Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження способів оптимізації безсерверних Flutter застосунків
(тема)

Виконала:
здобувачка 2 року навчання
групи ІПЗм-23-1
Юлія КОБА
(Власне ім'я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного забезпечення.
(код і повна назва спеціальності)
Тип програми освітньо-наукова

Керівник доц. Олексій НАЗАРОВ
(посада, Власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту
Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук

Кафедра програмної інженерії

Рівень вищої освіти другий (магістерський)

Спеціальність 121 – Інженерія програмного забезпечення

Тип програми освітньо-наукова

Освітня програма Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентці Кобі Юлії Юріївні
(Прізвище, ім'я, по батькові)

1. Тема роботи: «Дослідження способів оптимізації безсерверних Flutter застосунків»

Затверджена наказом по університету №290 Ст від 15.04.2025

2. Термін подання студентом роботи до екзаменаційної комісії 12.06.2025

3. Вихідні дані до роботи Хмарний сервіс Supabase, вимоги до розробки бази даних та застосунку, мова програмування Dart (фреймворк Flutter), середовище розробки Visual Studio Code, Microsoft Excel для аналізу результатів проведеного дослідження.

4. Перелік питань, які потрібно опрацювати в роботі:

Аналіз предметної галузі, опис та аналіз наукової літератури, проектування структури бази даних, розробка архітектури застосунку, створення бази даних у хмарі, розробка програмного забезпечення для проведення дослідження, проведення експериментального дослідження обраних методів оптимізації рендерингу.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	19.05.2025	*виконано*
2	Аналіз предметної галузі і постановка задачі	20.05.2025	*виконано*
3	Огляд й аналіз літературних, наукових джерел	21.05.2025	*виконано*
4	Вирішення задачі багатокритеріального вибору для обрання безсервеного рішення	22.05.2025	*виконано*
5	Проектування структури бази даних	23.05.2025	*виконано*
6	Розробка архітектури застосунку	26.05.2025	*виконано*
7	Створення бази даних у хмарі	27.05.2025	*виконано*
8	Розробка програмного забезпечення для проведення дослідження	28.05.2025	*виконано*
9	Проведення експериментального дослідження	29.05.2025	*виконано*
10	Підготовка до апробації результатів дослідження. Публікація матеріалів	30.05.2025	*виконано*
11	Програмна реалізація	01.06.2025	*виконано*
12	Підготовка пояснювальної записки	02.06.2025	*виконано*
13	Підготовка презентації та доповіді	03.06.2025	*виконано*
14	Перевірка на плагіат	04.06.2025	*виконано*
15	Нормоконтроль	05.06.2025	*виконано*
16	Рецензування	05.06.2025	*виконано*
17	Попередній захист	06.06.2025	*виконано*
18	Занесення диплома в електронний архів	09.06.2025	*виконано*
19	Допуск до захисту у зав. кафедри	10.06.2025	*виконано*

Дата видачі завдання 19.05.2025р.

Студентка _____ Юлія КОБА
(підпис)

Керівник роботи _____ доц. Олексій НАЗАРОВ
(підпис) (посада, Власне ім’я, прізвище)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить: 138 с., 34 рис., 14 табл., 30 джерел.

АЛГОРИТМИ ПРОГНОЗУВАННЯ, БЕЗСЕРВЕРНІ ЗАСТОСУНКИ, КРОСПЛАТФОРМНІСТЬ, ОПТИМІЗАЦІЯ, FLUTTER, SUPABASE, VISUAL STUDIO CODE.

Об’єктом дослідження є Flutter застосунки і способи їх оптимізації.

Метою роботи є проведення дослідження продуктивності методів програмної реалізації безсерверних Flutter застосунків, а також способи оптимізації рендерингу.

Методами розробки та проектування є аналіз проблемної області дослідження, вибір хмарного API для проведення дослідження шляхом вирішення багатокритеріальної задачі прийняття рішень, побудова рівнянь регресії для порівняння методів оптимізації рендерингу UI потоку.

У результаті кваліфікаційної роботи було розроблено програму з використанням хмарного сервісу Supabase з використанням технік оптимізації рендерингу.

PREDICTION ALGORITHMS, SERVERLESS APPLICATIONS, CROSSPLATFORM, OPTIMIZATION, FLUTTER, SUPABASE, VISUAL STUDIO CODE.

The object of research is Flutter applications and methods of their optimization.

The purpose of the work is to conduct research on the productivity of software implementation methods of serverless Flutter applications, as well as ways to optimize rendering.

The development and design methods are the analysis of the problem area of the study, the selection of a cloud API for conducting the study by solving a multi-criteria

decision-making problem, the construction of regression equations for the comparison of optimization methods for UI flow rendering.

As a result of qualified work, a program was split up using the poor Supabase service and using different techniques for optimizing rendering.

Завідувачу кафедри
ПІ_____
(скорочена назва кафедри)
проф. Кирилу СМЕЛЯКОВУ
(вчене звання, власне ім'я, прізвище)

ЗАЯВА

щодо самостійності виконання кваліфікаційної роботи та можливості її публікації (та/або публікації анотації кваліфікаційної роботи) в електронному архіві відкритого доступу ElAr KhNURE

Я,

_____Коба Юлія Юріївна_____
(прізвище, ім'я, по батькові)

здобувачка вищої освіти на другому (магістерському) рівні вищої освіти академічної групи _____ІПЗм-23-1_____
кафедра _____програмної інженерії_____
(повна назва кафедри)

заявляю: моя кваліфікаційна робота на тему

Дослідження способів оптимізації безсерверних Flutter застосунків_____,
(назва роботи)

що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в репозиторії "ElArKhNURE". Погоджуюся з авторським договором, відповідно до Положення про репозиторій ХНУРЕ "ElArKhNURE". Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений (а) з вимогами академічної доброчесності, згідно з якими виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

Дата Підпис

ЗМІСТ

ВСТУП

У сучасних умовах стрімкого розвитку мобільних технологій та зростаючих вимог до продуктивності застосунків, питання їх оптимізації набуває особливого значення. Ефективна оптимізація дозволяє підвищити якість користувацького досвіду, що є ключовим фактором успіху мобільного застосунку. Flutter, як один із найпопулярніших кросплатформних фреймворків, дозволяє розробляти застосунки для різних операційних систем з мінімальними витратами ресурсів і часу. Проте, для забезпечення найкращого користувацького досвіду, важливо не лише швидко створити застосунок, але й оптимізувати його роботу. Це особливо актуально в умовах сучасної конкуренції на ринку мобільних технологій.

Кваліфікаційна робота присвячена дослідженню безсерверних Flutter-застосунків та способам їх оптимізації. Обрана тема відображає актуальність використання передових технологій для вдосконалення процесу розробки програмного забезпечення.

По-перше, у роботі розглядаються різні методи програмної реалізації безсерверних архітектур за допомогою хмарних сервісів. Безсерверна архітектура, або "serverless", дозволяє розробникам зосередитися на функціональності застосунку, не турбуючись про управління серверами, що сприяє спрощенню розробки і масштабування. Цей підхід також знижує витрати на інфраструктуру та спрощує обслуговування проєкту.

Для цього було обрано хмарну платформу Supabase. Ця платформа забезпечує широкий спектр функцій, таких як інтеграція баз даних, авторизація та функції реального часу. По-друге, досліджуються методи оптимізації рендерингу інтерфейсу користувача (UI) у Flutter-застосунках. Рендеринг відіграє ключову роль у забезпеченні плавності роботи застосунку, оскільки саме він відповідає за відображення елементів інтерфейсу на екрані. Оптимізація рендерингу дозволяє не лише покращити продуктивність, але й зменшити енергоспоживання пристрою.

Оптимізація цього процесу дозволяє значно зменшити затримки та підвищити загальну продуктивність застосунку. У рамках дослідження

аналізуються різні підходи до покращення продуктивності UI-потоку, включаючи використання технік, що зменшують навантаження на обчислювальні ресурси пристрою. Такі підходи дозволяють створювати застосунки, які забезпечують плавний та інтуїтивно зрозумілий інтерфейс. По-третє, у процесі роботи було застосовано багатокритеріальний підхід для вибору оптимального хмарного сервісу.

Вибір відповідної платформи має велике значення для продуктивності застосунку, оскільки різні сервіси пропонують різні можливості, від масштабованості до інтеграції з інструментами для оптимізації. Це дозволяє знайти рішення, що найбільш відповідає потребам конкретного проєкту. У результаті виконаного дослідження було розроблено оптимізований Flutter-застосунок з використанням хмарної платформи Supabase, що демонструє підвищену продуктивність рендерингу користувацького інтерфейсу.

Систематизовано методи оптимізації UI-потоку та створено практичні рекомендації щодо їх застосування. Ці рекомендації можуть бути використані розробниками як базис для покращення їхніх проєктів. Результати роботи можуть бути використані розробниками мобільних застосунків для підвищення продуктивності своїх проєктів та оптимізації процесу розробки. Таким чином, дослідження пропонує нові можливості для вдосконалення мобільного програмного забезпечення.

Загалом, робота робить значний внесок у розуміння того, як можна оптимізувати рендеринг у Flutter-застосунках та підвищити їх продуктивність через ефективне використання хмарних платформ. Це забезпечує конкурентну перевагу для розробників, що прагнуть створювати сучасні та ефективні мобільні рішення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі дослідження

Flutter є однією з найбільш популярних технологій для розробки програмного забезпечення в сучасному світі. Цей кросплатформний фреймворк, розроблений компанією Google, надає розробникам широкий набір інструментів для створення різноманітних додатків – від мобільних і вебзастосунків до настільних програм і вбудованих систем. Завдяки високій продуктивності та можливості створювати ефективні користувацькі інтерфейси з одним кодовим базисом для кількох платформ, Flutter стає вибором як для невеликих команд, так і для великих компаній.

Актуальність дослідження для платформи Flutter полягає в тому, що ця технологія швидко еволюціонує та постійно отримує нові функції і вдосконалення. Оскільки Flutter активно розвивається, розробники повинні постійно оновлювати свої знання та навички, щоб залишатися на передовій сучасних тенденцій і повністю використовувати можливості фреймворку. Зокрема, це стосується нових інструментів для оптимізації продуктивності, покращення рендерингу інтерфейсів та інтеграції з різними хмарними платформами й сервісами.

