain Layer працює з Data Layer через інтерфейси репозиторіїв; Data Layer реалізує доступ до конкретних джерел даних.

У процесі розробки застосовуються такі архітектурні патерни: Clean Architecture — для організації структури проєкту; Repository Pattern — для абстрагування доступу до даних; BLoC Pattern — для керування станами; Dependency Injection — для керування залежностями; Unit of Work — для транзакційних операцій на сервері.

Принцип інверсії залежностей (Dependency Inversion Principle) є фундаментальним для обраної архітектури. Він передбачає, що модулі верхнього рівня не повинні залежати від модулів нижнього рівня. Обидва рівні повинні залежати від абстракцій. У контексті Synergy Hub це означає, що BLoC-компоненти залежать від інтерфейсів репозиторіїв (IAuthRepository, IChatRepository), а не від конкретних реалізацій Firebase.

Така архітектура забезпечує можливість заміни джерела даних (наприклад, перехід з Firebase на власний бекенд) без модифікації бізнес-логіки та інтерфейсу. Кожен рівень може тестуватися ізольовано: Domain Layer — через юніт-тести з моками репозиторіїв, Presentation Layer — через bloc_test з моками Use Cases, Data Layer — через інтеграційні тести з Firebase Emulator.

Організація коду за фічерним принципом (feature-based structure) забезпечує модульність проєкту. Кожна функціональна область (auth, matches,

chat, user, onboard_survey, notifications) містить власні domain, data та presentation рівні. Це дозволяє розробникам працювати над окремими фічами незалежно та мінімізує конфлікти при паралельній розробці.

Файлова структура фічерного модуля має такий вигляд:

```
features/
  matches/
    data/
      models/          // DTO для серіалізації
      repositories/    // Firebase реалізації
    domain/
      models/          // Бізнес-моделі (Freezed)
      repositories/    // Інтерфейси
      usecases/        // Сценарії використання
    presentation/
      bloc/            // BLoC/Cubit компоненти
      pages/           // Екрани та віджети
    di/
      matches_injection.dart // DI налаштування
```

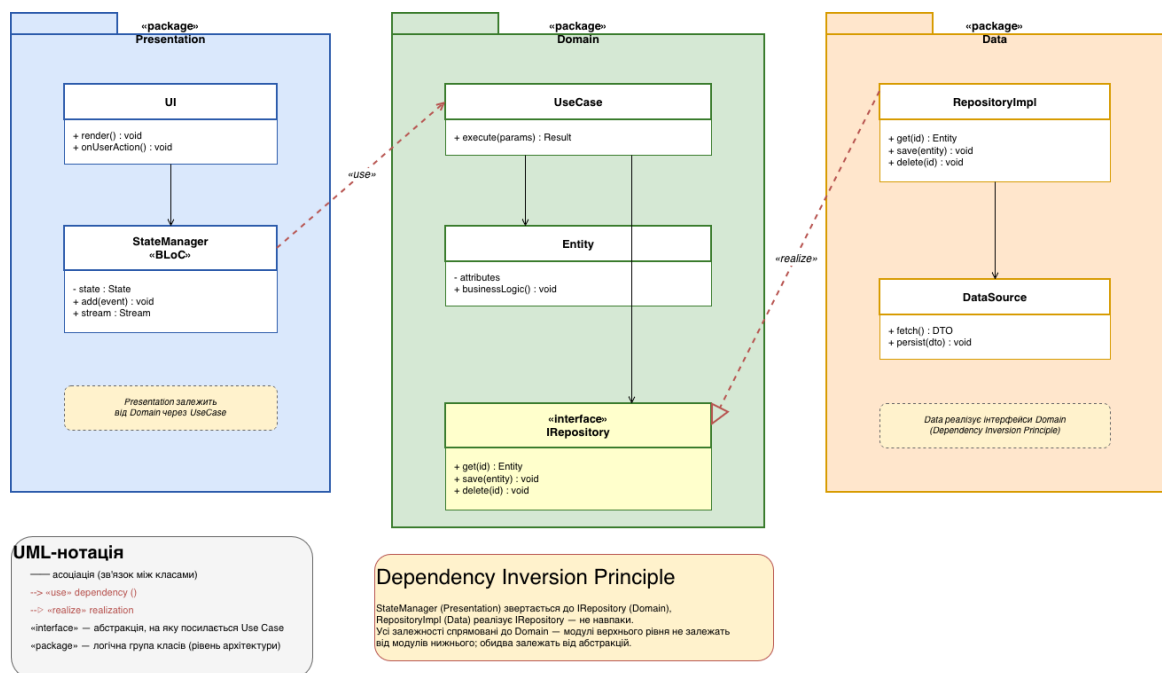


Рис. 2.2. Діаграма залежностей між рівнями Clean Architecture

2.4. Проєктування бази даних на основі Firebase Firestore

Для зберігання та обробки даних у мобільному застосунку Synergy Hub використовується хмарна NoSQL-база даних Firebase Firestore. Дане рішення

забезпечує високу масштабованість, синхронізацію в реальному часі та інтеграцію з мобільними платформами.

Firestore використовує ієрархічну модель зберігання даних, що базується на колекціях, документах та вкладених підколекціях.

На кореневому рівні бази даних розміщено такі основні колекції:

- **users** — зберігання інформації про користувачів (uid, firstName, lastName, age, avatarUrl, aboutMe, profiler);
- **chats** — зберігання чатів з підколекцією messages для повідомлень;
- **matches** — зберігання інформації про матчі між користувачами.

Документ користувача містить вкладену модель Profiler, яка описує професійну спеціалізацію:

- рівень досвіду (Junior, Middle, Senior, Lead/Expert);
- роль у проєкті;
- основна спеціалізація (Frontend, Backend, Mobile, DevOps тощо);
- перелік навичок (Skills);
- напрямки зацікавленості (searchingSpecializations).

У межах кожного документа користувача створено набір вкладених підколекцій:

- **onboardStatus** — зберігає стан проходження онбордингу;
- **notificationConfig** — зберігає FCM-токени та налаштування сповіщень;
- **notifications** — зберігає історію сповіщень користувача;
- **chats** — посилання на чати, в яких бере участь користувач.

Документ матчу ідентифікується за ключем, що формується сортованою конкатенацією UID обох користувачів (наприклад, uid1_uid2). Це гарантує унікальність запису незалежно від напрямку ініціації матчу.

Детальна структура бази даних наведена у Додатку Б.

Вибір Firebase Firestore як основної бази даних обумовлений кількома факторами. По-перше, Firestore забезпечує реальночасову синхронізацію даних через Snapshot Listeners, що є критичним для системи чатів. По-друге,

Firestore має вбудований офлайн-режим з автоматичною синхронізацією при відновленні з'єднання. По-третє, Firestore масштабується автоматично і не вимагає ручного налаштування серверної інфраструктури.

Для забезпечення консистентності даних при операціях матчингу використовуються Firestore Transactions. Транзакція гарантує, що всі операції (читання стану матчу, створення чату, оновлення статусу) виконуються атомарно. Якщо будь-яка операція в транзакції завершується помилкою, всі зміни відкочуються.

Індексація даних у Firestore налаштована для оптимізації основних запитів: пошук чатів за учасником (array-contains на полі participants), пошук матчів за статусом та учасником, пошук чатів за ключовими словами (searchKeywords). Складені індекси створюються автоматично при першому запиті через Firebase Console.

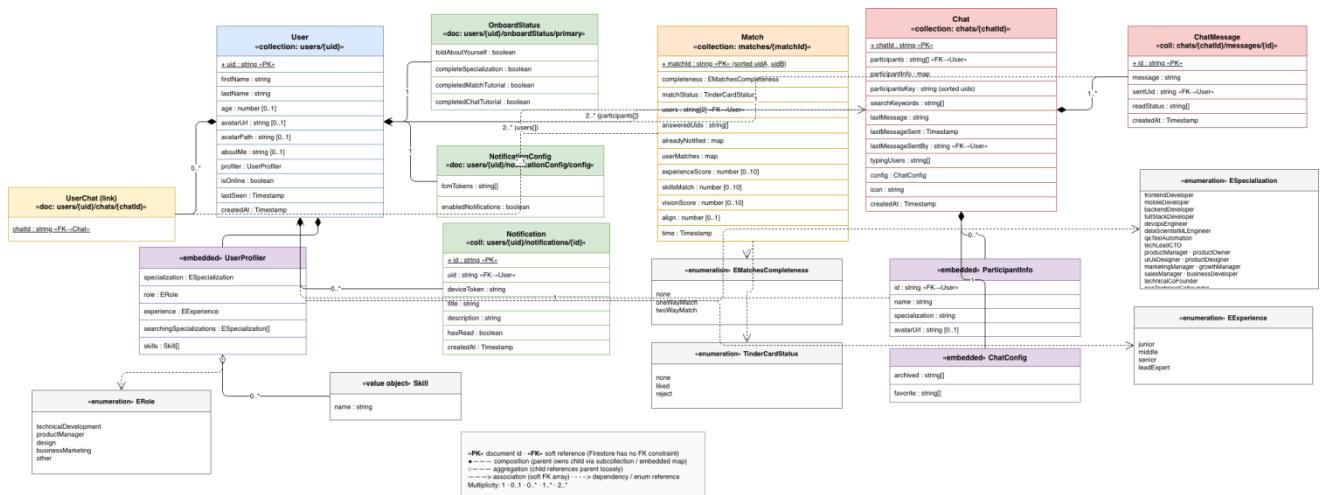


Рис. 2.3. Датологічна модель БД (NoSQL Firebase)

2.5. Моделювання взаємодії користувачів у системі

Взаємодія користувачів у системі Synergy Hub побудована на принципі двостороннього підтвердження (mutual matching). Процес взаємодії включає кілька послідовних етапів:

Етап 1: Перегляд профілю. Користувач переглядає картку іншого учасника на Discover-екрані, де відображається ім'я, спеціалізація, рівень досвіду, навички та опис.

Етап 2: Прийняття рішення. Користувач виконує свайп вправо (лайк) або вліво (дизлайк). Рішення фіксується через виклик Cloud Function matchUser.

Етап 3: Обробка матчу. Cloud Function перевіряє наявність зустрічної реакції. Якщо обидва користувачі виразили зацікавленість, відбувається двосторонній матч (twoWayMatch).

Етап 4: Створення чату. Після двостороннього матчу автоматично створюється чат між учасниками. Обидва користувачі отримують push-сповіщення.

Етап 5: Комунікація. Користувачі обмінюються повідомленнями у реальному часі через Firebase Firestore. Кожне нове повідомлення генерує push-сповіщення для отримувача.

При двосторонньому матчі розраховуються три показники сумісності:

- **Experience Score** — оцінка співвідношення рівнів досвіду (формула: $\max(10 - \text{diff} \times 3, 2)$);
- **Skills Score** — оцінка перетину технічних навичок ($\text{intersection} / \max(\text{skills_a}, \text{skills_b}) \times 10$);
- **Vision Score** — оцінка співпадіння бачення на основі текстового аналізу полів aboutMe.

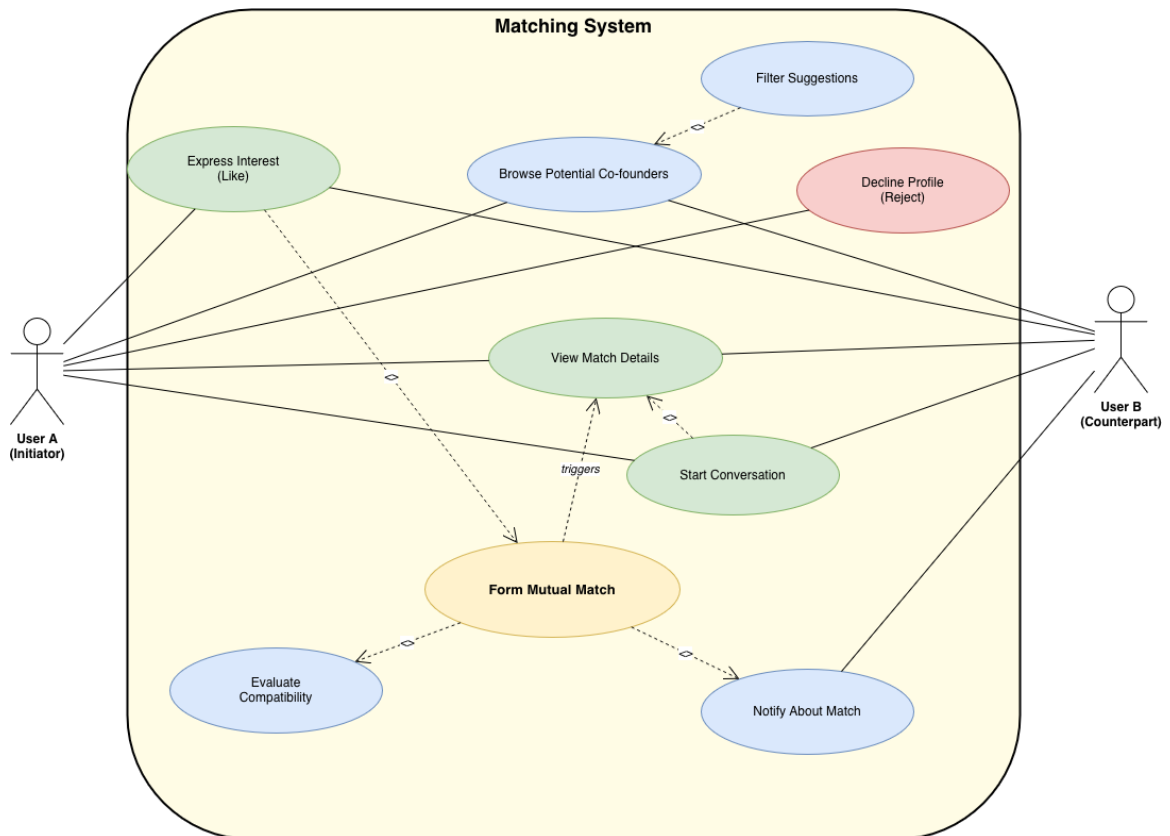


Рис. 2.4. Діаграма взаємодії користувачів у системі матчингу

Стани матчу визначаються перелічувальним типом `CompletenessStatus`:

- **none** — один з учасників виконав дизлайк, матч не відбувся;
- **oneWayMatch** — лише один учасник виразив зацікавленість (лайк), очікується рішення другого;
- **twoWayMatch** — обидва учасники виразили зацікавленість, створюється чат.

Важливим архітектурним рішенням є генерація унікального ідентифікатора матчу через сортовану конкатенацію UID обох користувачів. Наприклад, для користувачів з UID “abc123” та “xyz789” ідентифікатор матчу буде “abc123_xyz789” (незалежно від того, хто першим виконав свайп). Це забезпечує відсутність дублікатів і дозволяє обом учасникам знайти запис матчу за єдиним ключем.

Після створення чату система автоматично генерує пошукові ключові слова (`searchKeywords`) на основі імен учасників. Алгоритм генерації створює посимвольні префікси для підтримки автодоповнення при пошуку чатів.

Наприклад, для імені “Oleksandr” будуть створені ключові слова: “o”, “ol”, “ole”, “olek” і так далі.

Система сповіщень інтегрована у процес матчингу на двох рівнях. При односторонньому лайку відправляється сповіщення з текстом “A potential co-founder liked your profile”. При двосторонньому матчі обидва учасники отримують сповіщення “It’s a Match!” з даними для навігації до екрану матчу.

2.6. Проєктування інтерфейсу користувача

Інтерфейс користувача Synergy Hub спроектовано з урахуванням принципів Material Design та сучасних підходів до мобільного UX-дизайну. Навігація побудована на основі бібліотеки GoRouter з типобезпечною маршрутизацією.

Структура навігації включає:

- **екран авторизації** — головний екран з вибором методу входу (email/password або Google);
- **онбординг** — багатоетапна форма збору інформації про користувача;
- **Discovery** — основний екран з картками для свайпів;
- **Matches** — список матчів та активних чатів;
- **Chat** — екран обміну повідомленнями;
- **Profile** — перегляд та редагування профілю;
- **Notifications** — список сповіщень.

Маршрутизація реалізована за допомогою вкладених ShellRoute та StatefulShellRoute, що забезпечує збереження стану між вкладками та плавні анімації переходів (FadeTransition).