Крім того, Flutter є фреймворком з відкритим вихідним кодом, що дає змогу розробникам по всьому світу співпрацювати та робити внесок у його вдосконалення. Це сприяє формуванню сильної спільноти, яка активно бере участь у розвитку та розширенні можливостей Flutter. Дослідження в цій сфері може допомогти зміцнити співпрацю між розробниками та поширити знання, що, у свою чергу, сприятиме подальшому розвитку фреймворку та появі нових інноваційних рішень на його базі.

Продуктивність застосунку має йти в ногу з його коректною та безперебійною розробкою. Повільні або не реагуючі застосунки викликають невдоволення користувачів, негативні відгуки та їх відтік. Тому для бізнесу важливо надавати пріоритет оптимізації продуктивності. Швидкий рендеринг є ключовим аспектом продуктивності застосунків з кількох причин. По-перше, він

безпосередньо впливає на враження користувачів: миттєве відображення елементів інтерфейсу створює відчуття гладкості та професійності, що позитивно сприймається користувачами.

По-друге, швидкий рендеринг допомагає зменшити навантаження на пристрій, що особливо важливо для мобільних телефонів з обмеженими ресурсами. Коли застосунок швидко реагує на дії користувача, це знижує ризик фризів і затримок, що може викликати розчарування і призвести до відмови від використання.

По-третє, оптимізація рендерингу дозволяє зекономити енергію акумулятора, що є важливим для тривалого використання застосунку. В результаті, користувачі отримують не лише швидкий, а й енергоефективний продукт, що позитивно впливає на їхню задоволеність і лояльність до бренду.

Безсерверні (serverless) рішення для Flutter стають дедалі популярнішими через їхні численні переваги, які значно спрощують розробку та масштабування мобільних застосунків. По-перше, безсерверна архітектура дозволяє розробникам зосередитися на написанні коду та реалізації функціональності, не турбуючись про управління серверами та інфраструктурою. Це знижує витрати на розробку та спрощує процес запуску нових функцій.

По-друге, безсерверні рішення забезпечують автоматичне масштабування в залежності від навантаження, що дозволяє застосункам ефективно справлятися з піковими навантаженнями без необхідності ручного налаштування серверів. Це особливо важливо для популярних мобільних застосунків, які можуть зазнавати раптових стрибків у кількості користувачів.

По-третє, безсерверні платформи зазвичай мають вбудовані функції безпеки, що дозволяє зменшити ризики, пов’язані з управлінням серверною інфраструктурою. Розробники можуть скористатися такими сервісами, як аутентифікація, зберігання даних і функції обробки подій, що підвищує надійність та безпечність застосунків.

Завдяки цим перевагам, використання безсерверних рішень для Flutter може суттєво підвищити продуктивність розробки, знизити витрати та забезпечити

кращу стабільність і масштабованість кросплатформних застосунків [1]. Це робить їх важливими для сучасних розробників, які прагнуть створити конкурентоспроможні та інноваційні продукти.

1.2 Виявлення проблем та актуалізація рішень

У контексті Flutter-застосунків продуктивність переважно зосереджується на двох ключових показниках [2]:

- швидкість рендерингу – швидкість, з якою Flutter може згенерувати пікселі, що складають інтерфейс застосунку на екрані. Ідеально, щоб Flutter міг рендерити кожен кадр приблизно за 16 мілісекунд (мс), щоб досягти плавного відтворення на 60 кадрах на секунду (FPS). Це забезпечує безшовну та чуйний взаємодію для користувача;
- частота кадрів (FPS). FPS вказує на кількість разів за секунду, коли інтерфейс додатку оновлюється та перерисовується на екрані. Вища частота кадрів призводить до більш плавного та рідкого користувацького досвіду. У свою чергу, низька частота кадрів може викликати ривки, затримки та відчуття повільності.

Забезпечення оптимальної швидкості рендерингу та високої частоти кадрів є критично важливим для досягнення високої продуктивності та задоволення користувачів у Flutter-застосунках.

На продуктивність Flutter-застосунку можуть впливати кілька факторів [3]. Ось основні з них:

- складність дерева віджетів: Flutter будує інтерфейс застосунку за допомогою ієрархії віджетів. Складне дерево віджетів з багатьма вкладеними елементами може потребувати більше часу на рендеринг, що вплине на продуктивність;
- частота переробки віджетів: Flutter переробляє все піддерево віджетів щоразу, коли відбувається зміна в його стані, навіть якщо зміна стосується лише невеликої частини інтерфейсу. Це може стати вузьким

місцем у продуктивності для часто оновлюваних віджетів або тих, що глибоко вкладені в дерево віджетів;

- стратегія управління станом: те, управляється стан застосунку, може суттєво вплинути на продуктивність. Неправильні практики управління станом можуть викликати непотрібні переробки віджетів, що призводить до уповільнення;
- складність інтерфейсу: візуально складні інтерфейси з багатими анімаціями, важкими макетами або великими зображеннями можуть вимагати більше обчислювальних ресурсів для рендерингу, що потенційно вплине на продуктивність;
- можливості пристрою: продуктивність застосунку також буде залежати від пристрою користувача. Пристрої з низькою обчислювальною потужністю, обмеженою пам'яттю або повільними мережевими з'єднаннями зазнають уповільнення в роботі застосунку.

Зважаючи на ці фактори, важливо ретельно оптимізувати Flutter-застосунок для забезпечення найкращої продуктивності та досвіду користувачів. Для проведення дослідження варто виокремити наступні способи оптимізації:

- уникнення непотрібної переробки віджетів;
- використання константних конструкторів;
- мінімізація використання Stateful віджетів;
- мінімізація обсягу build-методів;
- мінімізація використання helper-методів;
- рендеринг тільки тих віджетів, які видно на поточному екрані;
- мінімізація використання opacity у віджетах;
- ефективне використання асинхронних функцій та багатопоточності;
- оптимізація network-запитів;
- кешування даних.

Зростання популярності безсерверних технологій для Flutter-додатків супроводжується певними викликами, які потребують оптимізації та ретельного

вибору інструментів [4]. Безсерверні рішення, такі як Firebase, AWS Amplify, Azure Functions, DynamoDB та Supabase, надають потужні можливості для зберігання даних, обробки запитів і управління функціоналом. Однак їх застосування потребує врахування ключових аспектів, серед яких особливо важливими є оптимізація витрат, гнучкість масштабування та спрощене управління інфраструктурою [5]. Розглянемо основні безсерверні платформи та їх особливості.

Firebase від Google є одним з найпопулярніших безсерверних рішень для Flutter, характеристики якого наведені у таблиці 1.1.

Таблиця 1.1 – Можливості та проблеми Firebase

Можливості	Проблеми
Firebase Cloud Functions: дозволяє виконувати серверний код у відповідь на події	Висока вартість при значному масштабуванні
Cloud Firestore: масштабована NoSQL база даних	Обмежена гнучкість порівняно з традиційними серверними рішеннями
Firebase Authentication: готове рішення для автентифікації	Залежність від екосистеми Google
Firebase Storage: сховище для файлів	
Firebase Analytics: аналітика користувацької поведінки	

Amazon Web Services пропонує комплексне рішення для Flutter через AWS Amplify (див. табл. 1.2).

Таблиця 1.2 – Можливості та проблеми AWS Amplify

Можливості	Проблеми
AWS Lambda: виконання серверних	Потреба в глибокому розумінні AWS

Кінець таблиці 1.2

Можливості	Проблеми
функцій	екосистеми
API Gateway: управління API	Крута крива навчання
Cognito: автентифікація та авторизація	Складна конфігурація та налаштування
S3: об'єктне сховище	

Microsoft Azure пропонує безсерверні обчислення через Azure Functions (див. табл. 1.3).

Таблиця 1.3 – Можливості та проблеми Microsoft Azure Functions

Можливості	Проблеми
Підтримка різних мов програмування	Обмежена документація для Flutter
Інтеграція з іншими сервісами Azure	Менша спільнота порівняно з Firebase та AWS
Автоматичне масштабування	Складність налаштування локального середовища розробки
Pay-per-use модель оплати	

Amazon DynamoDB пропонує потужне рішення для зберігання даних (див. табл. 1.4).

Таблиця 1.4 – Можливості та проблеми Amazon DynamoDB

Можливості	Проблеми
Повністю керована NoSQL база даних	Складна модель ціноутворення.
Автоматичне масштабування	Обмеження у запитах та індексації.
Висока доступність	Потреба в ретельному плануванні структури даних.
Інтеграція з AWS Lambda	

Supabase позиціонується як альтернатива Firebase з відкритим кодом (див. табл. 1.5).

Таблиця 1.5 – Можливості та проблеми Amazon DynamoDB

Можливості	Проблеми
PostgreSQL база даних	Молода платформа з можливими проблемами стабільності
Автентифікація	Обмежена функціональність порівняно з Firebase
Realtime підписки	Менша екосистема готових рішень
Storage для файлів	
Edge Functions	

Для об'єктивного вибору оптимального безсерверного рішення доцільно використовувати методи багатокритеріального аналізу. Цей підхід дозволяє формалізувати процес прийняття рішень шляхом врахування множини критеріїв, таких як вартість, продуктивність, надійність, функціональність та простота розробки. Застосування методів TOPSIS (Technique for Order Preference by Similarity to Ideal Solution) або методу зваженої суми (Weighted Sum Method) дозволяє отримати кількісну оцінку кожного рішення та обрати найбільш відповідне для конкретного проекту. TOPSIS базується на концепції, що обране рішення повинно мати найкоротшу геометричну відстань від позитивного ідеального рішення та найдовшу від негативного ідеального рішення. Метод зваженої суми, в свою чергу, дозволяє врахувати відносну важливість кожного критерію через систему вагових коефіцієнтів. Це особливо важливо при виборі безсерверного рішення, оскільки різні проекти можуть мати різні пріоритети щодо продуктивності, вартості чи простоти розробки.

Такий комплексний підхід дозволить максимально ефективно використовувати переваги безсерверних рішень, мінімізуючи їх недоліки та забезпечуючи гнучкість у виборі технологій для конкретних потреб проекту.

2 ОГЛЯД Й АНАЛІЗ ЛІТЕРАТУРНИХ, НАУКОВИХ ДЖЕРЕЛ

2.1 Критерії вибору джерел

Для забезпечення високої якості аналізу були використані лише перевірені й авторитетні джерела. До них відносяться:

- статті, опубліковані у провідних наукових журналах, таких як IEEE Access, ACM Transactions, Elsevier Journals, та інших виданнях, що мають високий імпакт-фактор;
- матеріали міжнародних наукових конференцій, зокрема ACM SIGPLAN, IEEE Cloud Computing Conference, Flutter Vikings, Google I/O тощо;
- рецензовані збірники статей, які містять результати сучасних досліджень з галузі серверлес-обчислень та технології Flutter.