Ієрархія маршрутів:

```
Root Shell (RootPageRoute)
  Authentication Shell
    / - AuthorizationPageRoute
    /sign-up - RegisterPageRoute
    /sign-in - LoginPageRoute
  Onboarding Shell
    /onboard - OnboardSurveyPageRoute
    /onboard-about - OnboardAboutYouPageRoute
  Home Stateful Shell
```

```
/discovery - DiscoveryPageRoute  
/matches - MatchesPageRoute  
/chat - ChatPageRoute (?chatId)  
/profile - ProfilePageRoute  
/notification - NotificationPageRoute
```

StatefulShellRoute забезпечує збереження стану вкладок при навігації між ними. Це означає, що при переході з Discovery на Matches і назад позиція скролу та завантажені дані зберігаються.

Для забезпечення адаптивності інтерфейсу використовується ResponsiveFramework з такими breakpoints: MOBILE (0-450px), TABLET (451-900px), DESKTOP (901-1920px), 4K (1921px+). Це дозволяє коректно відображати інтерфейс як на мобільних пристроях, так і у веб-версії.

Кольорова палітра базується на відтінках Coral/Peach (#F27361, #FA9E73, #FFBF8C), що відображає фірмовий стиль Synergy Hub та використовується у GLSL-шейдерах для анімованого фону.

2.7. Розробка онбордингу та збору інформації про користувача

Онбординг у Synergy Hub є ключовим елементом, що забезпечує збір необхідної інформації для роботи алгоритму матчингу. Він складається з двох основних етапів:

Етап 1 — About You. Користувач заповнює основну персональну інформацію: ім'я, прізвище, вік, фото профілю та короткий опис (aboutMe). Ці дані використовуються для відображення на картці профілю та для розрахунку Vision Score.

Етап 2 — Professional Survey. Багатоетапне опитування, що включає чотири стадії:

- **Specialization Stage** — вибір основної спеціалізації (Frontend Developer, Backend Developer, Mobile Developer, DevOps Engineer, Data Scientist, UI/UX Designer, Product Manager тощо);
- **Experience Stage** — визначення рівня досвіду (Junior, Middle, Senior, Lead/Expert);

- **Role Stage** — вибір ролі у стартап-проекті (Technical Development, Product Manager, Design, Business/Marketing);
- **Goal Stage** — визначення пошукових преференцій (які спеціалізації шукає користувач для своєї команди).

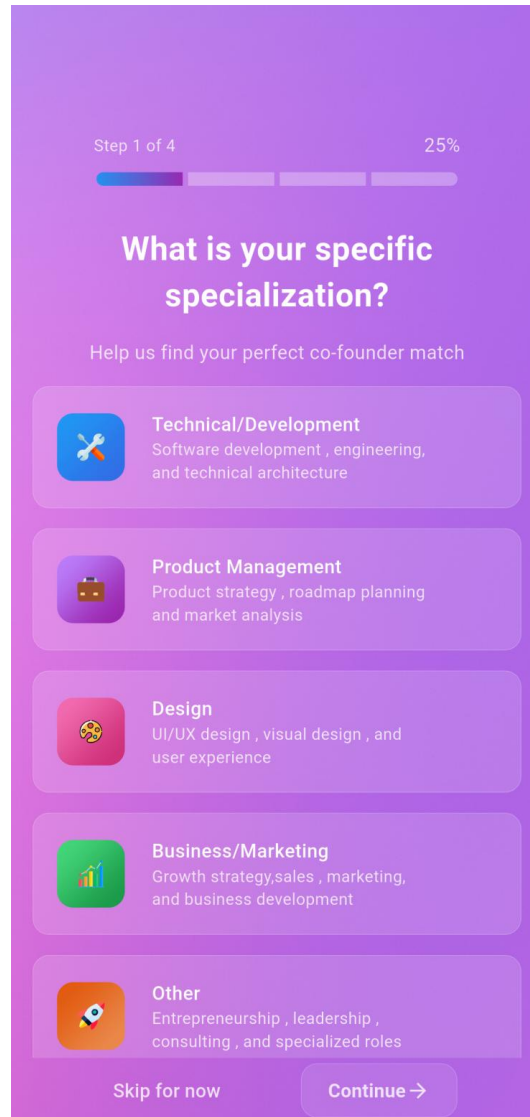


Рис. 2.5. Онбординг: Вибір спеціалізації

Стан онбордингу зберігається у підколекції `onboardStatus` документа користувача у Firestore. Це дозволяє відновлювати процес онбордингу після виходу з додатку.

Онбординг включає проміжні мотиваційні екрани (overlays): `PerfectChoiceOverlay` — після вибору спеціалізації з підтримуючим повідомленням; `ActiveCommunityOverlay` — інформація про активну

спільноту; ProcessingOverlay — анімація обробки даних перед переходом до основного інтерфейсу.

Прогрес онбордингу відображається через StepProgressIndicator — візуальний індикатор поточного етапу. Це допомагає користувачу зрозуміти, скільки етапів залишилося, і зменшує ймовірність передчасного виходу.

Дані, зібрані під час онбордингу, використовуються у трьох контекстах: для відображення на карточці профілю (ім'я, фото, спеціалізація); для алгоритму матчингу (досвід, навички, aboutMe); для фільтрації кандидатів (searchingSpecializations).

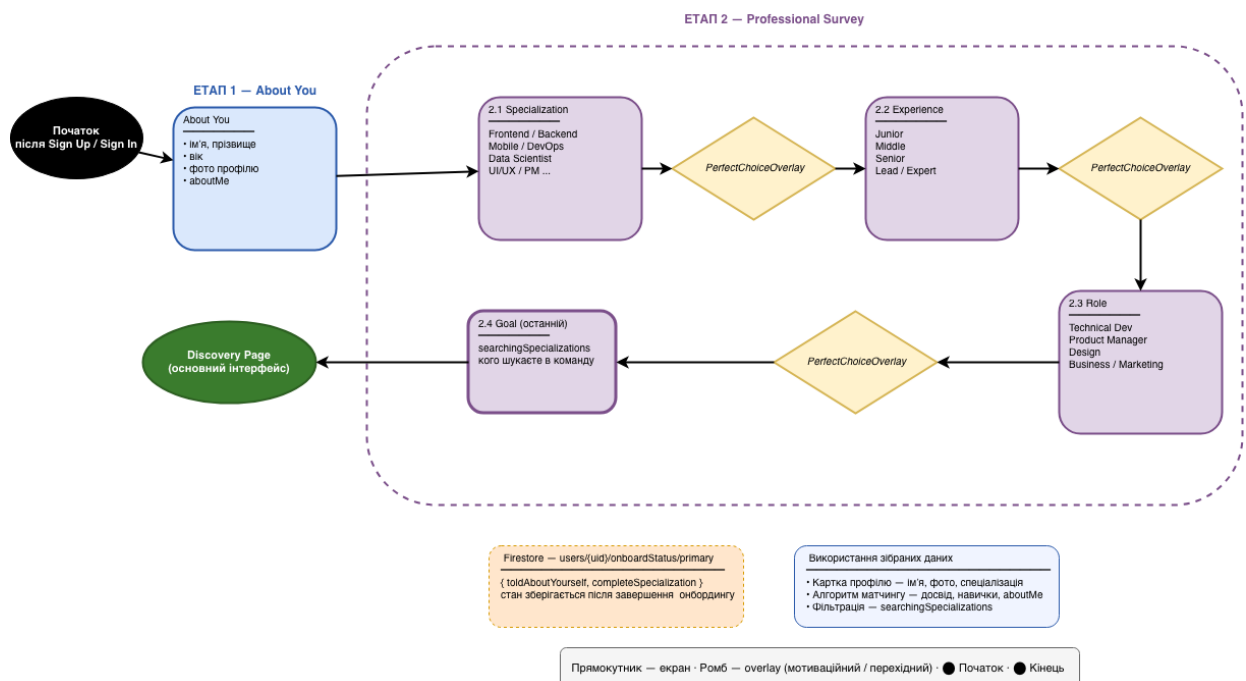


Рис. 2.6. Етапи онбордингу Synergy Hub

2.8. Проєктування механізму пошуку та свайп-системи

Свайп-система є центральним елементом взаємодії користувача з додатком. Механізм пошуку побудовано на принципі послідовного перегляду карток профілів з можливістю прийняття рішення за допомогою жестів.

Проєктні рішення:

- **стек карток** — одночасно відображається поточна картка та попередня для створення ефекту глибини;

- **жестова взаємодія** — обробка drag-жестів з розрахунком зміщення та кута повороту картки;
- **порогове значення** — для фіксації рішення зміщення має перевищити 75 пікселів по горизонталі;
- **візуальний відгук** — індикація лайку або дизлайку на карточці під час свайпу;
- **фільтрація** — виключення вже переглянутих профілів з подальших показів.

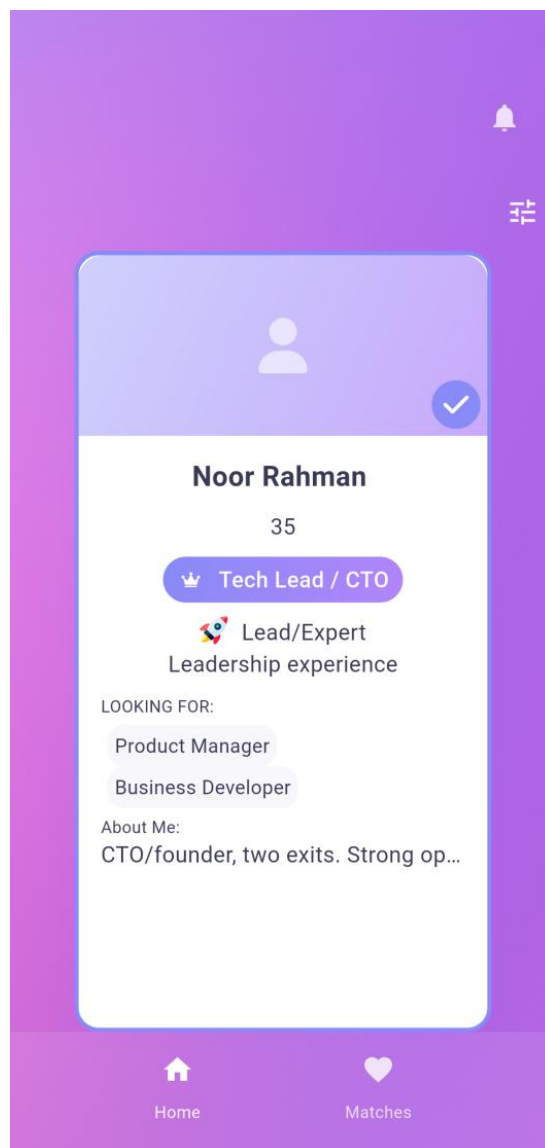


Рис. 2.7. Discover-екран зі свайп-картками

Керування станом свайп-картки реалізовано через TinderCardCubit, який обробляє події onPanStart, onPanUpdate та onPanEnd для відстеження позиції та визначення статусу (liked, reject, none).

Анімація картки включає два компоненти: горизонтальне зміщення (translate) та обертання (rotate). Кут обертання розраховується пропорційно горизонтальному зміщенню з максимумом 45 градусів. При відпусканні картки, якщо зміщення не досягло порогового значення, картка плавно повертається на початкову позицію з використанням пружинної анімації.

Для оптимізації завантаження кандидатів використовується пагінація. PeopleDataBloc завантажує кандидатів пакетами та запитує наступний пакет, коли поточний наближається до кінця. GetExcludedMatchesUseCase забезпечує виключення вже переглянутих профілів з наступних запитів.

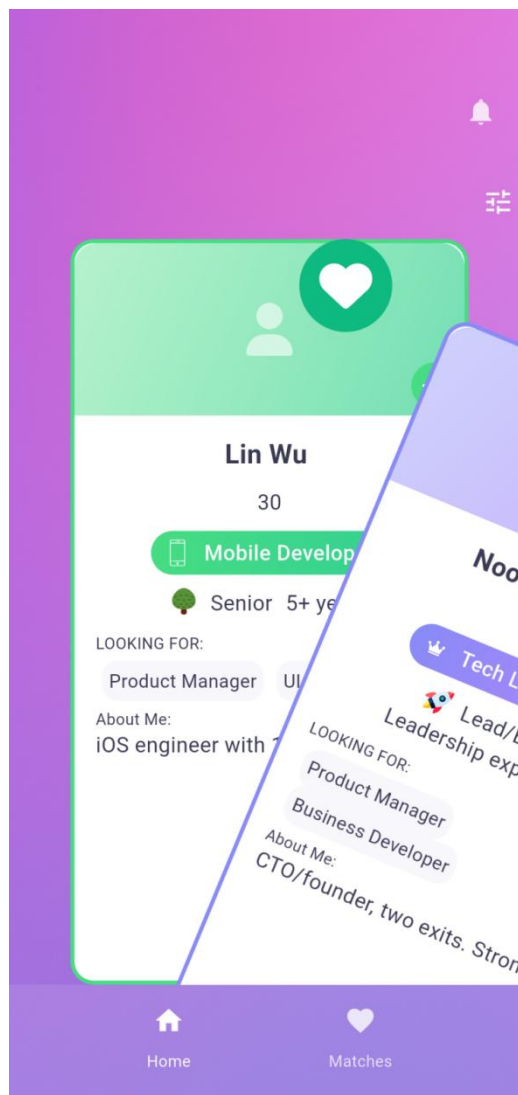


Рис. 2.8. Макет свайп-екрану з індикацією рішення

2.9. Проектування системи чатів та повідомлень

Система чатів спроектована для забезпечення обміну повідомленнями між користувачами, що здійснили взаємний матч. Архітектура чатів базується на реальночасовій синхронізації через Firebase Firestore.

Модель чату включає такі поля:

- chatId — унікальний ідентифікатор чату;
- participants — масив UID учасників;
- participantInfo — мапа інформації про учасників (ім'я, аватар, спеціалізація);
- lastMessage, lastMessageSent — інформація про останнє повідомлення;
- typingUsers — масив UID користувачів, які зараз набирають повідомлення;
- searchKeywords — ключові слова для пошуку чату;
- config — налаштування (архівовані, улюблені чати).

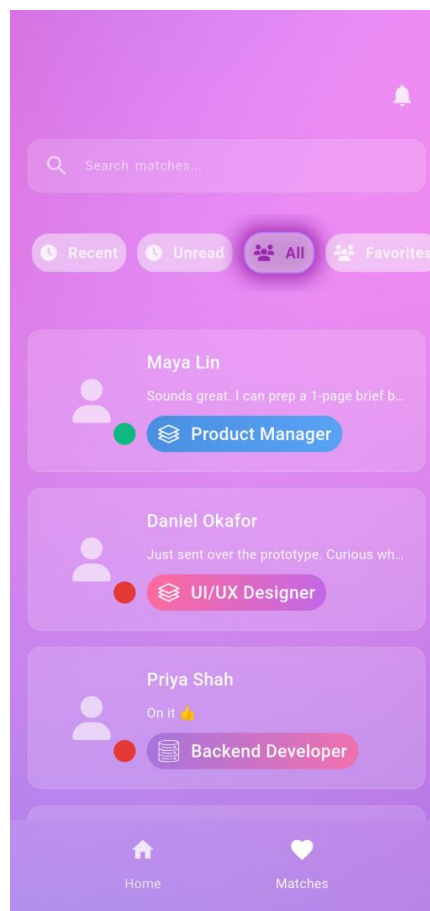


Рис. 2.9. Список чатів та матчів

Повідомлення зберігаються у вкладеній підколекції messages кожного чату. Підписка на оновлення реалізується через Firestore Snapshot Listeners, що забезпечує миттєву доставку нових повідомлень без необхідності політгу.

Синхронізація профілю у чатах забезпечується Cloud Function onUserUpdate, яка автоматично оновлює participantInfo у всіх чатах користувача при зміні його імені, аватару чи спеціалізації.

Архітектурно система чатів розроблена з урахуванням можливості майбутнього розширення. Поточна реалізація підтримує лише текстові повідомлення, проте структура моделі ChatMessage дозволяє додати підтримку зображень, файлів та голосових повідомлень без зміни архітектури.

Для забезпечення консистентності даних при одночасному оновленні чату декількома учасниками використовуються атомарні операції Firestore: arrayUnion та arrayRemove для масивів (typingUsers), FieldValue.serverTimestamp для серверного часу (lastMessageSent).