Авторитетність джерела підтверджується репутацією видавництва, рівнем рецензування робіт, а також популярністю серед дослідників.

У зв’язку з швидким розвитком технологій, особлива увага була приділена роботам, опублікованим у період 2019–2024 років. Це дозволяє включити в огляд найбільш сучасні та релевантні дослідження, які відображають поточний стан і тренди в галузі. Для дотримання актуальності робіт були застосовані наступні підходи:

- у фокусі були новітні розробки, інноваційні підходи та результати практичного застосування серверлес-технологій у комбінації з Flutter;
- були виключені застарілі матеріали, які не враховують останні зміни в архітектурі серверлес-обчислень або оновлення у Flutter SDK.

Для формування неупередженого огляду використовувалися лише ті джерела, які:

- базуються на незалежних дослідженнях, а не на маркетингових або рекламних матеріалах;
- містять прозору методологію, яка детально описує процес експериментів, тестування або моделювання;

– підтримують свої висновки доказами, такими як порівняльні тести, аналітичні моделі або статистичні дані.

Віддавалась перевага публікаціям з чіткими обґрунтуваннями, що виключає ризик суб’єктивності висновків.

Лише ті роботи, які мають високий рівень достовірності, були включені до аналізу. Основні показники достовірності:

– наявність експериментальних результатів, які були підтверджені практичними впровадженнями в реальних проєктах;
– включення до досліджень чисельного моделювання або детального опису проведених тестувань;
– публікації, що отримали визнання в науковій спільноті або містять посилання на подібні незалежні дослідження.

Таким чином, відібрані джерела забезпечують повноту, актуальність та надійність огляду, а також слугують міцною базою для розробки рекомендацій або прийняття стратегічних рішень у зазначеній сфері.

2.2 Аналіз літератури

Усі проаналізовані літературні джерела були розподілені за темами, які були у них досліджені:

а) серверлес-обчислення: концепції та моделі:

1. Luo et al. [2] досліджують ефективне розміщення компонентів серверлес-додатків для підвищення гнучкості та продуктивності; основною методологією є використання моделей оптимізації з урахуванням динамічного завантаження серверів;

2. Raza et al. [3] застосували статистичне навчання для визначення оптимальної конфігурації серверлес-додатків; їхній підхід підвищує ефективність роботи в багатокомпонентних системах;

3. Lin & Khazaei [4] представили моделі продуктивності та витрат для серверлес-додатків, враховуючи різні сценарії використання хмарних платформ;

4. Eismann et al. [5] визначили ключові причини та стратегії впровадження серверлес-додатків, зокрема для зниження витрат та підвищення еластичності.

б) оптимізація та масштабування:

1. Rajan [6] фокусується на оптимізації оркестрації серверлес-функцій для складних робочих процесів, використовуючи динамічні алгоритми маршрутизації;

2. Ship et al. [7] пропонують рішення для одночасного автошкалу серверлес-додатків, що враховує багатофакторний аналіз продуктивності;

3. Mampage et al. [8] запропонували алгоритм глибокого підкріплювального навчання для масштабування серверлес-додатків з оптимізацією часу та витрат; результати показують значне зменшення затримок при високих навантаженнях;

в) архітектура та управління серверлес-додатками:

1. Winzinger & Wirtz [9] провели аналіз архітектурних моделей серверлес-додатків, зосередившись на структурній ефективності;

2. Kallas et al. [10] досліджували виконання мікросервісів у серверлес-середовищі, підкресливши важливість коректної синхронізації функцій;

3. Rausch et al. [11] представили методи оптимізованого планування контейнерів для серверлес-обчислень на периферійних пристроях, акцентуючи увагу на даних з високою інтенсивністю;

г) тестування та ефективність:

1. De Silva & Hewawasam [12] проаналізували вплив різних підходів до тестування серверлес-додатків на їхню продуктивність, включаючи інтеграційні та модульні тести;

2. Ali et al. [13] вивчили оптимізацію подачі запитів на серверлес-платформах, використовуючи методи машинного навчання для покращення точності та зменшення затримок;

д) розробка з використанням Flutter:

1. Nanavati et al. [14] зосереджуються на оптимізації продуктивності додатків Flutter, пропонуючи техніки фінального налаштування;

2. Szczepanik & Kedziora [15] описують підходи до управління станом і архітектури в кросплатформних додатках Flutter;

3. Boukhary & Colmenares [16] представили "чисту" архітектуру для Flutter, яка підвищує підтримуваність і модульність коду.

Нижче наведено таблицю повірняння методів дослідження (див. табл. 2.1).

Таблиця 2.1 – Порівняння методів

Автори	Цілі дослідження	Використані методи	Результати та висновки
Luo et al. (2024)	Оптимізація розміщення компонентів	Моделі оптимізації	Підвищення продуктивності серверлес-додатків
Raza et al. (2023)	Конфігурація серверлес-додатків	Статистичне навчання	Зменшення затримок у роботі додатків
Mampage et al. (2023)	Масштабування додатків	Глибоке підкріплювальне навчання	Ефективне використання ресурсів
Nanavati et al. (2024)	Оптимізація Flutter-додатків	Техніки фінального налаштування	Підвищення продуктивності та стабільності

2.3 Оцінка актуальності і новизни

Аналізовані джерела охоплюють ключові аспекти серверлес-обчислень і розробки кросплатформних додатків, які відповідають сучасним викликам і тенденціям розвитку технологій. Вони забезпечують цінну інформацію для розуміння поточних проблем і можливостей у цих галузях.

Новизна:

- використання методів статистичного навчання та глибокого підкріплювального навчання для оптимізації серверлес-систем є новаторським внеском;
- запропоновані архітектурні моделі дозволяють підвищити продуктивність і зменшити витрати на інфраструктуру;
- у сфері Flutter-додатків зроблено акцент на інтеграцію сучасних архітектурних рішень для покращення зручності розробки.

Вплив:

- запропоновані методики вже знаходять застосування в практичних системах, зокрема в хмарних обчисленнях і розробці мобільних додатків;
- інноваційні підходи до управління серверлес-обчисленнями значно зменшують затримки та витрати, що робить ці системи більш привабливими для широкого використання;
- архітектурні підходи до Flutter дозволяють спрощувати підтримку коду та підвищувати його продуктивність.

Таким чином, проаналізовані дослідження демонструють значний прогрес у сфері серверлес-обчислень та розробки кросплатформних додатків, закладаючи основу для майбутніх інновацій у цих галузях.

2.4 Висновки з огляду

Загальні висновки з огляду:

- серверлес-технології як інструмент оптимізації. Серверлес-обчислення демонструють значний потенціал у підвищенні продуктивності хмарних платформ завдяки автоматизації масштабування, управлінню ресурсами та зниженню операційних витрат. Вони забезпечують гнучкість у розгортанні та експлуатації додатків, що робить їх привабливими для сучасних інноваційних проєктів;
- роль методів машинного навчання. Інтеграція машинного навчання у процеси оптимізації та управління серверлес-додатками дозволяє автоматично визначати найефективніші конфігурації для різних сценаріїв

використання. Це сприяє вдосконаленню розподілу ресурсів, зменшенню затримок і підвищенню загальної продуктивності систем;

– кросплатформні додатки: інноваційні підходи. У розробці додатків із використанням Flutter значна увага приділяється управлінню станами та архітектурі програм. Новітні підходи, такі як "чиста" архітектура та ефективне управління станами, дозволяють створювати більш підтримувані, масштабовані та продуктивні додатки, що задовольняють потреби сучасних користувачів.

Можна виділити наступні прогалини у дослідженнях:

– відсутність уніфікованих стандартів для серверлес-платформ. Незважаючи на зростання популярності серверлес-технологій, різні хмарні провайдери пропонують відмінні моделі реалізації, що ускладнює інтеграцію та перенесення додатків між платформами. Це створює бар'єри для розробників і збільшує залежність від конкретного постачальника;
– недостатня увага до безпеки. Хоча серверлес-архітектури значно спрощують управління інфраструктурою, вони також відкривають нові вразливості, пов’язані з багатокористувацькими середовищами та функціональною інтеграцією. Питання забезпечення безпеки, зокрема захисту даних та запобігання зловмисним атакам, залишаються недостатньо дослідженими;
– міжплатформна інтеграція. Хоча серверлес-додатки та Flutter активно використовуються окремо, можливості їхньої інтеграції для створення єдиних екосистем поки залишаються обмеженими. Це стримує розробку універсальних рішень для багатоплатформного середовища, які могли б ефективно поєднувати переваги обох технологій.

Можна виокремити наступні причини для подальших досліджень:

– розробка стандартів для серверлес-систем. Для подолання проблем з інтероперабельністю серверлес-платформ необхідно зосередити увагу на розробці уніфікованих стандартів, які сприятимуть взаємодії між різними

провайдерами та зменшать залежність від конкретних сервісів. Це дозволить розробникам створювати більш портативні та масштабовані рішення;

– інтеграція безпеки у серверлес-додатки. Подальші дослідження мають включати розробку спеціалізованих інструментів і методів для забезпечення безпеки серверлес-архітектур. Наприклад, створення автоматизованих систем моніторингу загроз і запобігання атакам, а також впровадження засобів шифрування даних у реальному часі. розширення можливостей Flutter у багатоплатформному середовищі.

Розробка нових бібліотек і фреймворків для інтеграції Flutter із серверлес-платформами дозволить створювати більш ефективні рішення для обробки великих обсягів даних, виконання складних розрахунків і побудови інтерактивних інтерфейсів. Особливу увагу слід приділити оптимізації обміну даними та синхронізації між компонентами.

Загалом, ці напрями досліджень сприятимуть подальшому розвитку серверлес-технологій і підвищенню ефективності їхнього застосування у кросплатформній розробці.

3 ПОСТАНОВКА ЗАДАЧІ

Після проведення аналізу предметної галузі, виокремлення критеріїв оптимізації необхідно сформувати список задач для проведення дослідження.

Для забезпечення достовірності дослідження необхідно вибрати поширену предметну область, щоб створена модель максимально відповідала реальним умовам. Потрібно описати обрану предметну область, розробити структуру бази даних та спроектувати архітектуру застосунку, яка б відповідала вимогам цієї галузі.

Для успішного впровадження безсерверного Flutter-застосунку необхідно обрати оптимальне хмарне рішення, яке забезпечить високу продуктивність, масштабованість та відповідність потребам проєкту. Вибір слід здійснити на основі задачі багатокритеріального вибору, де кожна платформа (така як Firebase, AWS Lambda або Google Cloud Functions) оцінюється за низкою критеріїв. Вирішення задачі багатокритеріального вибору при виборі безсерверного рішення є важливим кроком, оскільки різні платформи мають унікальні переваги та недоліки, що можуть суттєво вплинути на результативність і ефективність застосунку. Наприклад, одні платформи можуть забезпечувати вищу продуктивність, але бути дорожчими або менш зручними для інтеграції з Flutter, тоді як інші мають кращу підтримку інструментів для моніторингу, але поступаються в масштабованості або швидкості.

Задачу вибору оптимального безсерверного рішення для Flutter-застосунку доцільно вирішувати за допомогою методу лінійної адитивної згортки з ваговими коефіцієнтами, оскільки цей метод дозволяє врахувати різні критерії одночасно і порівнювати альтернативи за комплексом показників. Лінійна адитивна згортка є простим і ефективним способом, що передбачає надання кожному критерію вагового коефіцієнта, пропорційного його значущості [17]. Це дозволяє підвищити об’єктивність вибору, фокусуючись на найбільш важливих характеристиках.

Цей метод особливо ефективний для систематизації та порівняння складних альтернативних рішень, де різні критерії мають різну вагу. Згортка з ваговими

коефіцієнтами дозволяє отримати загальну оцінку для кожної платформи, інтегруючи її по всіх критеріях, що сприяє швидкому й обґрунтованому визначенню найкращого варіанту. Це робить метод зручним та універсальним інструментом для вибору оптимального безсерверного рішення у контексті специфічних вимог дослідження.

Для проведення дослідження продуктивності оптимізаційних методів було прийнято рішення побудувати математичну модель експерименту у вигляді лінійної моделі без залежностей між факторами з кількох причин, що обґрунтовані специфікою дослідження методів оптимізації рендерингу в безсерверних Flutter-застосунках:

- незалежність методів оптимізації: основною причиною вибору лінійної моделі є те, що кожен метод оптимізації рендерингу UI-шару застосовується окремо, і їх ефективність не залежить один від одного. Це дозволяє використовувати просту лінійну модель, де кожен фактор (метод оптимізації) має власний вплив на результат (час рендерингу), не створюючи взаємозалежностей між ними. Таким чином, зміна одного методу не буде безпосередньо впливати на результати інших, що дозволяє побудувати модель без урахування складних взаємодій;
- спрощування складності експерименту: лінійна модель є однією з найпростіших математичних моделей, що дозволяє ефективно оцінювати окремі впливи факторів без необхідності ускладнювати модель взаємозалежними змінними. Оскільки дослідження фокусується на порівнянні ефективності різних методів оптимізації, простота лінійної моделі дозволяє зберегти прозорість аналізу та зменшити можливість помилок при трактуванні результатів;
- можливість вимірювання та порівняння: для кожного методу оптимізації буде виміряно час рендерингу UI-шару в двох сценаріях: з використанням технік оптимізації та без них. Лінійна модель дозволяє чітко порівняти ці два варіанти для кожного методу, оцінюючи, який саме метод впливає на покращення продуктивності. Кожен метод можна

розглядати як окремий фактор, вплив якого вимірюється незалежно від інших методів;

- зручність для аналізу результатів: лінійна регресія дозволяє чітко оцінити внесок кожного методу оптимізації в загальний результат. Це дає можливість оцінити не лише загальний час рендерингу, але й кількісно визначити, наскільки кожен метод впливає на ефективність роботи застосунку. Такий підхід дозволяє отримати конкретні та інтуїтивно зрозумілі результати, що зручні для подальшого аналізу та прийняття рішень щодо вибору найбільш ефективних методів;
- спростування впливу випадкових змінних: Лінійна модель без залежностей між факторами дозволяє мінімізувати вплив випадкових змінних та шуму в даних. Оскільки кожен метод оцінюється окремо, можна більш точно виміряти його ефективність без спотворення результатів взаємодією методів, що забезпечує високу достовірність експерименту.

З огляду на вищезгадані фактори, лінійна модель є оптимальним вибором для дослідження ефективності методів оптимізації рендерингу, оскільки вона забезпечує точність, простоту та зручність аналізу при мінімізації складності експерименту.

Отже, під час дослідження необхідно вирішити наступний список задач:

- обрати оптимальне безсерверне рішення для розробки Flutter-застосунку, враховуючи різні критерії, які впливають на зручність використання, функціональні можливості, масштабованість і витрати;
- спроектувати структуру бази даних – визначити основні сутності, їх властивості та зв'язки між ними, що забезпечать ефективне зберігання та доступ до даних у хмарному середовищі;
- розробити схему архітектури застосунку – визначити компоненти системи, їх взаємодію та основні вимоги до реалізації серверної та клієнтської частин;

- створити базу даних у хмарі – налаштувати базу даних для інтеграції з безсерверною логікою застосунку у обраному хмарному сервісі;
- спроектувати та розробити програмне забезпечення для дослідження – створити прототип застосунку, який дозволить тестувати різні методи оптимізації рендерингу інтерфейсу;
- провести експериментальне дослідження обраних методів оптимізації рендерингу – використовувати лінійну регресійну модель для аналізу продуктивності різних методів, зокрема, вимірюючи час рендерингу інтерфейсу з застосуванням і без застосування оптимізаційних технік.

Ці завдання спрямовані на побудову математичної моделі, яка дозволить об'єктивно оцінити вплив кожного методу оптимізації на продуктивність інтерфейсу без залежностей між факторами. Після реалізації прототипу будуть проведені заміри часу на рендеринг UI-шару, що дозволить на практиці порівняти ефективність методів.

Крім того, для реалізації даного проєкту необхідно врахувати певні обмеження, пов'язані з вибором інструментів і ресурсів, серед яких: обмеження на обсяги даних, вимоги до швидкості відгуку, особливості хмарного середовища та можливі технічні обмеження хмарних провайдерів.

Перелік необхідних ресурсів:

- доступ до обраного хмарного сервісу для реалізації бази даних;
- технічні засоби для виконання тестів (наприклад, симулятори або реальні пристрої для тестування продуктивності застосунку);
- інструменти для моніторингу та аналізу продуктивності (наприклад, Flutter DevTools);
- доступ до математичних бібліотек для створення та аналізу лінійних регресійних моделей.

Цей підхід дозволить комплексно вирішити задачі проєкту та отримати достовірні результати щодо ефективності методів оптимізації рендерингу для безсерверних Flutter-застосунків.

4 ТЕОРЕТИЧНЕ ДОСЛІДЖЕННЯ

4.1 Вирішення задачі багатокритеріального вибору для обрання безсервеного рішення

Вибір задачі багатокритеріального вибору для обрання безсерверного рішення для Flutter-застосунку є важливим етапом в рамках дослідження оптимізації безсерверних застосунків із кількох причин:

- вплив на оптимізацію застосунку: вибір платформи є ключовим фактором для подальшої оптимізації. Різні платформи мають свої переваги та обмеження в аспектах продуктивності, масштабованості, підтримки функціональності тощо. Обрання найбільш підходящої платформи допоможе забезпечити оптимальну роботу застосунку;
- мультикритеріальний підхід для повноцінного аналізу: для ефективної оптимізації необхідно враховувати низку критеріїв, таких як простота інтеграції, швидкодія, масштабованість та вартість. Багатокритеріальний аналіз дозволяє об'єктивно оцінити, яке рішення забезпечить найкращий баланс між продуктивністю, вартістю та функціональністю;
- практичне значення для подальших етапів дослідження: результати цієї задачі створюють базу для аналізу способів оптимізації. Вивчення різних рішень і їх характеристик дозволить визначити оптимальні стратегії для вдосконалення архітектури та продуктивності Flutter-застосунків;
- забезпечення наукової обґрунтованості дослідження: використання наукового підходу для обрання безсерверного рішення створює обґрунтовану основу для подальших експериментів і тестувань, що підвищить якість і валідність отриманих результатів у рамках дослідження.

Таким чином, розв'язання задачі багатокритеріального вибору є необхідним кроком для побудови ефективної методології оптимізації безсерверних Flutter-застосунків.

Нижче наведено опис множини альтернатив для задачі вибору:

- Firebase – це комплексне рішення від Google, яке надає розробникам широкий спектр інструментів для створення та підтримки мобільних додатків. Основними перевагами Firebase є його простота у використанні та потужні можливості для управління даними в реальному часі. Проте, великі обсяги даних або складні запити можуть призвести до проблем з продуктивністю. Для їх вирішення можна використовувати функції кешування та оптимізувати структуру бази даних;
- AWS Amplify – це платформа, яка забезпечує простий спосіб інтеграції хмарних сервісів Amazon у ваші додатки. Вона підтримує аутентифікацію, аналітику, зберігання файлів та бази даних. Основним викликом при використанні AWS Amplify є управління складністю проекту, оскільки безліч сервісів можуть потребувати налаштувань. Використання шаблонів та керованих сервісів може спростити цю задачу;
- Azure Functions від Microsoft пропонує можливість запуску коду в безсерверному середовищі, що дозволяє зменшити витрати на управління інфраструктурою. Вона відмінно підходить для реалізації мікросервісів. Проте, затримки при запуску функцій можуть вплинути на користувацький досвід. Оптимізація функцій та використання тригерів можуть допомогти вирішити цю проблему;
- DynamoDB – це NoSQL база даних від Amazon, яка забезпечує високу швидкість роботи та масштабованість. Вона ідеально підходить для додатків, що вимагають швидкого доступу до великих обсягів даних. Однак, неправильна структура таблиць або непродумане використання запитів можуть призвести до низької продуктивності. Важливо враховувати принципи оптимізації, такі як використання індексів та відповідна модель даних;
- Supabase – це сучасна альтернатива Firebase, яка надає безкоштовний та відкритий доступ до хмарних сервісів. Вона включає в себе базу даних, аутентифікацію та реального часу. Основною проблемою, з якою можуть

зіткнутися розробники, є обмежена документація та підтримка. Залучення спільноти та використання прикладів з відкритих джерел можуть допомогти в актуалізації рішень.