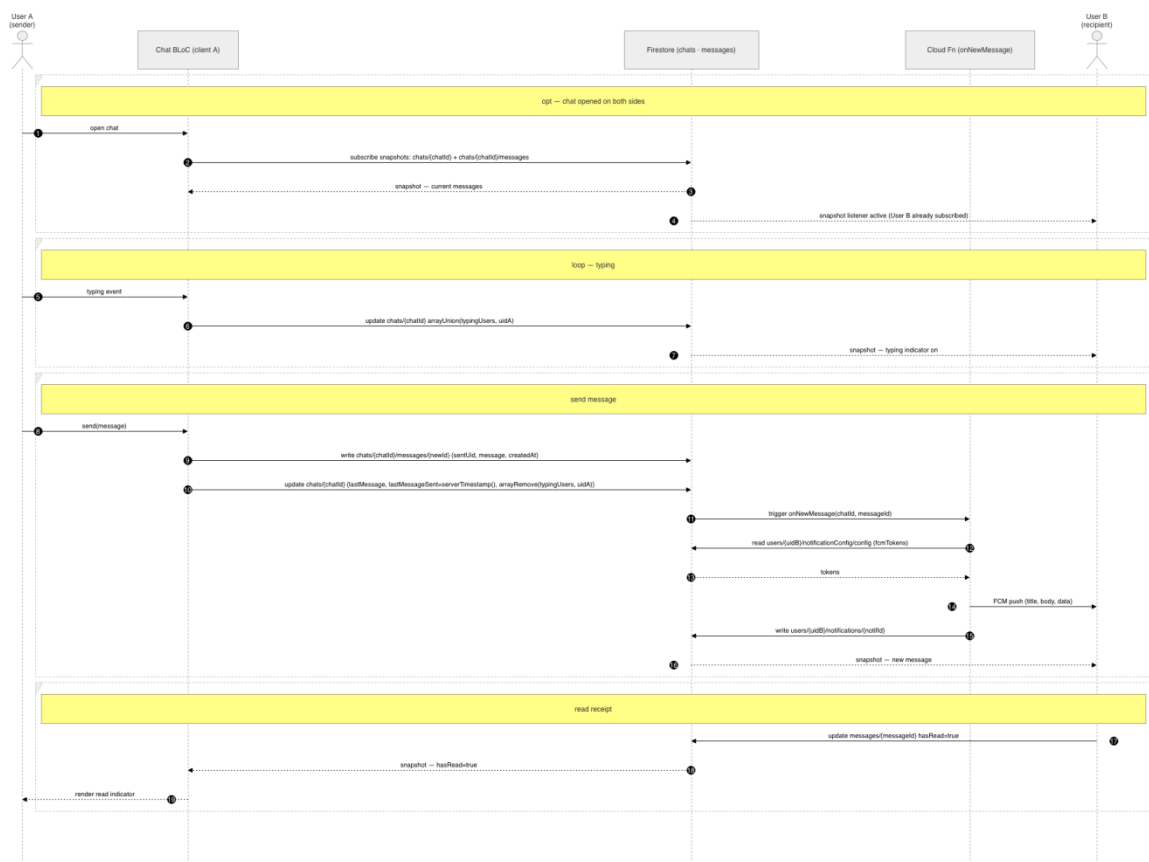


Рис. 2.10. Діаграма послідовності обміну повідомленнями

Висновки до розділу

У даному розділі було здійснено проектування мобільного застосунку Synergy Hub, спрямованого на ефективне формування стартап-команд та забезпечення взаємодії між користувачами. На основі аналізу було обґрунтовано вибір сучасного технологічного стеку, зокрема Flutter та Dart для клієнтської частини, Firebase як хмарної платформи Backend-as-a-Service, а також TypeScript для реалізації серверної логіки у Cloud Functions.

Було визначено основні бібліотеки та інструменти, які забезпечують стабільність, масштабованість та підтримуваність системи, включаючи засоби кодогенерації, функціонального та реактивного програмування, а також механізми контролю якості коду. Це дозволило зменшити обсяг шаблонного коду та підвищити надійність програмного продукту.

Розроблено загальну архітектуру системи, яка базується на підході Clean Architecture та принципах модульності. Система поділена на клієнтську та серверну частини, між якими організовано взаємодію через Firestore, Cloud Functions та Firebase Cloud Messaging. Такий підхід забезпечує масштабованість, гнучкість та легкість подальшого розвитку системи.

Детально описано реалізацію багаторівневої архітектури (Presentation, Domain, Data), яка дозволяє чітко розмежувати відповідальності компонентів та забезпечує незалежність бізнес-логіки від зовнішніх технологічних залежностей. Використання патернів Repository, BLoC та Dependency Injection підвищує тестованість і підтримуваність системи.

Було спроектовано структуру бази даних на основі Firebase Firestore, визначено основні колекції, підколекції та принципи зберігання даних. Особливу увагу приділено механізмам забезпечення цілісності даних, використанню транзакцій, індексації та real-time синхронізації.

Описано модель взаємодії користувачів у системі, яка базується на принципі взаємного матчингу, а також алгоритми розрахунку сумісності між користувачами за трьома ключовими параметрами: досвід, навички та

бачення. Це забезпечує більш якісний підбір потенційних учасників стартап-команд.

Окремо було розглянуто проєктування ключових функціональних модулів застосунку, зокрема системи онбордингу, свайп-механізму, пошуку, чатів та повідомлень. Кожен із цих компонентів інтегрований у загальну архітектуру системи та забезпечує зручну та інтуїтивну взаємодію користувача із застосунком.

Отримані результати підтверджують правильність вибраного архітектурного підходу та технологічних рішень, а також створюють основу для подальшої реалізації мобільного застосунку Synergy Hub, включаючи розробку бізнес-логіки, інтерфейсу користувача та серверної взаємодії.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ МОБІЛЬНОГО ДОДАТКА

3.1. Архітектура реалізації клієнтської частини на Flutter та Dart

Клієнтська частина застосунку організована за фічерним принципом, де кожна функціональна область ізольована у власному модулі з дотриманням Clean Architecture.

Основні фічерні модулі:

- **auth** — реєстрація та авторизація;
- **onboard_survey** — онбординг та збір інформації;
- **matches** — свайп-система та матчинг;
- **chat** — обмін повідомленнями;
- **user** — керування профілем;
- **notifications** — сповіщення;
- **home** — головна навігація та вкладки.

Кожен модуль містить три рівні:

- domain/ — моделі, інтерфейси репозиторіїв, Use Cases;
- data/ — DTO, реалізації репозиторіїв;
- presentation/ — BLoC/Cubit, сторінки, віджети.

Для генерації коду використовуються: Freezed — для створення імутабельних data-класів з підтримкою copyWith та pattern matching; JsonSerializable — для автоматичної серіалізації/десеріалізації JSON; go_router_builder — для типобезпечної генерації маршрутів.

Для керування станом застосовується патерн BLoC з використанням бібліотеки flutter_bloc. BLoC отримує події (Events), обробляє їх та емітить нові стани (States). Для простих випадків використовується Cubit — спрощена версія BLoC без подій.

Приклад структури BLoC-подій для модуля автентифікації:

```
@freezed
sealed class AuthEvent {
  const factory AuthEvent.init() = _Init;
  const factory AuthEvent.listen() = _Listen;
  const factory AuthEvent.signInViaGoogle() = _SignInViaGoogle;
  const factory AuthEvent.signOut() = _SignOut;
}
```

BLoC-компоненти підключаються до дерева віджетів через MultiBlocProvider у кореновому віджеті додатку. Це забезпечує доступність станів автентифікації, користувача та навігації на всіх рівнях ієрархії:

```
MultiBlocProvider(  
  providers: [  
    BlocProvider<UserBloc>.value(  
      value: userBloc,  
    ),  
    BlocProvider<AuthBloc>.value(  
      value: authBloc,  
    ),  
  ],  
  child: MaterialApp.router(...)  
)
```

Для спостереження за станом BLoC-компонентів під час розробки реалізовано AppObserver, який логує події створення, зміни стану та виникнення помилок. Це значно спрощує процес налагодження та дозволяє відслідковувати послідовність подій.

Моделі даних генеруються через Freezed, що забезпечує: імутабельність (всі поля final); автоматичну реалізацію equality (== та hashCode); метод copyWith для створення модифікованих копій; підтримку JSON серіалізації через @JsonSerializable; sealed classes для вичерпного pattern matching.

Приклад моделі UserData:

```
@freezed  
sealed class UserData {  
  const factory UserData({  
    required String uid,  
    String? avatarUrl,  
    String? firstName,  
    String? lastName,  
    int? age,  
    String? aboutMe,  
    required UserProfiler profiler,  
    @TimestampConverter()  
    required DateTime createdAt,  
  }) = _UserData;  
}
```

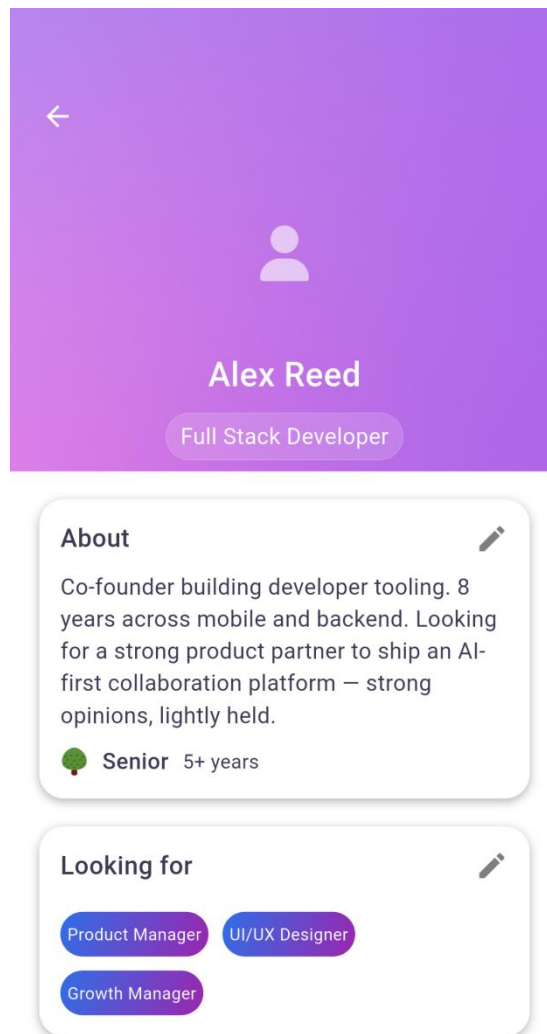


Рис. 3.1. Профіль користувача

3.2. Використання патерну Repository, Unit of Work та Dependency Injection

Repository Pattern. Патерн Repository забезпечує абстрагування доступу до даних. У Domain Layer визначені інтерфейси (IAuthRepository, IChatRepository, IMatchesRepository, IUserRepository), а в Data Layer — їх реалізації з використанням Firebase.

Unit of Work. На сервері (TypeScript) реалізовано патерн Unit of Work через клас FirebaseUnitOfWork, який обертає всі операції в єдину Firestore-транзакцію. Це гарантує атомарність операцій матчингу:

```
async run<T>(fn: (ctx: ITransactionContext) => Promise<T>) {  
  return firestore().runTransaction(async (tx) => {  
    const ctx: ITransactionContext = {
```

```

        matches: this.factories.matchesRepo(tx),
        notifications: this.factories.notificationsRepo(tx),
        chats: this.factories.chatsRepo(tx),
        userData: this.factories.userDataRepo(tx),
    };
    return fn(ctx);
  });
}

```

Dependency Injection. На клієнті використовується GetIt (Service Locator) для керування залежностями. Ініціалізація виконується при запуску додатку:

```

final getIt = GetIt.instance;

Future<void> initDependencies() async {
  getIt.registerLazySingleton(
    () => FirebaseFirestoreService(),
  );
  initAuthFeature(getIt);
  initMatchingFeature(getIt);
  initChatFeature(getIt);
}

```

На сервері DI реалізовано через клас Container, який створює та зв'язує всі залежності: репозиторії, сервіси, Use Cases та Unit of Work.

Ключовою особливістю серверного DI є фабричний метод для створення транзакційних репозиторіїв. Кожен репозиторій приймає опціональний параметр Transaction, що дозволяє використовувати один і той самий репозиторій як у транзакційному, так і в нетранзакційному контексті:

```

chatRepo = (tx?: Transaction) =>
  new ChatFirebaseRepository(
    firestore, participantIdentifier,
    chatKeyWordGenerator, tx
  );

```

Це забезпечує гнучкість використання: для операцій матчингу репозиторії створюються з транзакцією (для атомарності), а для операцій синхронізації профілю — без транзакції (для кращої продуктивності).

На клієнті кожен фічерний модуль має власну функцію ініціалізації DI (наприклад, initMatchingFeature, initChatFeature), яка реєструє всі залежності модуля в контейнері GetIt. Це забезпечує чітке розмежування відповідальності та полегшує рефакторинг.

3.3. Реалізація анімованого фону з використанням GLSL-шейдерів

Для створення візуально привабливого анімованого фону в застосунку використовуються GLSL-шейдери (OpenGL Shading Language), інтегровані через бібліотеку `flutter_shaders`. Шейдери виконуються безпосередньо на GPU, що забезпечує плавну анімацію без навантаження на основний потік.

Основний шейдер `animated_gradient.frag` реалізує складний анімований градієнт з такими ефектами:

- множинні хвильові рухи на різних швидкостях для створення живого ефекту;
- палітра кольорів Coral/Salmon/Peach для фірмового стилю;
- пульсуючий ефект від центру з використанням функції `length()`;
- процедурний шум для текстурності;
- динамічна зміна яскравості.

Фрагмент основної функції шейдера:

```
vec3 getGradientColor(vec2 uv, float time) {  
    float wave1 = sin(uv.x * 3.0 + time * 0.8) * 0.5 + 0.5;  
    float wave2 = cos(uv.y * 3.5 + time * 0.6) * 0.5 + 0.5;  
    vec3 color1 = vec3(0.95, 0.45, 0.38); // Coral  
    vec3 color2 = vec3(0.98, 0.62, 0.45); // Orange  
    vec3 mixColor = mix(color1, color2, wave1);  
    return mixColor;  
}
```

Шейдер приймає два `uniform`-параметри: `uSize` (розмір екрану) та `uTime` (поточний час для анімації). Відображення шейдера здійснюється через `CustomPaint` з використанням `ShaderPainter`, який передає параметри у фрагментний шейдер кожного кадру.

Окрім `animated_gradient.frag`, у проєкті використовуються додаткові шейдери:

- **`check_pulse.frag`** — анімація пульсуючої позначки, що використовується при завершенні онбордингу для візуального зворотного зв'язку;
- **`gradient_smooth.frag`** — плавний градієнтний ефект для фону окремих екранів;

- **ve.frag** — розширений градієнтний ефект з додатковим змішуванням кольорів.

Всі шейдери декларуються у `pubspec.yaml` в секції `flutter.shaders` та компілюються разом з додатком. Flutter забезпечує кросплатформну підтримку шейдерів: на мобільних платформах використовується Impeller (або Skia), на веб-платформі — WebGL.

Інтеграція шейдера з Flutter-віджетом реалізована через ланцюжок: завантаження шейдера через `FragmentProgram.fromAsset()` → створення `FragmentShader` → передача `uniform`-параметрів → відображення через `CustomPaint` з `ShaderPainter`. Анімація забезпечується через `AnimationController`, який оновлює параметр `uTime` кожного кадру.

Використання GPU-шейдерів замість стандартних Flutter-анімацій забезпечує значно вищу продуктивність, оскільки обчислення виконуються паралельно на графічному процесорі, не навантажуючи основний UI-потік.

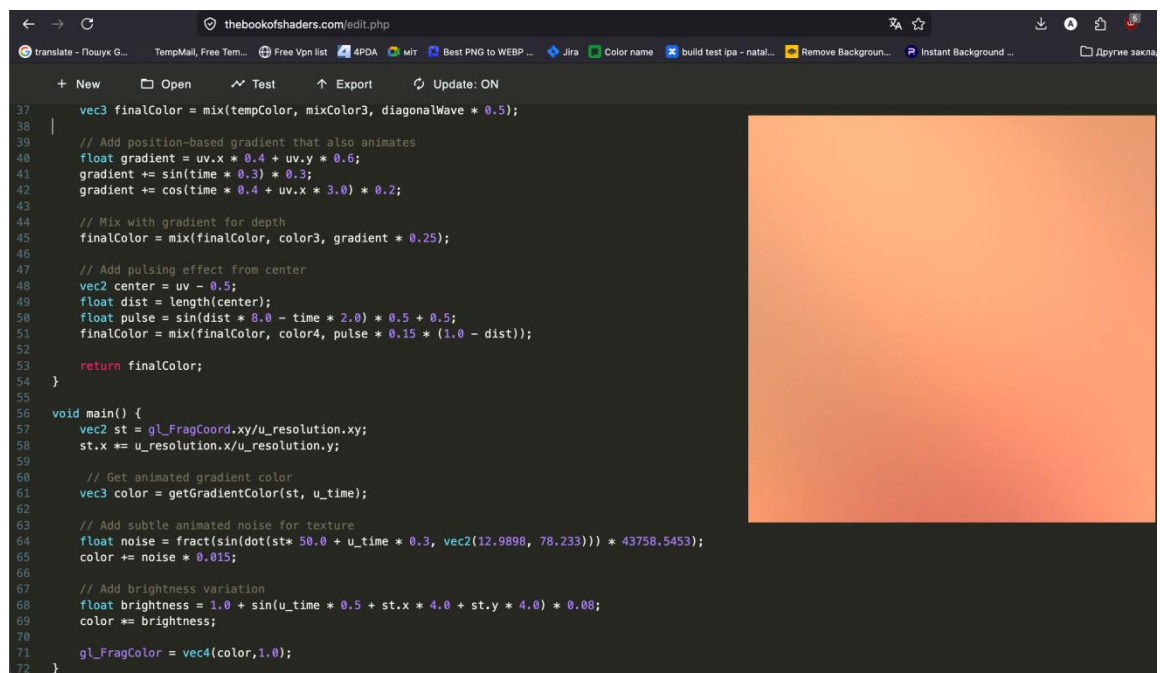


Рис. 3.2. GLSL Анімований фон сторінки з результатами матчингу

3.4. Реалізація системи автентифікації користувачів

Система автентифікації Synergy Hub підтримує два методи входу: електронна пошта з паролем та Google OAuth. Для веб-платформи реалізовано окрему імплементацію Google Sign-In.