Вибір критеріїв для оцінки безсерверних платформ обґрунтований прагненням отримати об’єктивний і багатосторонній аналіз, що враховує як функціональні, так і економічні показники платформи, важливі для розробки та оптимізації Flutter-застосунків:

- популярність платформи серед Flutter-розробників (дані з опитування Stack Overflow [18]): популярність відображає досвід і задоволеність розробників платформою, що є індикатором її зручності, стабільності та наявності ресурсів для підтримки. Популярні платформи, як правило, мають активну спільноту, багато готових рішень і документацію, що спрощує інтеграцію й усунення проблем;
- кількість безкоштовних запитів на читання на місяць: цей критерій є важливим показником, оскільки безкоштовна квота допомагає оптимізувати витрати на проєкт. Для багатьох Flutter-застосунків, які можуть часто запитувати дані (наприклад, новинні стрічки або інформаційні панелі), обсяг безкоштовних запитів дозволяє мінімізувати витрати на початкових етапах проєкту;
- вартість запитів на читання після використання квоти (за мільйон прочитань): цей критерій важливий для проєктів, що масштабуються. Після вичерпання безкоштовної квоти вартість запитів впливатиме на бюджет застосунку. Доступна ціна допомагає економічно підтримувати високі обсяги запитів;
- обсяг даних, який можна зберігати безкоштовно: оскільки зберігання даних є однією з основних функцій безсерверних платформ, розмір безкоштовної квоти на зберігання є критичним. Безкоштовний обсяг зберігання дозволяє оптимізувати витрати на початкових етапах і може зменшити необхідність раннього збільшення бюджету;

– кількість мов програмування, які підтримує платформа: підтримка декількох мов є перевагою, оскільки дозволяє гнучко інтегрувати різні функції, а також залучати до роботи над проєктом розробників з різними технічними знаннями. Це сприяє адаптивності платформи і полегшує інтеграцію з іншими сервісами.

Таким чином, ці критерії дозволяють охопити функціональні та економічні аспекти вибору, що є особливо актуальним для оптимізації безсерверних Flutter-застосунків.

Firebase має оцінку популярності 10. Кількість безкоштовних запитів на читання запитів за місяць – 1,5 млн. Вартість запитів на читання після використання квоти за мільйон прочитань – 0,39$. Обсяг даних, який може зберігатись безкоштовно – 1 гб [19]. Кількість мов програмування, які підтримує платформа – 7.

AWS Amplify має оцінку популярності 8. Кількість безкоштовних запитів на читання запитів за місяць – 0,5 млн. Вартість запитів на читання після використання квоти за мільйон прочитань – 0,3$. Обсяг даних, який може зберігатись безкоштовно – 5 гб [20]. Кількість мов програмування, які підтримує платформа – 10.

Azure Functions має оцінку популярності 7. Кількість безкоштовних запитів на читання запитів за місяць – 1 млн. Вартість запитів на читання після використання квоти за мільйон прочитань – 0,2$. Обсяг даних, який може зберігатись безкоштовно – 32 гб [21]. Кількість мов програмування, які підтримує платформа – 30.

DynamoDB має оцінку популярності 6. Кількість безкоштовних запитів на читання запитів за місяць – 0,5 млн. Вартість запитів на читання після використання квоти за мільйон прочитань – 0,3$. Обсяг даних, який може зберігатись безкоштовно – 5 гб [20] Кількість мов програмування, які підтримує платформа – 10.

Supabase має оцінку популярності 7. Кількість безкоштовних запитів на читання запитів за місяць – безлімітна. Вартість запитів на читання після

використання квоти за мільйон прочитань – 0$, оскільки ліміту немає. Обсяг даних, який може зберігатись безкоштовно – 5 гб [22]. Кількість мов програмування, які підтримує платформа – 7.

Далі необхідно описати та проаналізувати шкали.

Популярність платформи серед Flutter-розробників.

Тип шкали: Шкала порядку.

Опис: Оцінка популярності платформи серед розробників Flutter на основі результатів досліджень, таких як опитування розробників. Платформи оцінюються за шкалою від 1 до 10, де 10 означає найбільшу популярність.

Приклади значень:

– 10: платформа найпопулярніша серед Flutter-розробників;
– 5-9: середня популярність;
– 1-4: низька популярність.

Кількість безкоштовних запитів на читання запитів за місяць.

Тип шкали: Шкала співвідношень.

Опис: Оцінка кількості безкоштовних запитів на читання даних за місяць, яка визначає, скільки запитів платформа дозволяє виконати безкоштовно, оцінюючи рівень користування платформою.

Приклади значень:

– 1,5 млн запитів – 1,5: висока кількість безкоштовних запитів;
– 0,5 млн запитів – 0,5: мала кількість;
– 1 млн запитів – 1: середня кількість;
– безліміт – 10: найкраща опція з безлімітними запитами.

Вартість запитів на читання після використання квоти за мільйон прочитань.

Тип шкали: Шкала співвідношень.

Опис: Вартість запитів після використання безкоштовної квоти. Чим менша вартість, тим вигідніше для розробника. Шкала оцінює цю вартість у доларах за мільйон запитів.

Приклади значень:

– 0,39$ – 0,39: висока вартість запитів;
– 0,3$ – 0,3: середня вартість;
– 0,2$ – 0,2: низька вартість;
– 0$ – 0: безкоштовно після вичерпання квоти.

Обсяг даних, який може зберігатись безкоштовно

Тип шкали: Шкала співвідношень.

Опис: Оцінка обсягу безкоштовного зберігання даних. Шкала вимірює обсяг даних у гігабайтах, який платформа дозволяє зберігати безкоштовно.

Приклади значень:

– 1 ГБ – 1: мінімальний обсяг для зберігання даних;
– 5 ГБ – 5: середній обсяг для зберігання;
– 32 ГБ – 32: великий обсяг для зберігання даних.

Кількість мов програмування, які підтримує платформа

Тип шкали: Шкала співвідношень.

Опис: Шкала оцінює кількість мов програмування, які підтримує платформа. Чим більше мов підтримується, тим більша гнучкість платформи.

Приклади значень:

– 1-3 мови: кілька популярних мов підтримується;
– 4-6 мов: достатня кількість мов підтримується;
– більше 6 мов: велика кількість мов підтримується.

Отже, аналіз шкал можна підсумувати наступним чином:

– шкала порядку використовується для оцінки популярності, де є порядок значень, але різниця між ними не має однакового значення;
– шкала співвідношень використовується для оцінки кількості мов програмування, де значення є кількісними і можна визначати відношення між ними (наприклад, одна платформа підтримує вдвічі більше мов, ніж інша).

Для векторний опису альтернатив за обраними критеріями з використанням відповідних шкал необхідно сформулювати шкалу оцінок за критеріями:

а) кількість безкоштовних запитів на читання запитів за місяць:

1. 1,5 млн – 1,5;
2. 0,5 млн – 0,5;
3. 1 млн – 1;
4. безліміт – 10;

б) вартість запитів на читання після використання квоти за мільйон прочитань:

1. 0,39$ – 0,39;
2. 0,3$ – 0,3;
3. 0,2$ – 0,2;
4. 0$ – 0;

в) обсяг даних, який може зберігатись безкоштовно:

1. 1 гб – 1;
2. 5 гб – 5;
3. 32 гб – 32.

Нижче наведено таблицю 4.1 для моделювання задачі прийняття рішень.

Таблиця 4.1 – Моделювання задачі прийняття рішень для вибору хмарної платформи для безсерверних застосунків

Платформа	Популярність серед Flutter-розробників	Кількість безкоштовних запитів	Вартість запитів після перевищення квоти	Безкоштовний обсяг бази даних	Кількість мов програмування
Firebase	10	1,5	0,39	1	7
AWS Amplify	8	0,5	0,3	5	10
Azure Fn	7	1	0,2	32	30
DynamoDB	6	0,5	0,3	5	10
Supabase	7	10	0	5	7

Далі необхідно провести нормування по minmax та еталону. Значення еталонів та minmax та еталонів наведено у таблиці 4.2.

Таблиця 4.2 – Показники minmax та еталонні значення

Популярність (minmax)	
min	max
6	10
Кількість безкоштовних запитів	
еталон	10
Вартість запитів після перевищення квоти	
min	max
0	0,39
Безкоштовний обсяг бази даних	
еталон	32
Кількість мов програмування	
min	max
7	30

Minmax буде використовуватись для розрахунків популярності та кількості мов програмування (формула 4.1):

$$f = \frac{f_{\text{значення}} - f_{min}}{f_{max} - f_{min}} \tag{4.1}$$

Minmax функція буде набувати наступного вигляду для розрахування вартості запитів, оскільки чим менша ціна, тим краще (формула 4.2):

$$f = 1 - \frac{f_{\text{значення}} - f_{min}}{f_{max} - f_{min}} \tag{4.2}$$

Звичайне формування по еталону буде розраховуватись для безкоштовного обсягу бази даних та кількості безкоштовних запитів (формула 4.3):

$$f = \frac{f_{\text{значення}}}{f_{\text{еталон}}} \tag{4.3}$$

Нормовані значення наведено у таблиці 4.3.

Таблиця 4.3 – Нормалізовані дані таблиці 4.2.

Платформа	Популярність серед Flutter-розробників	Кількість безкоштовних запитів	Вартість запитів після перевищення квоти	Безкоштовний обсяг бази даних	Кількість мов програмування
Firebase	1	0,15	0	0,3	0
AWS Amplify	0,5	0,05	0,23	0,16	0,13
Azure Functions	0,25	0,1	0,49	1	1
DynamoDB	0	0,05	0,23	0,16	0,13
Supabase	0,25	1	1	0,16	0

За принципом Парето кількість варіантів можна скоротити наступним чином: “Варіант а вважається кращим за варіант b згідно з відношенням Парето, якщо а перевершує b хоча б за одним критерієм і при цьому за жодним іншим критерієм не є гіршим за b.” (див. рис. 4.1).

Платформа	Популярність серед Flutter-розробників	Кількість безкоштовних запитів	Вартість запитів після перевищення квоти	Безкоштовний обсяг бази даних	Кількість мов програмування
Firebase	1	0,15	0	0,3	0
AWS Amplify	0,5	0,05	0,23	0,16	0,13
Azure Functions	0,25	0,1	0,49	1	1
~~DynamoDB~~	~~0~~	~~0,05~~	~~0,23~~	~~0,16~~	~~0,13~~
Supabase	0,25	1	1	0,16	0

Рисунок 4.1 – Спрощення за принципом Парето (рисунок виконано самостійно)

З принципу Парето випливає, що ми можемо викреслити DynamoDB, оскільки AWS Amplify краще або дорівнює за усіма критеріями порівняння.

Маємо: $ОП_А = \{\text{Firebase}, \text{AWS Amplify}, \text{Azure Functions}, \text{Supabase}\}$.

Задачу вибору оптимального безсерверного рішення для Flutter-застосунку було вирішено робити за допомогою лінійної адитивної згортки з ваговими коефіцієнтами з кількох основних причин:

- множинність критеріїв: у задачі вибору є кілька критеріїв, що включають як кількісні, так і якісні параметри: популярність платформи, кількість безкоштовних запитів, вартість запитів після квоти, обсяг безкоштовного зберігання даних і кількість підтримуваних мов програмування. Лінійна адитивна згортка дозволяє взяти до уваги всі ці критерії одночасно, обчислюючи загальну оцінку для кожної альтернативи на основі їхньої індивідуальної оцінки;
- простота та зрозумілість методу: лінійна адитивна згортка є інтуїтивно зрозумілим і легко впроваджуваним методом. Він дозволяє скласти комбіновану оцінку для кожної платформи, просто додаючи вагові коефіцієнти до кожного критерію. Така проста математична модель дозволяє чітко зрозуміти, які критерії більше впливають на вибір, і дає змогу ефективно порівнювати різні альтернативи;
- гнучкість вагових коефіцієнтів: вагові коефіцієнти дозволяють врахувати важливість кожного з критеріїв у контексті конкретного проекту. Наприклад, для деяких застосунків більш важливою може бути вартість запитів, а для інших – популярність платформи або обсяг зберігання даних. Лінійна адитивна модель дозволяє гнучко коригувати ці ваги залежно від вимог і пріоритетів проекту;
- можливість порівняння альтернатив: лінійна адитивна згортка дає можливість отримати кількісну оцінку для кожної альтернативи, що дозволяє порівнювати різні платформи на основі їхньої загальної оцінки. Це дозволяє вибрати платформу, яка має найвищу оцінку, враховуючи всі критерії одночасно;
- можливість модифікації та адаптації: важливою перевагою є те, що цей метод можна легко адаптувати до змін у вимогах або умовах. Якщо з’являються нові критерії чи змінюються пріоритети, досить змінити

вагові коефіцієнти або додати нові компоненти в модель. Це робить лінійну адитивну згортку з ваговими коефіцієнтами універсальним інструментом для вирішення задач багатокритеріального вибору;

– математична простота: лінійна адитивна згортка є простою математичною технікою, яка дозволяє швидко обчислити сумарну оцінку для кожної альтернативи, використовуючи заздалегідь задані значення критеріїв і вагові коефіцієнти. Це дає змогу швидко отримати результат, що є важливим для прийняття рішень в умовах обмеженого часу;
– збалансування різних одиниць вимірювання: кожен з критеріїв може мати різні одиниці вимірювання (наприклад, кількість запитів, вартість, кількість мов тощо). За допомогою лінійної адитивної згортки можна ефективно комбінувати ці різні одиниці в одну загальну оцінку, скоригувавши їх за допомогою вагових коефіцієнтів. Це дозволяє створити єдину нормалізовану метрику для кожної альтернативи.

Розрахунки будуть проведені за формулою 4.4:

$$Z^* = \max_{i=1,m} \sum_{j=1}^{n} \alpha_j \beta_j a_{ij}, \tag{4.4}$$

де $\alpha_j = \frac{1}{\sum_{i=1}^{m} a_{ij}}$ – нормуючі множники,

β_j – вагові коефіцієнти, які відображають відносний внесок критеріїв до загального критерію,

a_{ij} – елемент у таблиці.

Вибір вагових коефіцієнтів для лінійної адитивної розгортки є ключовим етапом у моделюванні задачі багатокритеріального вибору. Кожен коефіцієнт відображає важливість конкретного критерію для вибору оптимального безсерверного рішення для Flutter-застосунку. Нижче наведено детальне обґрунтування для обраних ваг:

– кількість безкоштовних запитів (коефіцієнт 0,3): кількість безкоштовних запитів є одним з найважливіших критеріїв, оскільки безкоштовні запити

визначають, скільки разів можна здійснити операції з даними без додаткових витрат. Для розробників це означає менші витрати на використання платформи, що особливо важливо на ранніх етапах проекту або для невеликих застосунків. Чим більше безкоштовних запитів, тим більша можливість обробляти дані без фінансових витрат, що підвищує привабливість платформи. Оскільки цей критерій прямо впливає на економічну ефективність проекту, йому було присвоєно високий ваговий коефіцієнт (0,3);

- безкоштовний обсяг бази даних (коефіцієнт 0,3): обсяг безкоштовного зберігання даних також є важливим критерієм, оскільки зберігання великих обсягів даних без додаткових витрат може значно знизити витрати на інфраструктуру, особливо для застосунків, які обробляють багато даних або мають велике сховище. Це критично важливо для серверних і мобільних застосунків, де кожен мегабайт даних може збільшити витрати. Безкоштовний обсяг бази даних визначає, наскільки економічно вигідно буде працювати на платформі в довгостроковій перспективі. Через його високу значущість для бізнес-сторони вибору, цей критерій також отримав ваговий коефіцієнт 0,3;
- вартість запитів після перевищення квоти (коефіцієнт 0,2): вартість додаткових запитів після використання безкоштовної квоти має значення для розробників, оскільки після перевищення ліміту безкоштовних запитів можуть з’явитися додаткові витрати. Однак цей критерій менш критичний порівняно з кількістю безкоштовних запитів та обсягом зберігання, оскільки додаткові витрати виникають лише в разі перевищення квоти, що не завжди є обов'язковим. Тому цей критерій має середній коефіцієнт (0,2), який дозволяє врахувати його вплив на вибір, але не перевищує важливості безкоштовних запитів і обсягу зберігання;
- популярність серед Flutter-розробників (коефіцієнт 0,15): популярність платформи є важливим фактором для вибору, оскільки вибір широко використовуваної платформи може забезпечити більшу підтримку

спільноти, більше ресурсів для навчання та доступ до готових рішень. Однак популярність не є вирішальним фактором у виборі, оскільки важливіші критерії, такі як безкоштовні запити та обсяг зберігання даних, безпосередньо впливають на ефективність і витрати проекту. Зважаючи на це, популярність отримала вагу 0,15, що дозволяє врахувати її вплив, але не зробити її надто значущою;

– кількість підтримуваних мов програмування (коефіцієнт 0,05): кількість підтримуваних мов програмування є найменш важливим критерієм у цій задачі, оскільки для Flutter-застосунку основною мовою буде Dart, і підтримка інших мов може бути корисною, але не критичною для розробника. Тому цей критерій має найменший коефіцієнт (0,05), оскільки його вплив на кінцеве рішення є мінімальним. Платформа, яка підтримує кілька мов програмування, може бути більш універсальною, але для Flutter-розробників це не є визначальним фактором.

Для початку розрахуємо нормуючі коефіцієнти:

$$\alpha_1 = \frac{1}{1+0,5+0,25+0,25} = 0,5, \quad \alpha_2 = \frac{1}{0,15+0,05+0,1+1} = 0,769230769, \quad \alpha_3 = \frac{1}{0+0,23+0,49+1} = 0,58139535, \alpha_4 = \frac{1}{0,3+0,16+1+0,16} = 0,61728395, \alpha_5 = \frac{1}{0+0,13+1+0} = 0,88495575.$$

Проведемо розрахунки згортки для кожної альтернативи.

$$Z^*(Firebase) = 0,5 * 0,05 * 1 + 0,76923077 * 0,3 * 0,15 + 0,581395 * 0,2 * 0 + 0,617284 * 0,3 * 0,3 + 0,884956 * 0,05 * 0 = 0,115171,$$

$$Z^*(AWS\ Amplify) = 0,5 * 0,05 * 0,5 + 0,76923077 * 0,3 * 0,05 + 0,581395 * 0,2 * 0,23 + 0,617284 * 0,3 * 0,16 + 0,884956 * 0,05 * 0,13 = 0,086164,$$

$$Z^*(Azure\ Functions) = 0,5 * 0,05 * 0,25 + 0,76923077 * 0,3 * 0,1 + 0,581395 * 0,2 * 0,49 + 0,617284 * 0,3 * 1 + 0,884956 * 0,05 * 1 = 0,315737,$$

$$Z^*(Supabase) = 0,5 * 0,05 * 0,25 + 0,76923077 * 0,3 * 1 + 0,581395 * 0,2 * 1 + 0,617284 * 0,3 * 0,16 + 0,884956 * 0,05 * 0 = 0,382928.$$

Результати розрахунків лінійної адитивної розгортки наведено у таблиці 4.4.

Таблиця 4.4 – Лінійна адитивна розгортка з ваговими коефіцієнтами.

Платформа	Популярність серед Flutter-розробників	Кількість безкоштовних запитів	Вартість запитів після перевищення квоти	Безкоштовний обсяг бази даних	Кількість мов програмування	Z*
Firebase	1	0,15	0	0,3	0	0,115171
AWS Amplify	0,5	0,05	0,23	0,16	0,13	0,086164
Azure Functions	0,25	0,1	0,49	1	1	0,315737
Supabase	0,25	1	1	0,16	0	0,382928
Нормуючі множники	0,5	0,76923077	0,581395	0,617284	0,884956	
Вагові коефіцієнти	0,05	0,3	0,2	0,3	0,05	

З розрахунку лінійної адитивної розгортки випливає, що Supabase (переважає з найбільшим коефіцієнтом (0,382928)), є найкращим вибором для дослідження оптимізації безсерверних Flutter-застосунків завдяки своїм високим оцінкам за критеріями економічної вигоди (безкоштовні запити, зберігання даних) і популярності серед Flutter-спільноти. Інші сервіси, такі як Azure Functions і Firebase, також мають свої переваги, але не можуть забезпечити таку економічну ефективність, як Supabase. AWS Amplify виявився найменш привабливим варіантом через його високу вартість та обмеження за кількістю безкоштовних запитів і зберігання даних.