Архітектура автентифікації включає:

- **IAuthRepository** — інтерфейс у Domain Layer;
- **FirebaseAuthRepository** — реалізація з використанням Firebase Auth;
- **AuthBloc** — BLoC для керування станом автентифікації;
- **RegisterUseCase, LoginUseCase** — сценарії використання.

Firebase Auth забезпечує: створення облікових записів з email та паролем; автентифікацію через Google OAuth; управління сесіями; потік даних про стан авторизації через watchUser().

Для обробки помилок використовується функціональний тип Either з бібліотеки dartz, що дозволяє явно моделювати успішний та помилковий результат:

- UnauthorizedFailure — відсутність авторизації;
- UserExistsWithSuchEmailFailure — обліковий запис вже існує;
- LoginOrPasswordIncorrectFailure — невірні облікові дані;
- MaxTriesExceededException — перевищено кількість спроб.

Google Sign-In реалізовано з урахуванням платформних відмінностей через умовний імпорт:

```
import 'src/google_sign_in_service_mobile.dart'
  if (dart.library.html)
    'src/google_sign_in_service_web.dart';
```

Умовний імпорт у Dart дозволяє обирати різні реалізації залежно від платформи на етапі компіляції. Для мобільних платформ (Android, iOS) використовується пакет google_sign_in, який взаємодіє з нативним Google Sign-In SDK. Для веб-платформи використовується google_sign_in_web, який працює через JavaScript Google Identity Services API.

AuthBloc прослуховує зміни стану автентифікації через потік watchUser(). При успішному вході BLoC перевіряє наявність заповненого профілю (через GetCompletionStatusUseCase). Якщо профіль не заповнений, користувач перенаправляється на онбординг. Якщо профіль заповнений — на головний екран.

Безпека автентифікації забезпечується на кількох рівнях: Firebase Auth виконує валідацію облікових даних та генерацію JWT-токенів; Firestore Security Rules перевіряють автентифікацію при кожному запиті до бази даних; Cloud Functions перевіряють наявність auth-контексту перед виконанням операцій.

Для обробки помилок автентифікації визначено ієрархію типів відмов (Failure). Кожен тип відмови містить повідомлення, яке може бути відображено користувачу. Це забезпечує детальний зворотний зв'язок при помилках входу або реєстрації, що покращує користувацький досвід.

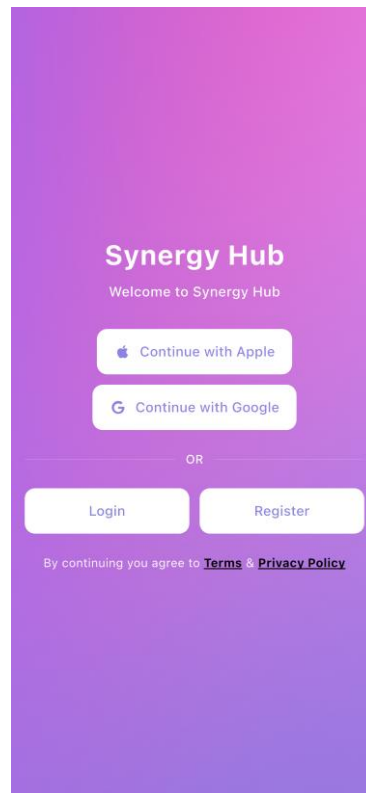


Рис. 3.3. Екран авторизації

3.5. Реалізація Discover-екрану та механізму свайпів

Discover-екран є центральним елементом взаємодії користувача з додатком. Він реалізований за допомогою двох BLoC-компонентів:

- **PeopleDataBloc** — завантаження та фільтрація кандидатів для перегляду;

– **TinderCardCubit** — керування жестами та анімацією картки.

TinderCardCubit обробляє три типи жестів:

– onPanStart — початок перетягування;

– onPanUpdate — оновлення позиції з розрахунком кута повороту;

– onPanEnd — визначення результату за пороговим значенням (75px).

Визначення статусу свайпу:

```
TinderCardStatus _getStatus(Offset position) {  
    const offset = 75.0;  
    if (position.dx > offset) return TinderCardStatus.liked;  
    if (position.dx < -offset) return TinderCardStatus.reject;  
    return TinderCardStatus.none;  
}
```

Після визначення рішення викликається Cloud Function `matchUser`, яка обробляє лайк або дизлайк у транзакції Firestore. Для запобігання повторного показу профілів використовується `GetExcludedMatchesUseCase`, який отримує список вже переглянутих UID.

Архітектура Discover-екрану побудована на принципі стеку карток (card stack). На екрані одночасно присутні дві картки: верхня (поточна, з якою взаємодіє користувач) та нижня (наступна, частково видима для створення ефекту глибини). Після свайпу поточної картки наступна стає поточною, а з сервера завантажується новий кандидат.

Візуальна індикація рішення реалізована через `BuildTinderStatus` — віджет, який відображає напис “LIKE” (зелений) або “NOPE” (червоний) з прозорістю, пропорційною зміщенню картки. Це забезпечує інтуїтивний зворотний зв’язок ще до відпускання картки.

Фільтрація кандидатів здійснюється через модель `MatchesFilter`:

```
const factory MatchesFilter({  
    EExperience? experience,  
    ESpecialization? specialization,  
    @Default([]) List<Skill> skills,  
})
```

3.6. Реалізація чатів та збереження повідомлень

Система чатів реалізована з використанням реальночасових Firestore Snapshot Listeners. Кожен чат зберігається як документ у колекції chats, а повідомлення — у вкладеній підколекції messages.

Архітектура чатів включає:

- **ChatBloc** — керування станом окремого чату (завантаження повідомлень, відправка, індикація набору тексту);
- **ChatsFilterBloc** — фільтрація списку чатів (усі, улюблені, архівовані);
- **IChatRepository** — інтерфейс для операцій з чатами.

Use Cases для чатів:

- ListenChatsUseCase — підписка на потік усіх чатів користувача;
- ListenChatByIdUseCase — підписка на оновлення конкретного чату;
- UpdateTypingInfoUseCase — трансляція статусу набору тексту;
- ToggleFavoriteChatUseCase — управління улюбленими чатами;
- SearchChatsUseCase — пошук чатів за ключовими словами.

Підписка на чати реалізується через emit.forEach у BLoC, що автоматично оновлює стан при надходженні нових даних з Firestore:

```
await emit.forEach(  
  _chatRepository.getStream(uid),  
  onData: (chats) => state.copyWith(  
    status: ChatStatus.loaded,  
    chats: chats,  
  ),  
);
```

Функціональність індикації набору тексту (typing indicator) реалізована через поле typingUsers у документі чату. При початку набору тексту UID користувача додається до масиву, при завершенні — видаляється. Оскільки Firestore забезпечує реальночасове оновлення, інші учасники чату миттєво бачать індикатор набору.

Система пошуку чатів базується на полі searchKeywords, яке містить посимвольні префікси імен учасників. Це дозволяє реалізувати пошук з

автодоповненням без використання зовнішніх пошукових сервісів (Algolia, Elasticsearch), що зменшує складність та вартість інфраструктури.

Конфігурація чату (ChatConfig) дозволяє кожному учаснику індивідуально архівувати або додавати чат до улюблених. Дані зберігаються у масивах `archived` та `favorite` всередині об'єкту `config` документа чату.

Модель повідомлення `ChatMessage` включає поля: `id` (унікальний ідентифікатор), `message` (текст повідомлення), `sentUid` (UID відправника) та `createdAt` (час створення). Повідомлення сортуються за часом створення для відображення у хронологічному порядку.

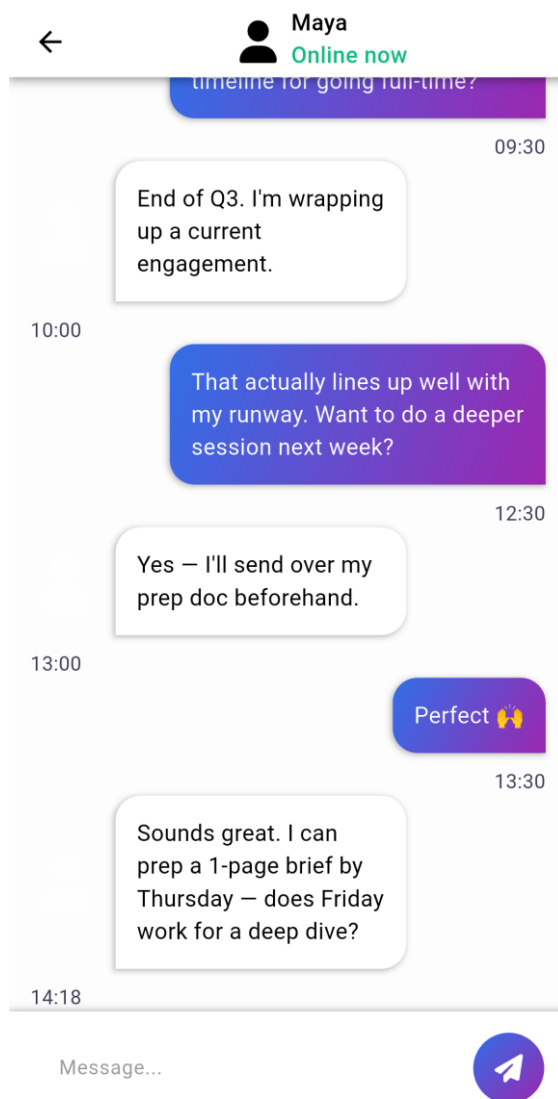


Рис 3.4. Екран чату з повідомленнями

3.7. Реалізація бізнес-логіки на основі патерну Use Case

Use Case (сценарій використання) є ключовим елементом Domain Layer, що інкапсулює одну конкретну бізнес-операцію. У застосунку Synergy Hub Use Cases використовуються як на клієнті (Dart), так і на сервері (TypeScript).

Клієнтські Use Cases (Dart). Кожен Use Case залежить від інтерфейсів репозиторіїв та не має прямих залежностей від Firebase чи Flutter. Приклади:

- LookForMatchesUseCase — пошук кандидатів з урахуванням фільтрів та виключенням переглянутих;
- CreateUserMatchesUseCase — фіксація рішення через виклик Cloud Function;
- ListenTwoWayMatchUseCase — підписка на потік взаємних матчів;
- ListenChatsUseCase — підписка на потік чатів;
- GetCompletionStatusUseCase — перевірка стану онбордингу.

Серверний Use Cases (TypeScript). На сервері Use Cases отримують ITransactionContext для виконання операцій у транзакції:

- MatchUseCase — обробка лайку/дизлайку в транзакції;
- ProcessTwoWayMatchesUseCase — створення чату та розрахунок сумісності;
- OnNewMessageUseCase — обробка нового повідомлення;
- SyncUserProfileInChatsUseCase — синхронізація профілю;
- SendNotificationsUseCase — відправка push-сповіщень.

Алгоритми оцінки сумісності реалізовані як окремі Use Cases:

- CalculateExperienceScoreUseCase — порівняння рівнів досвіду;
- CalculateSkillScoreUseCase — обчислення перетину навичок;
- CalculateVisionAlignUseCase — текстовий аналіз полів aboutMe.

Кожен Use Case дотримується принципу єдиної відповідальності (Single Responsibility Principle): один Use Case виконує одну бізнес-операцію. Це спрощує тестування, оскільки для кожного Use Case можна написати ізольовані юніт-тести з моками залежностей.

Патерн Use Case також забезпечує чітку точку входу для кожної бізнес-операції. BLoC-компоненти не працюють напряму з репозиторіями, а виключно через Use Cases. Це додає додатковий рівень абстракції, який дозволяє змінювати логіку операції (наприклад, додати валідацію або кешування) без модифікації Presentation Layer.

Обробка помилок у Use Cases реалізована через тип `Either<Failure, Result>` з бібліотеки `dartz`. `Left` містить об'єкт `Failure` з описом помилки, `Right` — успішний результат. Це дозволяє BLoC-компонентам елегантно обробляти обидва випадки:

```
final result = await loginUseCase.call(
  email: email, password: password,
);
result.fold(
  (failure) => emit(state.copyWith(
    status: AuthStatus.error,
    errorMessage: failure.message,
  )),
  (_) => emit(state.copyWith(
    status: AuthStatus.authorized,
  )),
);
```

3.8. Інтеграція з Firebase Firestore та Firestore Security Rules

Взаємодія з Firebase Firestore реалізована через набір сервісів, інкапсульованих у `core/services/firebase/`. Основним є `FirebaseFirestoreService`, який надає типізовані посилання на колекції та документи.

Для кожної колекції визначено типізовані псевдоніми:

```
typedef DocRef = DocumentReference<Map<String, dynamic>>;
typedef ColRef = CollectionReference<Map<String, dynamic>>;
typedef QuerySnap = QuerySnapshot<Map<String, dynamic>>;
```

Firestore Security Rules забезпечують безпеку даних на рівні бази. Основні правила:

- дані користувача доступні для читання та запису лише самому користувачу;
- чати доступні лише учасникам (перевірка наявності `UID` у масиві `participants`);

- повідомлення у чаті доступні лише учасникам чату;
- матчі доступні лише залученим користувачам;
- публічні дані профілю доступні для читання авторизованим користувачам.

Для оптимізації продуктивності використовуються складені індекси Firestore для запитів з фільтрацією та сортуванням. Кешування на клієнті забезпечується вбудованим механізмом Firestore offline persistence.

Для підтримки QA-тестування реалізовано підключення до Firebase Emulator:

```
if (AppConfig.isQA) {  
  await _firebaseAuth.useAuthEmulator(  
    AppConfig.emulatorAddress(),  
    AppConfig.authEmulatorPort,  
  );  
}
```

Firebase Emulator Suite дозволяє запускати локальні версії Auth, Firestore та Cloud Functions. Це забезпечує можливість тестування без підключення до хмарної інфраструктури, що прискорює цикл розробки та запобігає випадковим змінам у production-базі.

Конвертація типів між Firestore та Dart-моделями реалізована через кастомні конвертери. TimestampConverter автоматично перетворює Firestore Timestamp у Dart DateTime та навпаки. ColorSerializable забезпечує серіалізацію кольорів для збереження налаштувань теми.

Оптимізація запитів до Firestore включає: використання compound queries з фільтрацією на сервері замість клієнтської фільтрації; обмеження кількості документів через limit() для пагінації; використання orderBy() для сортування на рівні бази даних; кешування результатів через вбудований механізм Firestore offline persistence.

Взаємодія з Firestore у клієнтській частині здійснюється виключно через репозиторії Data Layer. Кожен репозиторій інкапсулює логіку конвертації DTO у доменні моделі та навпаки, обробку помилок та управління підписками на real-time оновлення.

3.9. Реалізація серверної логіки на основі Firebase Cloud Functions (TypeScript)

Серверна логіка застосунку реалізована у вигляді трьох Firebase Cloud Functions на TypeScript. Серверний проєкт використовує Node.js 20 та дотримується принципів Clean Architecture з Dependency Injection.

3.9.1. Функція матчингу користувачів у транзакції (matchUser)

Функція matchUser є Callable HTTPS Function, яка викликається з клієнта для обробки рішення користувача (лайк або дизлайк). Усі операції виконуються в рамках єдиної Firestore-транзакції через патерн Unit of Work.