4.2 Проектування структури бази даних

У сучасному суспільстві багато людей стикаються з труднощами у пошуку романтичного партнера, що часто стає темою численних соціологічних досліджень, науково-популярних статей та мистецьких творів. Потреба знайти "ідеальну" людину та побудувати стосунки, засновані на взаємній сумісності,

інтересах і цінностях, обумовлена як психологічними, так і соціальними факторами. Одним із креативних підходів до цієї проблеми, що набула популярності в масовій культурі, став сценарій серіалу "Як я зустрів вашу маму". Протягом усіх епізодів головний герой Тед Мосбі знаходиться в пошуку ідеальної партнерки. У 7-му епізоді першого сезону він дізнається про агентство, яке використовує алгоритм для обчислення коефіцієнта сумісності, що допомагає знаходити відповідну людину для кожного клієнта [23]. Цей сюжетний прийом демонструє інтерес до алгоритмічного підходу в розв'язанні романтичних питань, що актуально і в реальному житті.

З урахуванням зростаючого попиту на технологічні рішення в сфері особистих стосунків, у цій роботі пропонується реалізація застосунку, який дозволить користувачам проводити систематичний аналіз сумісності з потенційними партнерами на основі індивідуально заданих критеріїв. Метою розробки є створення інструменту, що дозволяє користувачу оцінювати своїх партнерів за обраними параметрами, на основі яких застосунок розраховуватиме рейтинг кожного партнера. Таким чином, користувач зможе визначити ступінь сумісності з кожним із них, що дасть змогу приймати більш обґрунтовані рішення щодо розвитку стосунків.

Застосунок надає можливість користувачу вводити дані про своїх партнерів і визначати важливі для нього критерії, наприклад, такі як спільність інтересів, емоційна підтримка, спільні життєві цінності та інші. Після заповнення критеріїв і виставлення оцінок застосунок обробляє введені дані, використовуючи алгоритм, що розраховує середньозважений рейтинг кожного партнера. Отримані результати візуалізуються у вигляді списку партнерів, ранжованих за рівнем відповідності бажаним характеристикам користувача. Таким чином, алгоритм допомагає виявити партнерів з найвищою сумісністю та потенційною перспективою розвитку стосунків.

Цей підхід до аналізу сумісності партнерів базується на науково обґрунтованих підходах, зокрема методах багатофакторного аналізу та

психологічних принципах вибору, що робить його ефективним інструментом у пошуку партнера.

Для розробки застосунку було розроблено ER-діаграму бази даних (див. рис. 4.2), яка складається із 4 таблиць: User, Partner, Criteria, Mark.

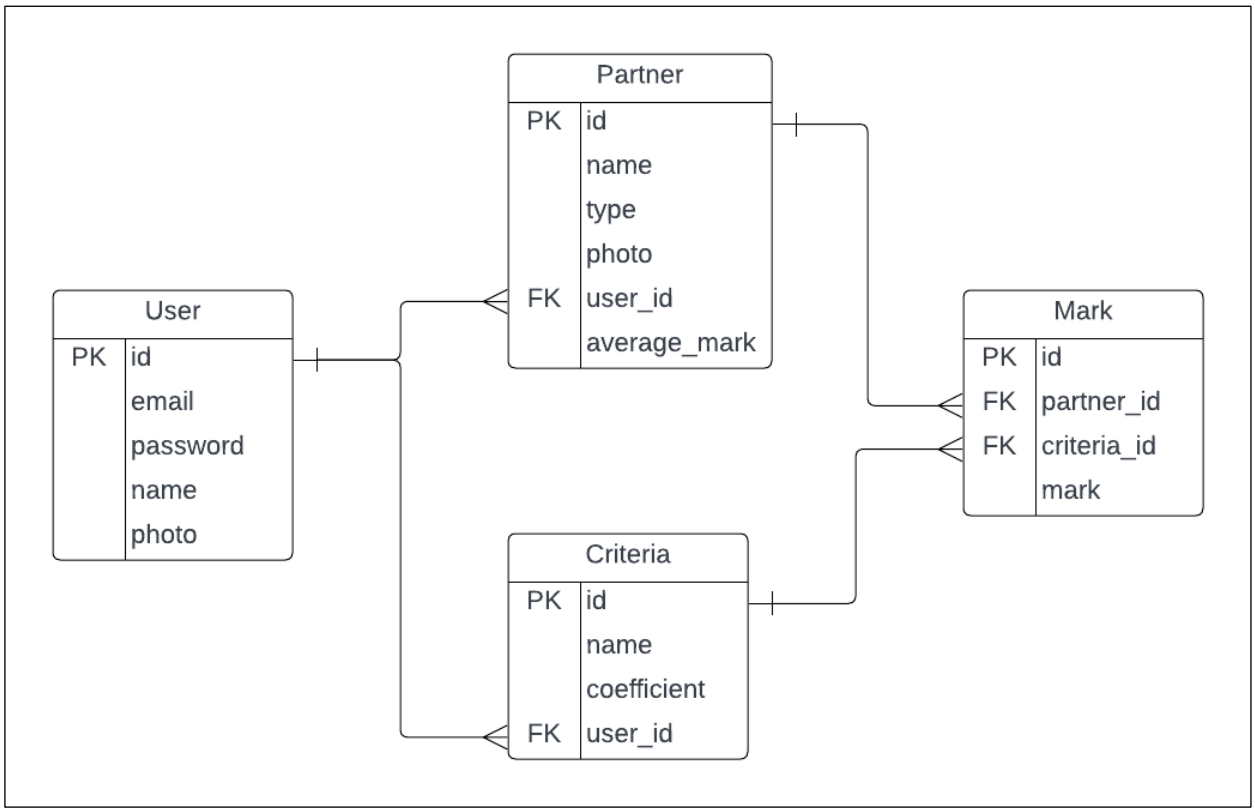

Рисунок 4.2 – Схема бази даних застосунку (рисунок виконано самостійно)

Сутності та їх атрибути:

а) User (Користувач): таблиця User зберігає інформацію про користувачів застосунку:
   1. id: первинний ключ (PK), унікальний ідентифікатор користувача;
   2. email: адреса електронної пошти користувача;
   3. password: пароль користувача для автентифікації;
   4. name: ім'я користувача;
   5. photo: посилання або шлях до фото профілю користувача;

б) Partner (Партнер): таблиця Partner містить інформацію про партнерів, яких додав користувач для оцінки:
   1. id: первинний ключ (PK), унікальний ідентифікатор партнера;

2. name: ім'я партнера;
3. type: тип партнера (наприклад, може містити інформацію про статус або категорію);
4. photo: посилання або шлях до фото партнера;
5. user_id: зовнішній ключ (FK), що посилається на id у таблиці User, показуючи, якому користувачу належить партнер;
6. average_mark: закешований результат середньої оцінки;

в) Criteria (Критерії): таблиця Criteria зберігає критерії оцінки, які користувач застосовує до своїх партнерів:

1. id: первинний ключ (PK), унікальний ідентифікатор критерію;
2. name: назва критерію (наприклад, "спільність інтересів", "емоційна підтримка" тощо);
3. coefficient: вага критерію, яка визначає його значимість у загальному рейтингу;
4. user_id: зовнішній ключ (FK), що посилається на id у таблиці User, вказуючи, якому користувачу належать критерії;

г) Mark (Оцінка): таблиця Mark містить оцінки, які користувачі виставляють партнерам за певними критеріями:

1. id: первинний ключ (PK), унікальний ідентифікатор оцінки;
2. partner_id: зовнішній ключ (FK), що посилається на id у таблиці Partner, визначає, якому партнеру належить оцінка;
3. criteria_id: зовнішній ключ (FK), що посилається на id у таблиці Criteria, вказує, за яким критерієм виставлена оцінка;
4. mark: оцінка за обраним критерієм (від 1 до 5).

Зв'язки між таблицями:

– таблиця User пов'язана з таблицями Partner і Criteria через user_id, що означає, що кожен користувач може мати власний список партнерів та критеріїв;
– таблиця Partner пов'язана з таблицею Mark через partner_id, що дозволяє зберігати різні оцінки для кожного партнера за різними критеріями;

– таблиця Criteria також пов'язана з таблицею Mark через criteria_id, що дозволяє зберігати оцінки за кожен критерій.

Логіка роботи:

– користувач додає партнерів до таблиці Partner та встановлює критерії оцінки у таблиці Criteria;
– для кожного партнера він може виставляти оцінки за певними критеріями, які зберігаються у таблиці Mark;
– на основі коефіцієнтів критеріїв та виставлених оцінок, система може розраховувати загальний рейтинг кожного партнера, використовуючи ваги критеріїв.

Примітки щодо безпеки:

– у таблиці User, важливо зберігати паролі у хешованому вигляді, а не у відкритому тексті, для захисту даних користувачів;
– доступ до таблиць повинен бути обмеженим, особливо до таблиці User, щоб уникнути витоку особистої інформації.

Спроектована забезпечує зручний і структурований спосіб управління інформацією, що необхідна для роботи додатку [6]. Можливість додавання нових партнерів, визначення індивідуальних критеріїв та аналізу сумісності робить її гнучкою та корисною для користувачів, які хочуть обґрунтовано підійти до питання вибору партнера.

4.3 Розробка архітектури застосунку

Під час створення застосунку був застосований підхід Clean Architecture, який дозволяє звести до мінімуму залежності між компонентами програмної системи [24]. Це забезпечує легкість у додаванні нової функціональності та внесенні змін. Основна ідея Clean Architecture полягає у поділі системи на кілька шарів, кожен із яких має чітко визначену роль і відповідний інтерфейс:

– Presentation layer: відповідає за взаємодію з користувачем, включаючи відображення інформації та обробку введених даних. Цей шар слугує інтерфейсом системи, не містячи бізнес-логіки;

– Domain layer: містить ключову бізнес-логіку та правила, які визначають функціонування системи. Це основний шар, де реалізуються всі бізнес-вимоги;
– Infrastructure layer: відповідає за взаємодію із зовнішніми сервісами, такими як бази даних, файлові системи або мережеві служби, забезпечуючи реалізацію технічних аспектів роботи системи;
– Data layer: займається зберіганням і доступом до даних. У ньому розташовані репозиторії, що забезпечують абстрагування від конкретних джерел даних.

Кожен із цих шарів (див. рис. 4.3) має свої чітко визначені функції та відповідальності, що дозволяє розвивати систему без необхідності змінювати її основні компоненти. Для збереження автономності шарів і мінімізації залежностей між ними було застосовано принцип інверсії залежностей (Dependency Inversion Principle) [25]. Це забезпечує незалежність доменного шару від реалізації інфраструктурного та шару даних, що значно спрощує тестування і підтримку системи [7].