Алгоритм роботи:

- перевірка автентифікації та валідація вхідних даних;
- запобігання самотчингу (userId == matchId);
- генерація унікального ключа матчу через сортовану конкатенацію UID;
- обробка дизлайку — оновлення запису з completeness: none;
- перевірка наявності зустрічного лайку;
- при взаємному лайку — виклик ProcessTwoWayMatchesUseCase;
- при односторонньому лайку — створення нового запису та відправка сповіщення.

Фрагмент точки входу Cloud Function:

```
export const matchUser = onCall(  
  async (req) => {  
    const { data, auth } = req;  
    if (!auth) throw new HttpsError(  
      'unauthenticated', 'User must be authenticated'  
    );  
    await container.matchUseCase.call(  
      auth.uid, data.matchId, data.status  
    );  
    return { success: true };  
  }  
);
```

При двосторонньому матчі ProcessTwoWayMatchesUseCase виконує:

- завантаження профілів обох користувачів;
- розрахунок Experience Score (max(10 – diff × 3, 2));

- розрахунок Skills Score (перетин навичок / максимум);
- розрахунок Vision Score (текстовий аналіз полів aboutMe);
- створення чату між користувачами;
- відправку push-сповіщень обом учасникам.

Повний лістинг функції matchUser наведено у Додатку А.

Алгоритм розрахунку Experience Score базується на відстані між рівнями досвіду. Рівні впорядковані: Junior (0), Middle (1), Senior (2), Lead/Expert (3). Формула: $score = \max(10 - |\text{pos_a} - \text{pos_b}| \times 3, 2)$. Наприклад, два Senior-розробники отримають максимальний бал 10, а пара Junior-Senior — бал 4.

Алгоритм Skills Score обчислює перетин множин навичок: $\text{intersection} = \text{skills_a} \cap \text{skills_b}$; $score = (|\text{intersection}| / \max(|\text{skills_a}|, |\text{skills_b}|)) \times 10$. Наприклад, якщо у першого користувача навички [Flutter, Firebase, Dart], а у другого [Flutter, React, Dart], перетин = [Flutter, Dart], $score = 2/3 \times 10 \approx 6.7$.

Vision Score використовує текстовий аналіз полів aboutMe. Алгоритм: нормалізація тексту (lowercase, видалення пунктуації) → розбиття на слова → фільтрація стоп-слів (a, the, and, startup, project тощо) → фільтрація коротких слів (менше 3 символів) → обчислення подібності як відношення перетину до максимуму $\times 10$.

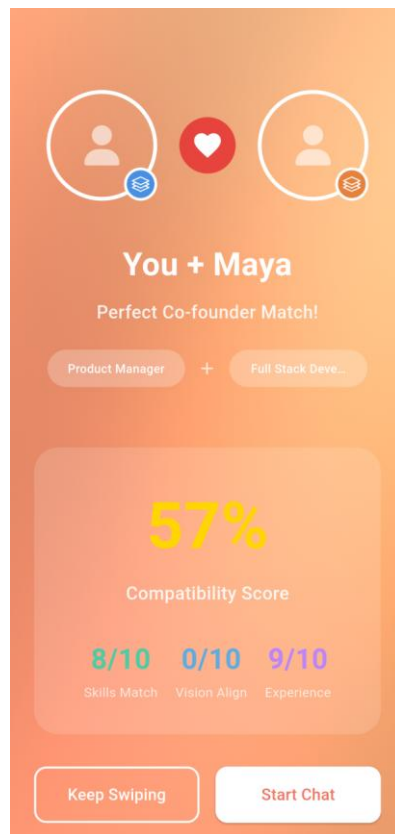


Рис. 3.5. Екран взаємного матчу

3.9.2. Обробка нових повідомлень та розсилка push-сповіщень (onNewMessage)

Функція onNewMessage є Firestore Trigger, що активується при створенні нового документа за шляхом `chats/{chatId}/messages/{msgId}`. Вона забезпечує відправку push-сповіщень усім учасникам чату, окрім відправника повідомлення.

Алгоритм роботи:

- витягнення chatId та даних повідомлення (sentUid, message);
- завантаження документа чату для отримання списку учасників;
- фільтрація — виключення відправника зі списку отримувачів;
- для кожного отримувача: завантаження профілю, отримання FCM-токенів, відправка сповіщення;
- збереження сповіщення у Firestore (історія сповіщень).

Тіло сповіщення містить ім'я відправника та текст повідомлення. Для Android встановлюється пріоритет high, для iOS — стандартний звук default.

Сповіщення включає payload з метаданими: chatId (для навігації до відповідного чату), sentUid (для ідентифікації відправника) та type: “chat” (для визначення типу сповіщення на клієнті). Клієнтський застосунок використовує ці метадані для навігації користувача безпосередньо до чату при натисканні на сповіщення.

Окрім push-сповіщення, OnNewMessageUseCase також зберігає запис у колекції notifications кожного отримувача. Це забезпечує історію сповіщень навіть у випадку недоставки push-повідомлення (наприклад, коли FCM-токен застарів або пристрій не підключений до мережі).

Продуктивність функції оптимізована через паралельне завантаження профілів та FCM-конфігурацій учасників. Для чатів з двома учасниками (поточна реалізація) це означає лише одне додаткове читання з бази даних, оскільки відправник виключається зі списку отримувачів.

3.9.3. Синхронізація профілю користувача в чатах (onUserUpdate)

Функція onUserUpdate є Firestore Trigger на шлях users/{userId}, який активується при оновленні документа користувача. Її призначення — синхронізація інформації профілю у всіх чатах, де бере участь користувач.

Функція порівнює стан before та after оновлення. Якщо змінилися firstName, lastName або avatarUrl, виконується оновлення participantInfo у кожному чаті користувача через SyncUserProfileInChatsUseCase.

Оновлення використовує nested field path для атомарної зміни:

```
const fieldPath = `participantInfo.${userId}`;  
ref.update({  
  [fieldPath]: {  
    name: newName,  
    specialization: newSpecialization,  
    avatarUrl: avatarUrl,  
  }  
});
```

3.10. Інтеграція Firebase Cloud Messaging для push-сповіщень

Push-сповіщення у Synergy Hub реалізовані через Firebase Cloud Messaging (FCM). Інтеграція включає клієнтську та серверну частини.

Клієнтська частина. FirebaseMessagingService відповідає за:

- запит дозволу на сповіщення (iOS);
- отримання та збереження FCM-токена пристрою;
- обробку foreground-повідомлень через `onMessage`;
- реєстрацію background handler для повідомлень у фоновому режимі;
- інтеграцію з `LocalNotificationService` для відображення локальних сповіщень.

Для веб-платформи додатково використовується VAPID-ключ для отримання push-сертифікату.

Серверна частина. Відправка сповіщень здійснюється через `SendNotificationsUseCase`, який використовує `Firebase Admin SDK` (`FCMService`). Кожне сповіщення включає:

- `title` та `body` — заголовок та текст;
- `data` — `payload` з типом події (`match` або `chat`) та ідентифікаторами;
- платформні налаштування (`Android`: `high priority`, `iOS`: `default sound`).

Сповіщення також зберігаються у `Firestore` (колекція `notifications користувача`), що забезпечує історію сповіщень навіть при недоставці push-повідомлення.

Конфігурація сповіщень зберігається у підколекції `notificationConfig` кожного користувача. `NotificationConfig` містить: `enabledNotifications` (прапорець увімкнення/вимкнення сповіщень) та `fcmTokens` (масив токенів пристроїв). Один користувач може мати кілька FCM-токенів, якщо використовує додаток на різних пристроях або платформах.

Обробка foreground-повідомлень на клієнті реалізована через `FirebaseMessaging.onMessage stream`. При отриманні повідомлення в foreground-режимі створюється локальне сповіщення через `flutter_local_notifications`, оскільки FCM за замовчуванням не відображає сповіщення, коли додаток активний.

Background handler реєструється одноразово при ініціалізації додатку через `FirebaseMessaging.onBackgroundMessage`. Ця функція виконується в

окремому ізоляті (isolate) Dart, що вимагає мінімальної ініціалізації Firebase без доступу до стану додатку.

Для веб-платформи push-сповіщення обробляються через Service Worker. FCM Web SDK автоматично реєструє Service Worker, який отримує повідомлення навіть коли вкладка браузера неактивна. VAPID-ключ (Voluntary Application Server Identification) використовується для ідентифікації сервера при відправці push-повідомлень.

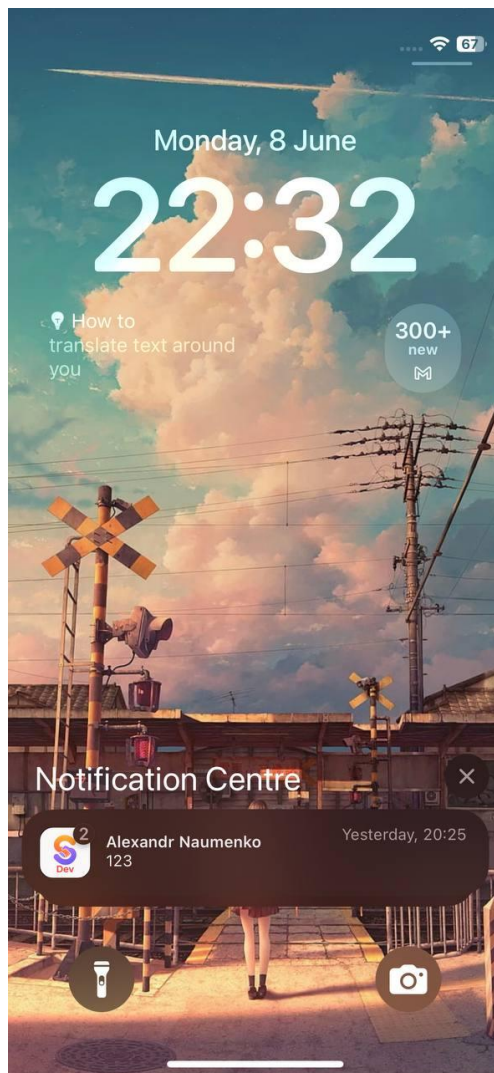


Рис 3.6. push-сповіщення отримане з чату

Висновки до розділу

У даному розділі було представлено реалізацію мобільного застосунку Synergy Hub, побудованого на основі сучасних архітектурних підходів та технологій Flutter і Dart. Детально описано клієнтську частину системи, яка

організована за принципами Clean Architecture та фічерної модульності, що забезпечує ізолюваність функціональних компонентів, простоту масштабування та подальшого супроводу проєкту. Використання BLoC/Cubit як основного підходу до керування станом дозволило чітко розділити бізнес-логіку та UI, а застосування генераторів коду (Freezed, JsonSerializable, go_router_builder) підвищило надійність та зменшило обсяг ручного коду.

Окрему увагу приділено реалізації ключових підсистем застосунку, зокрема автентифікації користувачів, механізму матчіну, чатів у режимі реального часу, а також Discover-екрану зі свайп-взаємодією. Продемонстровано використання патерну Repository, Use Case та Dependency Injection (GetIt), що забезпечило чітке розмежування шарів системи та інверсію залежностей. Серверна логіка реалізована через Firebase Cloud Functions із використанням транзакційного підходу (Unit of Work), що гарантує цілісність даних при виконанні критичних операцій.

Додатково реалізовано механізми push-сповіщень через Firebase Cloud Messaging, інтеграцію з Firestore у режимі реального часу, а також систему безпеки на рівні Firestore Security Rules. Візуальна частина застосунку доповнена GPU-оптимізованими GLSL-шейдерами, що забезпечують плавну анімацію та покращують користувацький досвід без додаткового навантаження на основний потік.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено кросплатформенний мобільний застосунок Synergy Hub для формування стартап-команд. Під час роботи було виконано всі поставлені завдання та досягнуто мети дослідження.

Основні результати роботи:

1. Проаналізовано предметну область формування стартап-команд та існуючі програмні рішення (Tinder, LinkedIn, Meetup). Визначено їх основні недоліки: відсутність спеціалізації на стартапах, відсутність алгоритмів оцінки сумісності та обмеженість комунікаційних інструментів.

2. Сформовано вимоги до програмного продукту та обґрунтовано концепцію застосунку Synergy Hub, що поєднує свайп-механізм пошуку, професійний профіль, алгоритм матчингу та інтегровану систему чатів.

3. Спроектовано архітектуру програмного забезпечення на основі Clean Architecture з розподілом на рівні Domain, Data та Presentation. Спроектовано базу даних Firebase Firestore з ієрархічною структурою колекцій.

4. Реалізовано клієнтську частину на Flutter та Dart з використанням патерну BLoC для керування станом, GoRouter для типобезпечної маршрутизації та GetIt для впровадження залежностей. Реалізовано анімований фон з використанням GLSL-шейдерів.

5. Реалізовано серверну логіку на основі Firebase Cloud Functions (TypeScript) з трьома Cloud Functions: matchUser (матчинг у транзакції), onNewMessage (push-сповіщення), onUserUpdate (синхронізація профілю). Застосовано патерн Unit of Work для забезпечення атомарності транзакцій.

6. Реалізовано алгоритм тривимірної оцінки сумісності учасників за параметрами досвіду, технічних навичок та бачення проєкту.

7. Проведено тестування застосунку на кількох рівнях: юніт-тести серверної логіки (Jest), тестування UI-компонентів (flutter_test), ручне тестування на реальних пристроях. Налаштовано CI/CD через Codemagic для автоматичного деплою веб-версії на Firebase Hosting.

Розроблений мобільний застосунок Synergy Hub є працездатним програмним продуктом, який може бути використаний для формування стартап-команд та має потенціал для подальшого розвитку та комерціалізації.

Застосування сучасних архітектурних підходів (Clean Architecture, BLoC, Repository Pattern, Unit of Work) забезпечило високу якість кодової бази, її тестованість та можливість масштабування. Використання Firebase як Backend-as-a-Service дозволило мінімізувати витрати на інфраструктуру та зосередитися на розробці бізнес-логіки.

Ключовою інновацією є тривимірна модель оцінки сумісності учасників (Experience Score, Skills Score, Vision Score), яка автоматизує процес підбору партнерів та надає кількісні показники якості матчу. Це принципово відрізняє Synergy Hub від існуючих рішень, які покладаються виключно на ручний підбір.

Перспективи подальшого розвитку включають кілька ключових напрямків:

- Удосконалення алгоритмів матчингу — впровадження машинного навчання для аналізу поведінкових патернів користувачів, використання NLP-моделей (sentence embeddings) для семантичного порівняння текстів Vision Score замість простого перетину слів, а також побудова рекомендаційної системи на основі колаборативної фільтрації.
- Розширення функціоналу фільтрації — додавання фільтрів за геолокацією (з використанням GeoFlutterFire), мовами спілкування, доступністю (повна/часткова зайнятість) та розширеною системою тегів. Завдяки обраній архітектурі це потребуватиме змін лише на рівні Data та Presentation layers без модифікації серверних Cloud Functions.
- Розширення платформ — публікація в Google Play та App Store після завершення програми закритого бета-тестування, оптимізація адаптивного макету для планшетів, а також розробка десктопних версій для macOS та Windows.

- Масштабування інфраструктури — використання автоматичних механізмів Firebase (горизонтальне масштабування Cloud Functions, розподіл даних Firestore, CDN для веб-версії), впровадження A/B тестування через Firebase Remote Config та інтеграція Firebase Analytics для відстеження поведінки користувачів.
- Монетизація — триступенева модель: Free tier з обмеженою кількістю свайпів на день, Premium з необмеженими свайпами, розширеними фільтрами та пріоритетним відображенням профілю, і Enterprise-план для інкубаторів та акселераторів з окремою панеллю управління.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bloch J., Effective Java. Boston: Pearson Education (US), 2018. 416 p.
2. Fain Y., Moiseev A. Angular Development with TypeScript. New York: Manning Publications, 2019. 560 p.
3. Fain Y., Moiseev A. Angular Development with TypeScript. New York: Manning Publications, 2019. 560 p.
4. Git [Електронний ресурс] // Режим доступу: <https://git-scm.com/about>
5. Martin R., Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston: Pearson Education (US), 2017. 432 p.
6. Flutter documentation. URL: <https://docs.flutter.dev/> (дата звернення: 15.03.2026).
7. Dart programming language. URL: <https://dart.dev/> (дата звернення: 15.03.2026).
8. Firebase documentation. URL: <https://firebase.google.com/docs> (дата звернення: 15.03.2026).
9. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017. 432 p.
10. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994. 395 p.
11. Bloc State Management Library. URL: <https://bloclibrary.dev/> (дата звернення: 15.03.2026).
12. GoRouter package for Flutter. URL: https://pub.dev/packages/go_router (дата звернення: 15.03.2026).
13. GetIt package for Dependency Injection. URL: https://pub.dev/packages/get_it (дата звернення: 15.03.2026).
14. Freezed package for immutable data classes. URL: <https://pub.dev/packages/freezed> (дата звернення: 15.03.2026).
15. Firebase Cloud Functions documentation. URL: <https://firebase.google.com/docs/functions> (дата звернення: 15.03.2026).
16. Firebase Cloud Firestore documentation. URL:

- <https://firebase.google.com/docs/firestore> (дата звернення: 15.03.2026).
17. Firebase Authentication documentation. URL: <https://firebase.google.com/docs/auth> (дата звернення: 15.03.2026).
18. Firebase Cloud Messaging documentation. URL: <https://firebase.google.com/docs/cloud-messaging> (дата звернення: 15.03.2026).
19. TypeScript documentation. URL: <https://www.typescriptlang.org/docs/> (дата звернення: 15.03.2026).
20. Jest Testing Framework. URL: <https://jestjs.io/> (дата звернення: 15.03.2026).
21. Codemagic CI/CD for Flutter. URL: <https://codemagic.io/> (дата звернення: 15.03.2026).
22. Material Design guidelines. URL: <https://m3.material.io/> (дата звернення: 15.03.2026).
23. OpenGL Shading Language (GLSL). Khronos Group. URL: https://www.khronos.org/opengl/wiki/OpenGL_Shading_Language (дата звернення: 15.03.2026).
24. Ries E. The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses. Crown Business, 2011. 336 p.
25. Blank S. The Four Steps to the Epiphany: Successful Strategies for Products that Win. K&S Ranch, 2020. 370 p.
26. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley, 2002. 560 p.
27. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley, 2003. 560 p.
28. Windmill E. Flutter in Action. Manning Publications, 2020. 368 p.
29. Zaccagnino A. Programming Flutter: Native, Cross-Platform Apps the Easy Way. Pragmatic Bookshelf, 2020. 370 p.
30. Google Cloud Platform documentation. URL: <https://cloud.google.com/docs> (дата звернення: 15.03.2026).

31. Node.js documentation. URL: <https://nodejs.org/docs/> (дата звернення: 15.03.2026).
32. RxDart package. URL: <https://pub.dev/packages/rxdart> (дата звернення: 15.03.2026).
33. Dartz package for functional programming. URL: <https://pub.dev/packages/dartz> (дата звернення: 15.03.2026)
34. Firebase Hosting documentation. URL: <https://firebase.google.com/docs/hosting> (дата звернення: 15.03.2026).
35. Cloud Functions налаштування
https://medium.com/@theapp_forge/a-deep-dive-into-firebase-cloud-functions-with-flutter-ea5710ee7ffd?source=tag_archive-----3-----
36. Налаштування Dev/Stage/Debug середовищ:
<https://medium.com/@towhae/production-ready-firebase-setup-for-flutter-flavors-dev-staging-production-crashlytics-7380c5b86026> (дата звернення: 15.03.2026).

ДОДАТОК А

Лістинг функції MatchUseCase (TypeScript)

```
import {CompletenessStatus, EMatchingStatus, Matching}
    from "../models/matching";

export class MatchUseCase {
    constructor(
        private readonly sendNotificationUseCase,
        private readonly processTwoWayMatchesUseCase,
        private matchesIdentifierService,
        private getNotificationConfigUseCase,
        private unitOfWork,
    ) {}

    async call(
        userId: string, matchId: string,
        matchStatus: EMatchingStatus
    ) {
        if (userId == matchId) return;

        await this.unitOfWork.run(async (ctx) => {
            const uid = this.matchesIdentifierService
                .matchesKey(userId, matchId);
            const possibleMatch = await ctx.matches
                .getById(uid);
            const found = possibleMatch !== undefined;

            const config = await this
                .getNotificationConfigUseCase
                .call(userId, ctx);
            const tokens = config?.fcmTokens ?? [];

            if (matchStatus == EMatchingStatus.reject) {
                await ctx.matches.update(uid,
                    [...new Set([userId,
                        ...(possibleMatch?.answeredUids ?? [])]),
                    ],
                    { [userId]: possibleMatch?.userMatches
                        [userId] ?? false,
                        [matchId]: false },
                    matchStatus, CompletenessStatus.none,
                    0, 0, 0);
                return;
            } else if (found && possibleMatch!
                .completeness == CompletenessStatus.none) {
                return;
            }

            if (found) {
                const otherLiked = possibleMatch
                    ?.userMatches[matchId] === true;
                if (otherLiked) {
                    await this.processTwoWayMatchesUseCase
                        .call(userId, matchId, tokens, ctx);
                }
                return;
            }
        })
    }
}
```

```

    await ctx.matches.create(uid,
      [userId, matchId], [userId],
      { [userId]: true, [matchId]: false },
      matchStatus,
      CompletenessStatus.oneWayMatch,
      0, 0, 0);

    await this.sendNotificationUseCase.call(
      matchId, {
        notification: {
          title: 'New Interest',
          body: 'A potential co-founder liked'
        },
        data: { type: 'match', matchId }
      }, tokens, ctx);
  });
}
}

```

ДОДАТОК Б

Структура бази даних Firebase Firestore

Нижче наведено повну ієрархічну структуру бази даних:

```
users/{userId}
├── uid: string
├── firstName: string
├── lastName: string
├── age: number
├── avatarUrl: string
├── aboutMe: string
├── profiler:
│   ├── experience: junior|middle|senior|leadExpert
│   ├── role: string
│   ├── specialization: string
│   ├── skills: string[]
│   └── searchingSpecializations: string[]
├── onboardStatus/ (sub-collection)
│   └── primary: {completeSpecialization, toldAboutYourself}
├── notificationConfig/ (sub-collection)
│   └── config: {enabledNotifications, fcmTokens[]}
├── notifications/ (sub-collection)
│   └── {docId}: {id, title, description, hasRead}
└── chats/ (sub-collection)
    └── {chatId}: {chatId}

chats/{chatId}
├── chatId: string
├── participants: string[]
├── participantInfo: Map<uid, {name, specialization, avatarUrl}>
├── lastMessage: string
├── lastMessageSent: Timestamp
├── typingUsers: string[]
├── searchKeywords: string[]
├── config: {archived: string[], favorite: string[]}
└── messages/ (sub-collection)
    └── {msgId}: {id, message, sentUid, createdAt}

matches/{matchId} // matchId = sorted uid1_uid2
├── matchId: string
├── users: string[]
├── answeredUids: string[]
├── userMatches: Map<uid, boolean>
├── matchStatus: liked|reject
├── completeness: none|oneWayMatch|twoWayMatch
├── experienceScore: number (0-10)
├── skillsMatch: number (0-10)
└── visionScore: number (0-10)
```


ДОДАТОК В

Конфігурація CI/CD Codemagic

```
workflows:
  web-deploy:
    name: Web Deploy
    environment:
      flutter: 3.41.7
    groups:
      - env
      - files
      - notifier
    vars:
      FIREBASE_PROJECT_ID: synergyhub-a3e1b
    cache:
      cache_paths:
        - ~/.pub-cache
        - .flutter
        - build/
    scripts:
      - name: Prepare
        script: set -euo pipefail
      - name: Get packages
        script: flutter pub get
      - name: Code generation
        script: dart run build_runner build -d
      - name: Create firebase.json file
        script: |
          cp "$FIREBASE_JSON" "$CM_BUILD_DIR/firebase.json"
      - name: Create .env file
        script: |
          echo "$env" > .env
      - name: Build web
        script: |
          flutter build web \
            --dart-define=DARTENV=development \
            --dart-define=DEV_SERVER_CLIENT_ID="$DEV_SERVER_CLIENT_ID" \
            --dart-define=DEV_WEB_PUSH_CERT="$DEV_WEB_PUSH_CERT" \
            --dart-define=WEB_URL="$WEB_URL" \
            --release
      - name: Deploy to Firebase
        script: |
          firebase deploy \
            --only hosting \
            --project $FIREBASE_PROJECT_ID \
            --token $FIREBASE_TOKEN
      - name: Telegram notify
        script: |
          if [ "$CM_BUILD_STATUS" = "success" ]; then
            STATUS="✅ Build succeeded"
          else
            STATUS="❌ Build failed"
          fi

          DEPLOY_URL="https://$FIREBASE_PROJECT_ID.web.app"
          MESSAGE="$STATUS%0A🚀 Deploy: $DEPLOY_URL%0A🏠 Build: $CM_BUILD_ID"
          curl -X POST \
            "https://api.telegram.org/bot$TELEGRAM_BOT_TOKEN/sendMessage" \
            -d chat_id=$TELEGRAM_CHAT_ID \
            -d text="$MESSAGE"

ios-workflow:
```

```

name: iOS Workflow
triggering:
  events:
    - push
  branch_patterns:
    - pattern: main
      include: true
cache:
  cache_paths:
    - ~/.pub-cache
    - .flutter
    - build/
    - ~/.cocoapods
    - ios/Pods
max_build_duration: 120
instance_type: mac_mini_m2
integrations:
  app_store_connect: sinergyhub
environment:
  groups:
    - store_credentials
    - files
    - env
  ios_signing:
    distribution_type: app_store
    bundle_identifier: $IOS_BUNDLE_ID
vars:
  BUNDLE_ID: "$IOS_BUNDLE_ID"
  APP_STORE_APPLE_ID: $APPSTORE_APP_ID
flutter: 3.41.1
xcode: latest
cocoapods: default
scripts:
  - name: Set up provisioning profiles
    script: xcode-project use-profiles --verbose
  - name: Clean Flutter project
    script: flutter clean
  - name: Get Flutter packages
    script: flutter pub get
  - name: Create firebase.json file
    script: |
      cat <<EOF > "$CM_BUILD_DIR/firebase.json"
      $FIREBASE_JSON
      EOF
  - name: Activate FlutterFire CLI
    script: |
      dart pub global activate flutterfire_cli
  - name: Install pods
    script: |
      find . -name "Podfile" -execdir pod install \;
  - name: Flutter build ipa
    script: |
      flutter build ipa \
        --release \
        --export-options-plist=ios/export_options.plist \
        --flavor=prod \
        --dart-define=DARTENV=prod \
        --dart-define=CLIENT_ID="$CLIENT_ID" \
        --dart-define=PUSH_CERTIFICATE="$PUSH_CERTIFICATE"
artifacts:
  - build/ios/ipa/*.ipa
  - /tmp/xcodebuild_logs/*.log
  - flutter_drive.log
  - $HOME/Library/Developer/Xcode/DerivedData/**/Build/**/*.*app

```

```
- $HOME/Library/Developer/Xcode/DerivedData/**/Build/**/*dSYM
publishing:
  app_store_connect:
    api_key: $APPSTORE_P8
    key_id: $APPSTORE_KEY_ID
    issuer_id: $APPSTORE_ISSUER_ID
    submit_to_testflight: true
    submit_to_app_store: false
```

Міністерство освіти і науки України
Державний заклад «Луганський національний університет
імені Тараса Шевченка»
Факультет Факультет технологій та інформаційних систем
(повна назва)
Кафедра Інформаційних технологій та систем
(повна назва)

Програма та методика тестування
на виконання програмної розробки (ПР) :
"МОБІЛЬНИЙ ДОДАТОК ДЛЯ ФОРМУВАННЯ СТАРТАП-КОМАНДИ"

ІТС. ІПЗ4.0126-03-МТ

ПОГОДЖЕНО
Керівник кваліфікаційної роботи

Володимир ДОНЧЕНКО

“ _____ ” _____ 2026р.

ВИКОНАВЕЦЬ
Студент групи 4 ІПЗ МС

Олександр НАУМЕНКО

“ _____ ” _____ 2026р.

Лубни 2026

ЗМІСТ

1. ОБ'ЄКТ ВИПРОБУВАНЬ.....	3
2. МЕТА ТЕСТУВАННЯ.....	4
3. РЕЗУЛЬТАТИ ВИПРОБУВАНЬ	5

1. ОБ'ЄКТ ВИПРОБУВАНЬ

Об'єктом випробувань є мобільний застосунок «Synergy Hub», призначений для підтримки процесу формування стартап-команд та комунікації між їхніми учасниками. Застосунок реалізовано на основі фреймворку Flutter (мова Dart) із серверною частиною на Firebase Cloud Functions (TypeScript).

Мобільний додаток повинен виконувати роль комунікатора між учасниками стартап-проектів — від моменту пошуку партнера до обміну повідомленнями всередині команди.

Система має виконувати наступні функції для всіх типів користувачів:

— система повинна надавати користувачу можливість створити персональний акаунт, який дозволить переглядати профілі інших учасників, фіксувати рішення та спілкуватися у чатах після матчу.

Для типу користувача «Учасник стартап-команди» система повинна виконувати наступні функції:

1. багаторічний онбординг для збору даних про спеціалізацію, досвід, навички та роль у команді;
2. перегляд профілів інших користувачів у форматі свайп-карток;
3. фіксація рішень користувача (лайк/дизлайк) та збереження історії взаємодій;
4. автоматичне створення чату між учасниками після взаємного матчу;
5. обмін текстовими повідомленнями у реальному часі;
6. отримання push-сповіщень про нові матчі та повідомлення;
7. редагування власного профілю користувача;
8. фільтрація користувачів за спеціалізацією та досвідом.

2. МЕТА ТЕСТУВАННЯ

Метою тестування мобільного застосунку «Synergy Hub» та його серверної частини є перевірка коректності роботи всіх необхідних функцій відповідно до вимог технічного завдання. Тестування охоплює модульну перевірку ключових алгоритмів сумісності користувачів (skill, experience, vision), логіку взаємодій між учасниками та обопільного матчу, генерацію пошукових ключів для чатів, інфраструктурний шар викликів Firebase Cloud Functions, а також BLoC-логіку автентифікації, чатів та use case надсилання повідомлень.

3. РЕЗУЛЬТАТИ ВИПРОБУВАНЬ

У таблицях 3.1–3.10 наведено тести випробування підсистем мобільного застосунку та серверної логіки.

Таблиця 3.1.

Тест випробування реєстрації та автентифікації через Google

Назва:	Тест автентифікації користувача через Google OAuth	
Функція/UseCase:	AuthBloc + SignInWithGoogleUseCase	
Дія:	Очікуваний результат:	Результат тесту:
Користувач натискає кнопку «Увійти через Google»	Користувач авторизований, відкривається головний екран застосунку	• пройдений • провалений • заблокований
Передумова:		
Запустити мобільний застосунок Synergy Hub	Відкрито стартовий екран із кнопкою входу	Пройдений
Перевірити, що користувач не авторизований	Стан AuthState.logout	Пройдений
Кроки тесту:		
Натиснути на кнопку «Увійти через Google»	Відкрито діалог вибору Google-акаунта	Пройдений
Обрати акаунт Google	AuthBloc переходить у стан loading	Пройдений
Дочекатися завершення автентифікації	AuthBloc переходить у стан authorized	Пройдений
Післяумова:		
Перевірити поточний стан застосунку	Відкритий головний екран із каруселлю профілів	Пройдений

Натиснути на іконку профілю	Відкритий екран з даними поточного користувача	Пройдений
-----------------------------	--	-----------

Таблиця 3.2.

Тест випробування багатоетапного онбордингу користувача

Назва:	Тест проходження онбордингу новим користувачем	
Функція/UseCase:	OnboardingBloc + SetCompletionStatusUseCase	
Дія:	Очікуваний результат:	Результат тесту:
Користувач послідовно заповнює крок онбордингу: спеціалізація, досвід, навички, роль, опис	Профіль користувача збережено у Firestore, прапор completion = true	• пройдений • провалений • заблокований
Передумова:		
Авторизуватися як новий користувач	Відкритий екран онбордингу, крок 1	Пройдений
Перевірити, що completion = false	Користувач знаходиться у потоці онбордингу	Пройдений
Кроки тесту:		
Обрати спеціалізацію та натиснути «Далі»	Виконано перехід на крок 2 (досвід)	Пройдений
Обрати рівень досвіду та натиснути «Далі»	Виконано перехід на крок 3 (навички)	Пройдений
Обрати ключові навички та натиснути «Далі»	Виконано перехід на крок 4 (роль)	Пройдений
Обрати бажану роль у команді	Виконано перехід на крок 5 (опис)	Пройдений

Заповнити поле «Про себе» та натиснути «Завершити»	Профіль збережено; completion = true	Пройдений
Післяумова:		
Перевірити стан застосунку	Відкритий головний екран з каруселлю профілів	Пройдений
Перезайти в застосунок	Онбординг більше не відображається	Пройдений

Таблиця 3.3.

Тест випробування свайп-перегляду профілів та фіксації лайку

Назва:	Тест лайку профілю іншого користувача	
Функція/UseCase:	TinderCardCubit + CreateUserMatchesUseCase + MatchUseCase	
Дія:	Очікуваний результат:	Результат тесту:
Користувач свайпає картку профілю праворуч (лайк)	У Firestore створено запис Matching зі статусом liked для пари користувачів	• пройдений • провалений • заблокований
Передумова:		
Авторизуватися та пройти онбординг	Відкритий головний екран з каруселлю	Пройдений
Дочекатися завантаження карток профілів	Відображено перший профіль у каруселі	Пройдений
Кроки тесту:		
Свайпнути картку профілю праворуч	Картка зникає з екрана з анімацією	Пройдений

Перевірити виклик firebase function matchUser	Виклик з data {matchId, status: 'liked'} виконано	Пройдений
Перевірити створення запису в колекції matches	Створено документ із completeness: oneWayMatch	Пройдений
Відобразити наступний профіль	У каруселі з'явилася наступна картка	Пройдений
Післяумова:		
Перейти на екран «Мої матчі»	У списку відображається запис про лайкнутого користувача	Пройдений

Таблиця 3.4.

Тест випробування обопільного матчу та push-сповіщення

Назва:	Тест автоматичного створення чату та сповіщення при матчі	
Функція/UseCase:	MatchUseCase + ProcessTwoWayMatchesUseCase + SendNotificationsUseCase	
Дія:	Очікуваний результат:	Результат тесту:
Другий користувач лайкає того, хто його вже лайкнув	completeness переходить у twoWayMatch, створюється чат, обом користувачам надсилаються push-сповіщення	• пройдений • провалений • заблокований
Передумова:		
User1 лайкає user2	Створено запис Matching у стані oneWayMatch	Пройдений
Перевірити, що у user2 в fcmTokens є валідний токен	fcmTokens: ['target-token']	Пройдений

Кроки тесту:		
User2 виконує свайп праворуч на профілі user1	Викликано matchUser зі статусом liked	Пройдений
Перевірити стан Matching після обробки сервером	completeness = twoWayMatch	Пройдений
Перевірити виклик ProcessTwoWayMatchesUseCase	Метод виконано один раз	Пройдений
Перевірити створення чату між user1 та user2	У колекції chats з'явився документ із обома учасниками	Пройдений
Перевірити надсилання push-сповіщень	sendNotificationsUseCase викликано для обох користувачів	Пройдений
Післяумова:		
Відкрити екран «Чати» в обох користувачів	У списку чатів з'явився новий чат	Пройдений

Таблиця 3.5.

Тест випробування алгоритму оцінки сумісності за навичками

Назва:	Тест CalculateSkillScoreUseCase	
Функція/UseCase:	CalculateSkillScoreUseCase	
Дія:	Очікуваний результат:	Результат тесту:
Викликати функцію розрахунку сумісності за набором навичок двох користувачів	Повернено значення у діапазоні $[0; 10]$, розраховане як $ A \cap B / \max(A , B) \times 10$	• пройдений • провалений • заблокований
Передумова:		
Підготувати два мок-користувачі з різними наборами навичок	Мок-об'єкти створені	Пройдений

Кроки тесту:		
Викликати з порожніми навичками: skills=[], []	Повернено 0	Пройдений
Викликати з одним порожнім: ['flutter'], []	Повернено 0	Пройдений
Викликати з однаковими: ['flutter','dart'] × 2	Повернено 10.0	Пройдений
Викликати з частковим збігом: ['flutter','firebase','dart'] vs ['flutter','react','dart']	Повернено 6.7	Пройдений
Викликати без спільних: ['java'] vs ['python']	Повернено 0	Пройдений
Перевірити симетричність: f(a,b) vs f(b,a)	Значення рівні	Пройдений
Післяумова:		
Перевірити, що результат не виходить за межі [0; 10]	Результат у заданому діапазоні	Пройдений

Таблиця 3.6.

Тест випробування алгоритму оцінки сумісності за досвідом

Назва:	Тест CalculateExperienceScoreUseCase	
Функція/UseCase:	CalculateExperienceScoreUseCase	
Дія:	Очікуваний результат:	Результат тесту:
Викликати функцію розрахунку сумісності для двох рівнів досвіду	Повернено бал, що зменшується зі збільшенням різниці між рівнями (junior→middle→senior→leadExpert)	• пройдений • провалений • заблокований

Передумова:		
Підготувати мок-користувачів із заданими рівнями досвіду	Мок-об'єкти створені	Пройдений
Кроки тесту:		
Викликати для однакового рівня: junior vs junior	Повернено 10	Пройдений
Викликати з різницею в один рівень: junior vs middle	Повернено 7	Пройдений
Викликати з різницею у два рівні: junior vs senior	Повернено 4	Пройдений
Викликати з великою різницею: junior vs leadExpert	Повернено 2 (мінімум)	Пройдений
Викликати з невідомим рівнем: godMode vs middle	Повернено 2 (fallback)	Пройдений
Перевірити симетричність: f(junior, senior) vs f(senior, junior)	Значення рівні	Пройдений
Післяумова:		
Перевірити коректність типу результату	Повернено число	Пройдений

Таблиця 3.7.

Тест випробування алгоритму оцінки збігу бачення

Назва:	Тест CalculateVisionAlignUseCase	
Функція/UseCase:	CalculateVisionAlignUseCase	
Дія:	Очікуваний результат:	Результат тесту:
Викликати функцію розрахунку збігу за полями aboutMe двох користувачів	Повернено значення у діапазоні [0; 10] на основі перетину значущих слів зі стоп-листом	• пройдений • провалений • заблокований
Передумова:		
Підготувати мок-користувачів із різними описами aboutMe	Мок-об'єкти створені	Пройдений
Кроки тесту:		
Викликати з порожніми описами	Повернено 0	Пройдений
Викликати з одним порожнім описом	Повернено 0	Пройдений
Викликати з ідентичними описами	Повернено 10.0	Пройдений
Викликати з описами зі стоп-словами (I am a / we are)	Значення > 0 та ≤ 10	Пройдений
Викликати з описами без спільних слів	Повернено 0	Пройдений
Перевірити симетричність функції	$f(a,b) === f(b,a)$	Пройдений
Післяумова:		
Перевірити, що результат не виходить за межі [0; 10]	Результат у заданому діапазоні	Пройдений

Тест випробування надсилання повідомлення у чат

Назва:	Тест SendMessageUseCase	
Функція/UseCase:	SendMessageUseCase + UpdateChatInfoUseCase	
Дія:	Очікуваний результат:	Результат тесту:
Користувач набирає текст повідомлення та натискає кнопку «Надіслати»	Повідомлення збережено у Firestore, метадані чату (lastMessage, lastMessageSent) оновлено	• пройдений • провалений • заблокований
Передумова:		
Авторизуватися в застосунку	Користувач авторизований; uid = 'user1'	Пройдений
Відкрити існуючий чат із співрозмовником	Відкрито екран чату, chatId визначено	Пройдений
Кроки тесту:		
Ввести текст «Hello» у поле повідомлення	Текст відображено у полі вводу	Пройдений
Натиснути кнопку «Надіслати»	Викликано chatMessageRepo.create з message='Hello' та sentUid='user1'	Пройдений
Перевірити отримання поточного чату	Викликано getChatByIdUseCase(chatId)	Пройдений
Перевірити оновлення метаданих чату	Викликано updateChatInfoUseCase з lastMessage='Hello'	Пройдений
Післяумова:		
Перевірити, що повідомлення відображається у стрічці чату	Повідомлення «Hello» у нижній частині списку	Пройдений

Перевірити, що у списку чатів оновився прев'ю	У картці чату відображено текст останнього повідомлення	Пройдений
---	---	-----------

Таблиця 3.9.

Тест випробування інфраструктурного шару викликів Firebase Functions

Назва:	Тест FirebaseFunctionsService та обробки помилок	
Функція/UseCase:	FirebaseFunctionsService	
Дія:	Очікуваний результат:	Результат тесту:
Здійснити виклик cloud-функції з різними сценаріями: успіх, таймаут, відмова доступу, відсутність функції	Кожен сценарій повертає коректний результат або кидає відповідний доменний виняток	• пройдений • провалений • заблокований
Передумова:		
Підготувати MockFirebaseFunctions та MockHttpsCallable	Мок-об'єкти ініціалізовано	Пройдений
Кроки тесту:		
Викликати function 'getUserData' з даними	Повернено очікувану відповідь сервера	Пройдений
Викликати function без даних	Повернено {timestamp: 1234567890}	Пройдений
Викликати з користувацьким таймаутом 60 с	Опції callable містять timeout = 60 с	Пройдений
Симулювати код 'not-found'	Кинута FunctionNotFoundException	Пройдений
Симулювати код 'deadline-exceeded'	Кинута FunctionDeadlineExceededException	Пройдений

Симулювати 'permission-denied'	Кинуто FunctionPermissionDeniedException	Пройдений
Симулювати 'unauthenticated'	Кинуто FunctionUnauthenticatedException	Пройдений
Симулювати 'internal'	Кинуто FunctionInternalException	Пройдений
Симулювати невідому помилку Exception	Кинуто FirebaseFunctionsException(code: 'unknown-error')	Пройдений
Післяумова:		
Перевірити коректну роботу сервісу після помилки	Наступні виклики виконуються нормально	Пройдений

Таблиця 3.10.

Тест випробування генерації ключових слів для пошуку чатів

Назва:	Тест ChatKeywordGeneratorService	
Функція/UseCase:	ChatKeywordGeneratorService	
Дія:	Очікуваний результат:	Результат тесту:
Викликати сервіс генерації ключових слів за іменем співрозмовника	Повернено масив слів у нижньому регістрі: повна фраза + окремі слова	• пройдений • провалений • заблокований
Передумова:		
Створити інстанс ChatKeywordGeneratorService	Сервіс ініціалізовано	Пройдений
Кроки тесту:		
Викликати з null	Повернено порожній масив []	Пройдений
Викликати з порожнім рядком "	Повернено порожній масив []	Пройдений

Викликати з 'Alex Doe'	Масив містить 'alex doe', 'alex', 'doe'	Пройдений
Післяумова:		
Перевірити, що повертається масив у нижньому регістрі	Усі елементи у lowercase	Пройдений

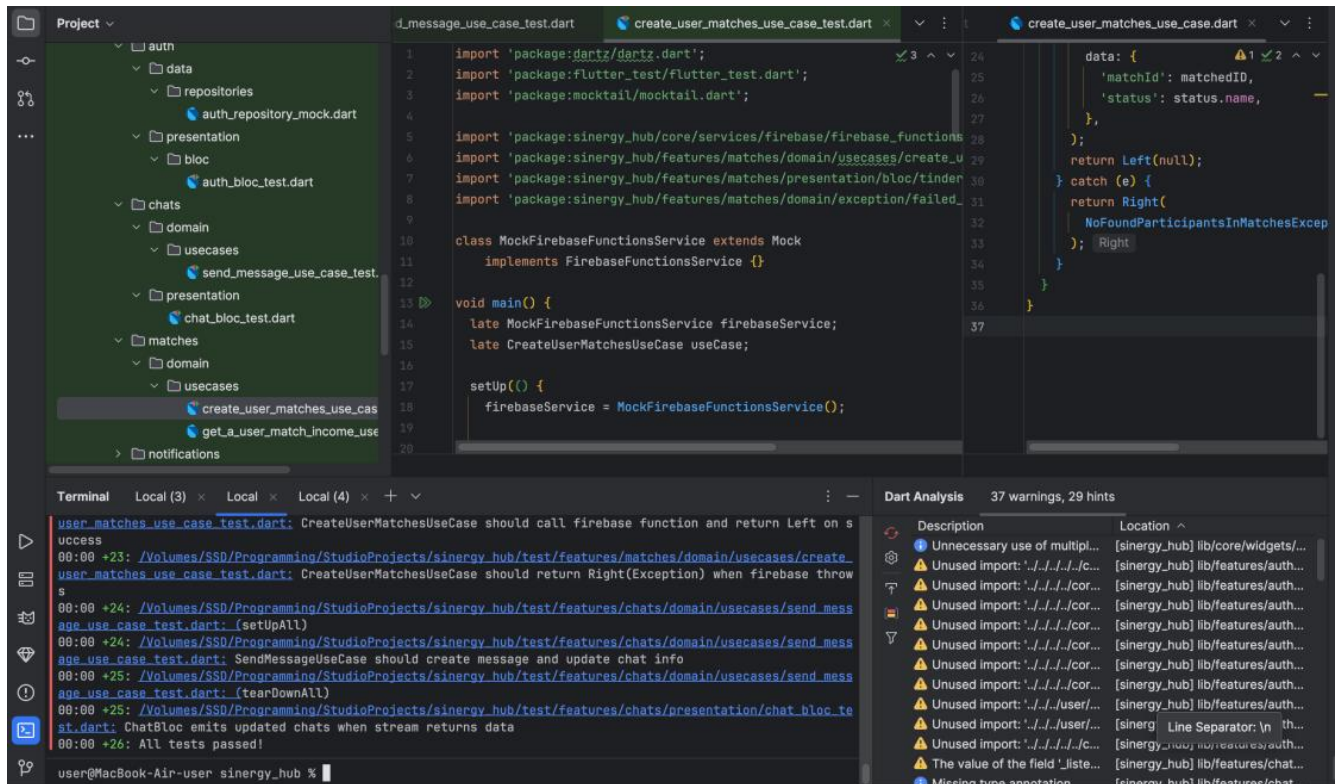


Рис. 3.1. Результати виконання тестів

Міністерство освіти і науки України
Державний заклад «Луганський національний університет
імені Тараса Шевченка»

Факультет

Факультет технологій та інформаційних систем
(повна назва)

Кафедра

Інформаційних технологій та систем
(повна назва)

КЕРІВНИЦТВО КОРИСТУВАЧА

на виконання програмної розробки (ПР):

"Мобільний додаток для формування стартап-команди"

ІТС. ІПЗ4.0126-04-КК

ПОГОДЖЕНО

Керівник кваліфікаційної роботи

Володимир ДОНЧЕНКО

“ ” 2026р.

ВИКОНАВЕЦЬ

Студент групи 4 ІПЗ МС

Олександр НАУМЕНКО

“ ” 2026р.

Лубни 2026

ЗМІСТ

1. ОПИС СТРУКТУРИ	3
2. ОПИС ЗМІСТУ ЕЛЕМЕНТІВ.....	5
2.1. Стартовий екран та вхід	5
2.2. Багатоетапний онбординг	6
2.3. Головний екран (свайп-карусель).....	8
2.4. Модальне вікно фільтрів	9
2.5. Список чатів.....	10
2.7. Екран чату	11
2.8. Профіль користувача	12
2.9. Drawer головного меню	13

1. ОПИС СТРУКТУРИ

Мобільний застосунок «Synergy Hub» — це кросплатформений мобільний продукт, призначений для пошуку партнерів та формування стартап-команд. Застосунок розроблено мовою програмування Dart з використанням фреймворку Flutter, що забезпечує підтримку платформ Android та iOS з єдиної кодової бази. Серверна частина реалізована на основі Firebase Cloud Functions (TypeScript), а зберігання даних здійснюється у Cloud Firestore.

Архітектура клієнтської частини побудована за принципами Clean Architecture з розподілом на три рівні: Presentation, Domain та Data. Для керування станом використовується патерн BLoC. Автентифікація користувачів виконується через Firebase Authentication з підтримкою Google OAuth

Застосунок має змогу:

- реєструвати користувача та авторизувати його через Google OAuth ;
- проводити багатоетапний онбординг із заповненням спеціалізації, рівня досвіду, навичок та бажаної ролі у команді;
- показувати інші профілі у форматі свайп-карток із фотографією, спеціалізацією, досвідом та переліком навичок;
- фіксувати рішення користувача (лайк свайпом праворуч або дизлайк свайпом ліворуч) та зберігати історію взаємодій у Firestore;
- автоматично створювати чат між учасниками після взаємного лайку (двостороннього матчу);
- обмінюватися текстовими повідомленнями у реальному часі через стрім Firestore;
- надсилати push-сповіщення про нові матчі та повідомлення через Firebase Cloud Messaging;
- редагувати власний профіль ;
- фільтрувати користувачів за спеціалізацією та рівнем досвіду.

Застосунок зберігає всі дані користувача (профіль, історію взаємодій, чати) на серверній стороні, що дозволяє безперешкодно продовжити роботу з будь-якого пристрою після повторної автентифікації. Для коректної роботи потрібне постійне підключення до мережі Інтернет.

2. ОПИС ЗМІСТУ ЕЛЕМЕНТІВ

При запуску застосунку користувач послідовно проходить кілька екранів — від стартового до головного. Нижче наведено опис кожного елемента інтерфейсу за порядком використання.

2.1. Стартовий екран та вхід

При першому запуску застосунку користувач бачить стартовий екран із логотипом «Synergy Hub» та коротким описом. Унизу екрана розташовано дві кнопки автентифікації: «Увійти через Google». Після натискання обраної кнопки відкривається системний діалог вибору облікового запису.

У разі успішної автентифікації та якщо у користувача вже заповнено профіль, відбувається перехід на головний екран. У разі першого входу — на екран онбордингу.

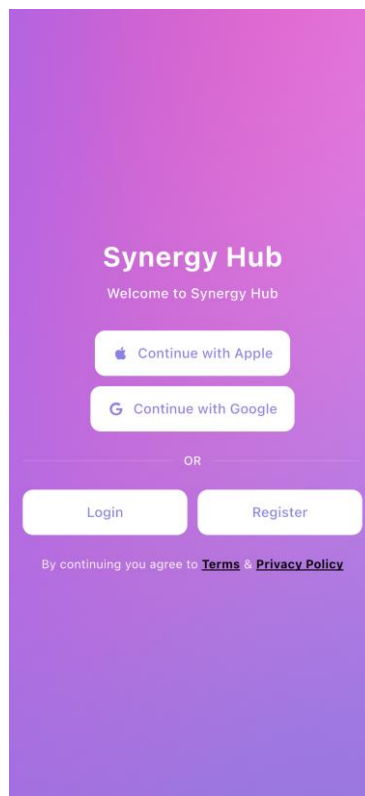


Рис. 1. Стартовий екран та екран входу

2.2. Багатоетапний онбординг

Онбординг складається з п'яти послідовних кроків, на кожному з яких користувач вводить дані для свого профілю. Угорі кожного кроку відображається індикатор прогресу, а внизу — кнопки «Назад» та «Далі».

- Крок 1. Спеціалізація — вибір однієї або кількох спеціалізацій зі списку (Mobile Developer, Web Developer, Designer, Product Manager тощо).
- Крок 2. Рівень досвіду — вибір одного з варіантів: Junior, Middle, Senior, Lead/Expert.
- Крок 3. Навички — додавання ключових технологій або інструментів (Flutter, Dart, Firebase, React, Figma і т. д.).
- Крок 4. Роль у команді — вибір бажаної ролі у майбутньому стартапі (Co-founder, Developer, Designer, Marketer).
- Крок 5. «Про себе» — текстове поле з коротким описом цілей, цінностей та бачення проєкту.

Після завершення останнього кроку профіль зберігається у Firestore, прапор completion = true, і користувач переходить на головний екран.

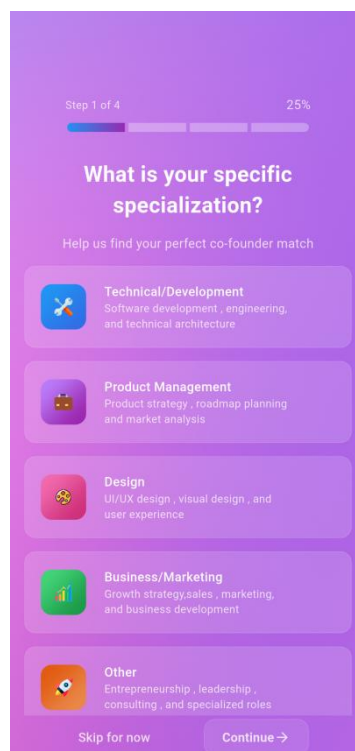


Рис. 2. Крок 1 вибір загальної спеціальності

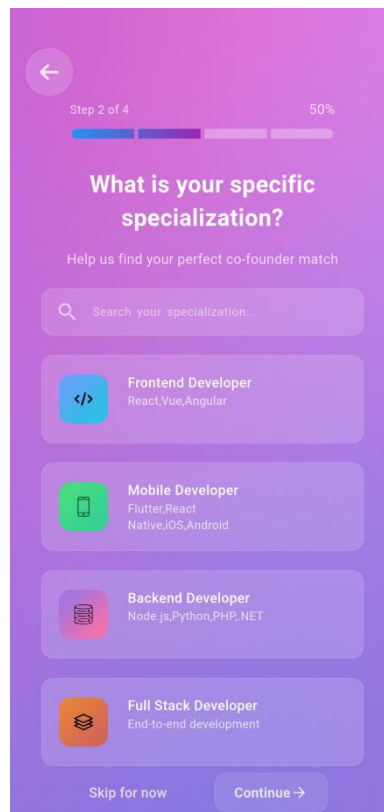


Рис. 3. Крок 2 - вибір конкретної спеціальності

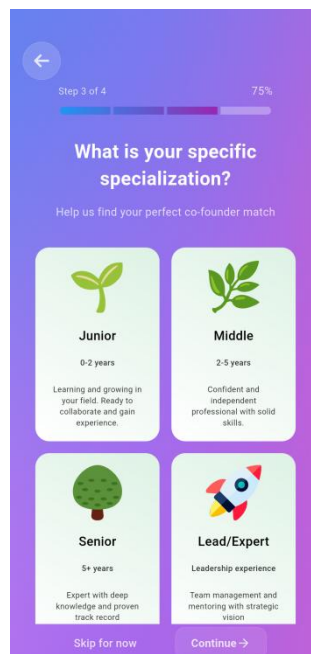


Рис. 4. Крок 3 - вибір досвіду

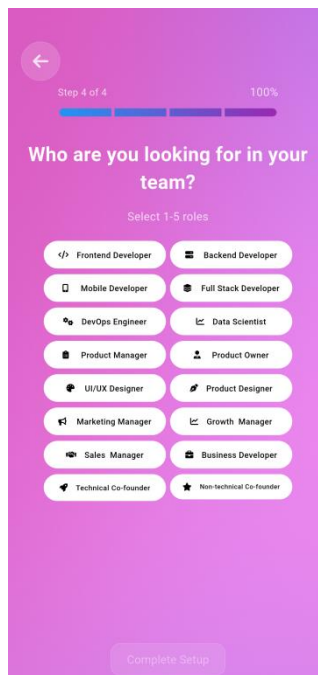


Рис. 5. Крок 4 - вибір спеціальностей з якими планується сформувати команду

2.3. Головний екран (свайп-карусель)

Головний екран являє собою каруселю карток з профілями інших користувачів. Кожна картка містить фотографію, ім'я, спеціалізацію, рівень досвіду та перелік ключових навичок співрозмовника. Користувач взаємодіє з картокою за допомогою жестів:

- свайп праворуч або натискання кнопки «♥» — лайк (виявлений інтерес);
- свайп ліворуч або натискання кнопки «X» — дизлайк (пропустити профіль);
- натискання на картку — відкрити повну версію профілю з усією інформацією.

У верхньому правому куті розміщено іконку фільтрів, у верхньому лівому — іконку профілю поточного користувача. Внизу екрана розташовано панель навігації з вкладками: Головна, Матчі, Чати, Профіль.

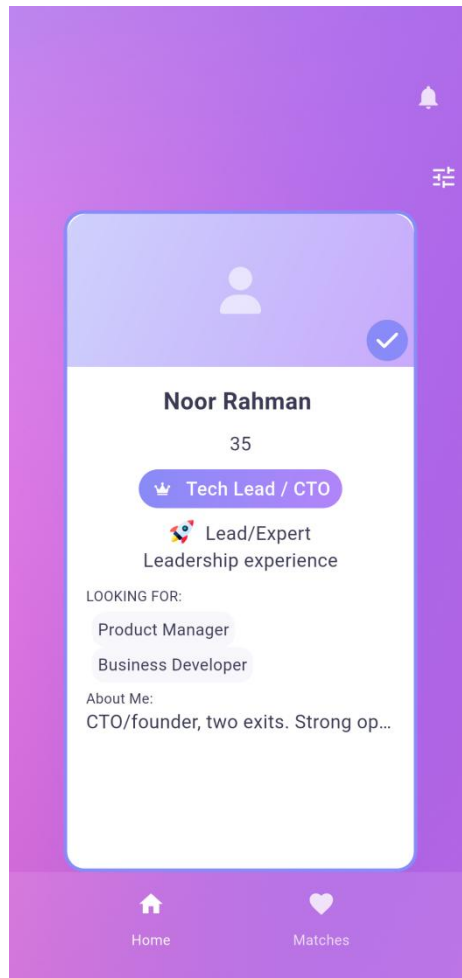


Рис. 6. Головний екран зі свайп-каруселлю профілів

2.4. Модальне вікно фільтрів

Натискання іконки фільтрів відкриває модальне вікно, на якому користувач може налаштувати критерії пошуку співрозмовників. Доступні поля:

- **спеціалізація** — множинний вибір зі списку спеціалізацій;
- **рівень досвіду** — діапазон від Junior до Lead/Expert;
- **навички** — множинний вибір ключових технологій або інструментів.

Внизу екрана розташована кнопка «Застосувати» (зберігає фільтри й закриває вікно із оновленою каруселлю).

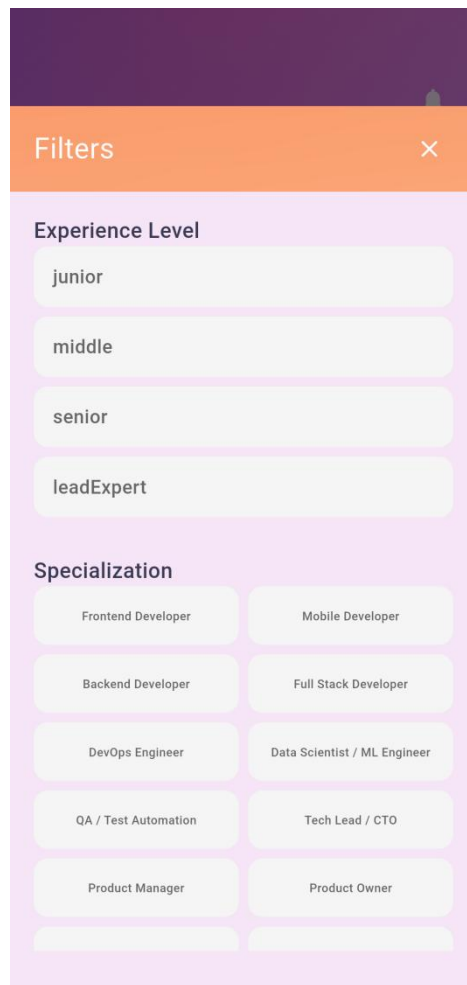


Рис. 7. Екран фільтрів профілів

2.5. Список чатів

Екран чатів містить усі активні діалоги поточного користувача. Угорі екрана розташовано поле пошуку за іменем співрозмовника. Кожен елемент списку містить:

- аватар співрозмовника;
- ім'я співрозмовника;
- текст останнього повідомлення;
- час відправлення останнього повідомлення;

Список упорядкований за часом останнього повідомлення (від нового до старого). Натискання на елемент відкриває екран чату.

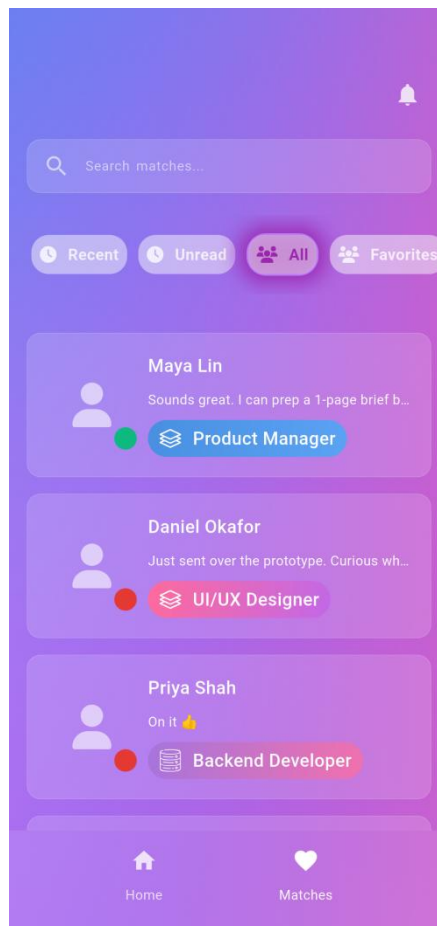


Рис. 8. Список чатів

2.7. Екран чату

Екран чату містить стрічку повідомлень між поточним користувачем та обраним співрозмовником. Власні повідомлення вирівняні праворуч, повідомлення співрозмовника — ліворуч. Внизу екрана розташовано поле вводу тексту та кнопку надсилання повідомлення (іконка «відправити»).

Біля кожного власного повідомлення відображається індикатор статусу доставки: годинник (надсилається), одна галочка (доставлено), дві галочки (прочитано). У верхній частині екрана показується ім'я та аватар співрозмовника, а також індикатор «набирає повідомлення», коли співрозмовник друкує.

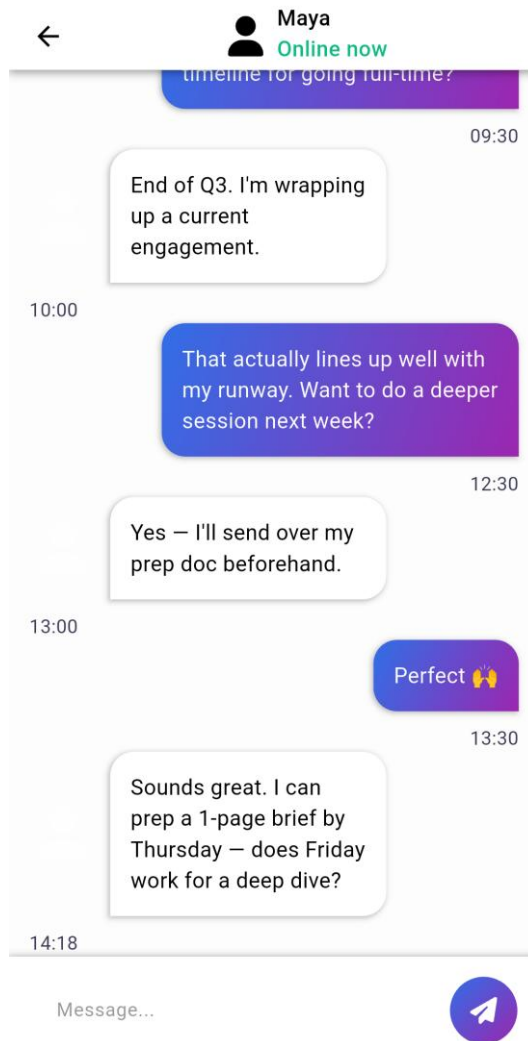


Рис. 9. Екран чату

2.8. Профіль користувача

Екран профілю поточного користувача містить аватар, ім'я, спеціалізацію, рівень досвіду, перелік навичок та поле «Про себе», заповнені на етапі онбордингу. Угорі праворуч розташовано іконку «Редагувати», що дозволяє перейти на форму редагування даних та зберегти зміни у Firestore.

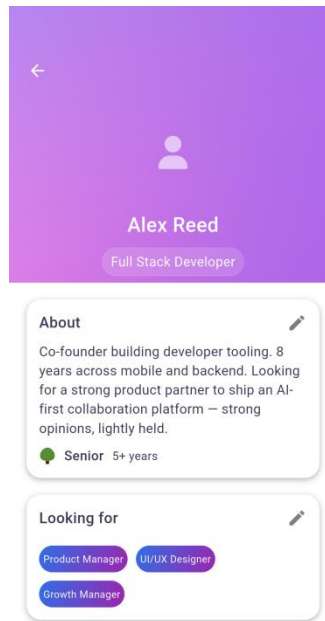


Рис. 10. Екран профілю користувача

2.9. Drawer головного меню

Drawer-меню має кнопки «Сповіщення» , «Вийти» , «Аккаунт» і «Політика приватності» . Натискання «Вийти» завершує сесію Firebase Authentication та повертає користувача на стартовий екран. Кнопка «Сповіщення» виконує ту саму дію що і на сторінці головного екрану - відкриває сторінку повідомлень , «Аккаунт» - відкриває екран профілю користувача і «Політика приватності» показує політику використання додатка .

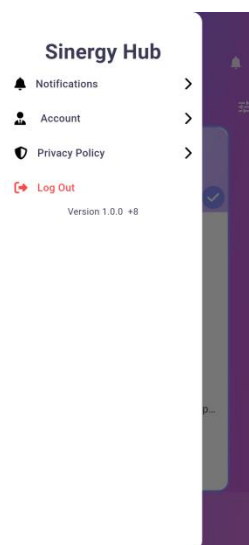


Рис. 11. Екран профілю користувача