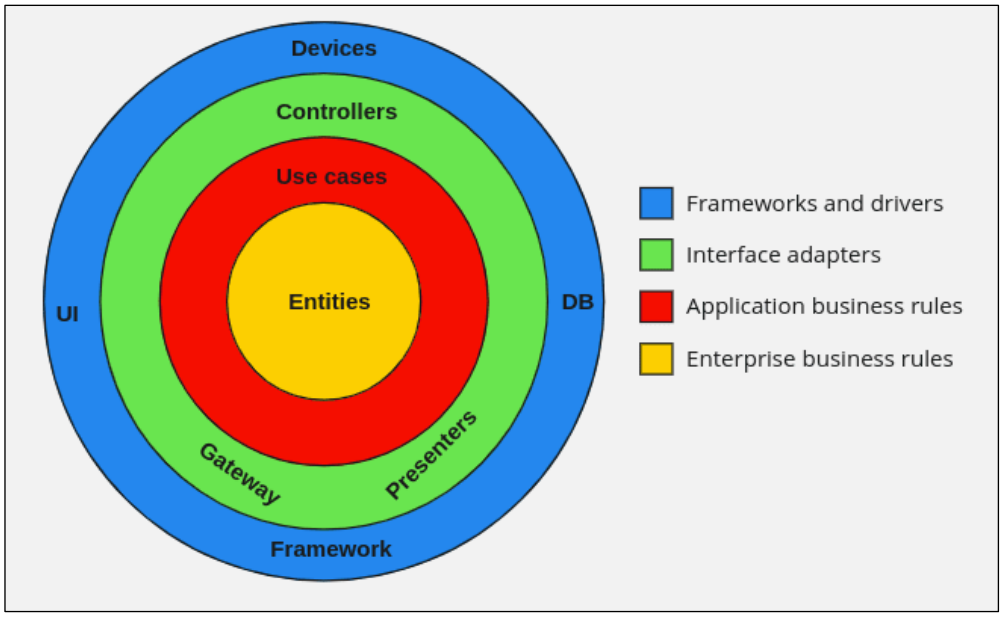

Рисунок 4.3 – Шари застосунку, спроектованого за підходом Clean Architecture (Джерело: Vasconcelos R. Clean Architecture: The concept behind the code. URL: https://dev.to/rubemfsv/clean-architecture-the-concept-behind-the-code-52do (дата звернення: 18.05.2025).)

Завдяки чіткому розмежуванню обов’язків між шарами система отримує кращу масштабованість та гнучкість. Кожен шар взаємодіє з іншими виключно через визначені інтерфейси, що дозволяє легко замінювати або модифікувати реалізацію окремих компонентів без впливу на інші частини архітектури. Наприклад, при зміні способу збереження даних або переході на іншу базу даних, достатньо оновити лише шар доступу до даних, не торкаючись логіки доменного шару.

Крім того, така архітектура сприяє впровадженню практик тестування – зокрема юніт-тестів. Оскільки бізнес-логіка не залежить від зовнішніх сервісів чи інфраструктурних рішень, її можна тестувати ізольовано, підключаючи замінники (mocks) або заглушки (stubs) необхідних інтерфейсів. Це не лише покращує якість коду, але й пришвидшує цикл розробки та підвищує стабільність системи в цілому.

Використання принципу інверсії залежностей [25] також полегшує впровадження нових функцій. Оскільки залежності передаються зовні (наприклад, через механізми впровадження залежностей – Dependency Injection), розробникам не потрібно переписувати існуючий код для інтеграції нових сервісів або компонентів. Це робить систему більш відкритою до змін та адаптації до нових вимог бізнесу чи технологічних змін.

Загалом, поділ на шари та дотримання принципів SOLID, зокрема інверсії залежностей, створює передумови для створення надійної, підтримуваної та розширюваної програмної системи. Такий підхід є особливо ефективним у великих проєктах, де важливо забезпечити стабільність і прогнозованість змін.

Для розробки застосунку за принципами Clean Architecture наведену вище структуру було розподілено на 5 основні шарів: Data, Domain, Presentation, Infrastructure та Executable. На основі цієї архітектури створено діаграму компонентів (рис. 4.4), яка ілюструє структуру системи, включаючи її компоненти, їхні взаємодії та зв’язки. У контексті Clean Architecture така діаграма наочно відображає поділ системи на шари, їхні складові, а також залежності та взаємозв’язки між компонентами [16].

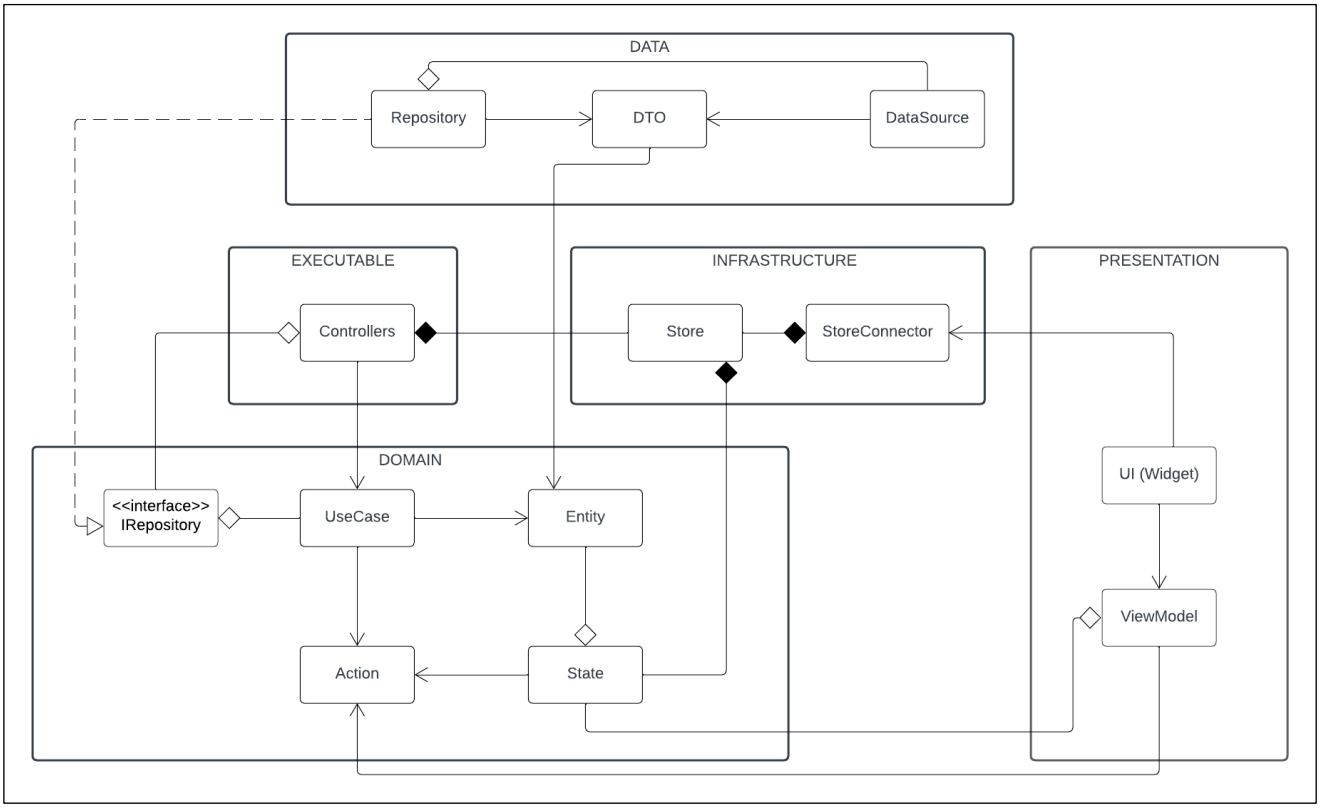

Рисунок 4.4 – Діаграма компонентів застосунку (рисунок виконано самостійно)

Далі наведено детальний опис кожного шару:

а) Data: відповідає за управління даними:

1. DataSource: відповідає за взаємодію із зовнішніми джерелами даних (наприклад, базами даних або API);
2. DTO (Data Transfer Object): призначений для передачі даних між шарами без зайвої логіки та отримує дані від DataSource і передає їх у Repository;
3. Repository: реалізує інтерфейс доступу до даних для інших шарів та використовує DTO для отримання або передачі даних;

б) Domain: концентрується на бізнес-логіці:

1. IRepository (інтерфейс): визначає абстрактний контракт для Repository та забезпечує незалежність бізнес-логіки від реалізації джерел даних;
2. UseCase: містить основну бізнес-логіку та операції та використовує IRepository для отримання/запису даних;
3. Entity: представляє бізнес-об'єкти з логікою та атрибутами;

4. Action: реалізує дії, які можуть виконуватися в системі;
5. State: відповідає за збереження та управління поточним станом об'єктів;

в) Executable: відповідає за виконання дій та координацію компонентів:
1. Controllers: контролери обробляють запити та викликають відповідні UseCase у шарі Domain**;**

г) Infrastructure: забезпечує технічну інфраструктуру для роботи з даними:
1. Store: відповідає за управління станом системи;
2. StoreConnector: забезпечує зв'язок між Store та іншими компонентами, наприклад, ViewModel у шарі Presentation**;**

д) Presentation: відповідає за представлення даних користувачеві:
1. UI (Widget): являє собою графічний інтерфейс користувача та отримує дані від ViewModel і відображає їх;
2. ViewModel: є посередником між UI та бізнес-логікою та отримує дані зі Store і надає їх UI у зручному форматі.

Взаємозв'язки:

- DataSource передає дані у DTO, а DTO використовується у Repository;
- Repository через інтерфейс IRepository використовується шаром Domain (UseCase);
- UseCase працює із State, Entity, та Action;
- Controllers викликають UseCase, координуючи дії між Domain та іншими компонентами;
- Store оновлюється через StoreConnector, а його стан передається у ViewModel для відображення в UI.

Щоб візуалізувати потік даних і послідовність дій у системі, було створено Data Flow діаграму (див. рис. 4.5). Діаграма демонструє, як дані проходять через усі рівні архітектури – від дій користувача до оновлення стану й відображення змін. Вона також показує модульність і розподіл відповідальності між різними компонентами, що спрощує тестування та масштабованість системи [15].

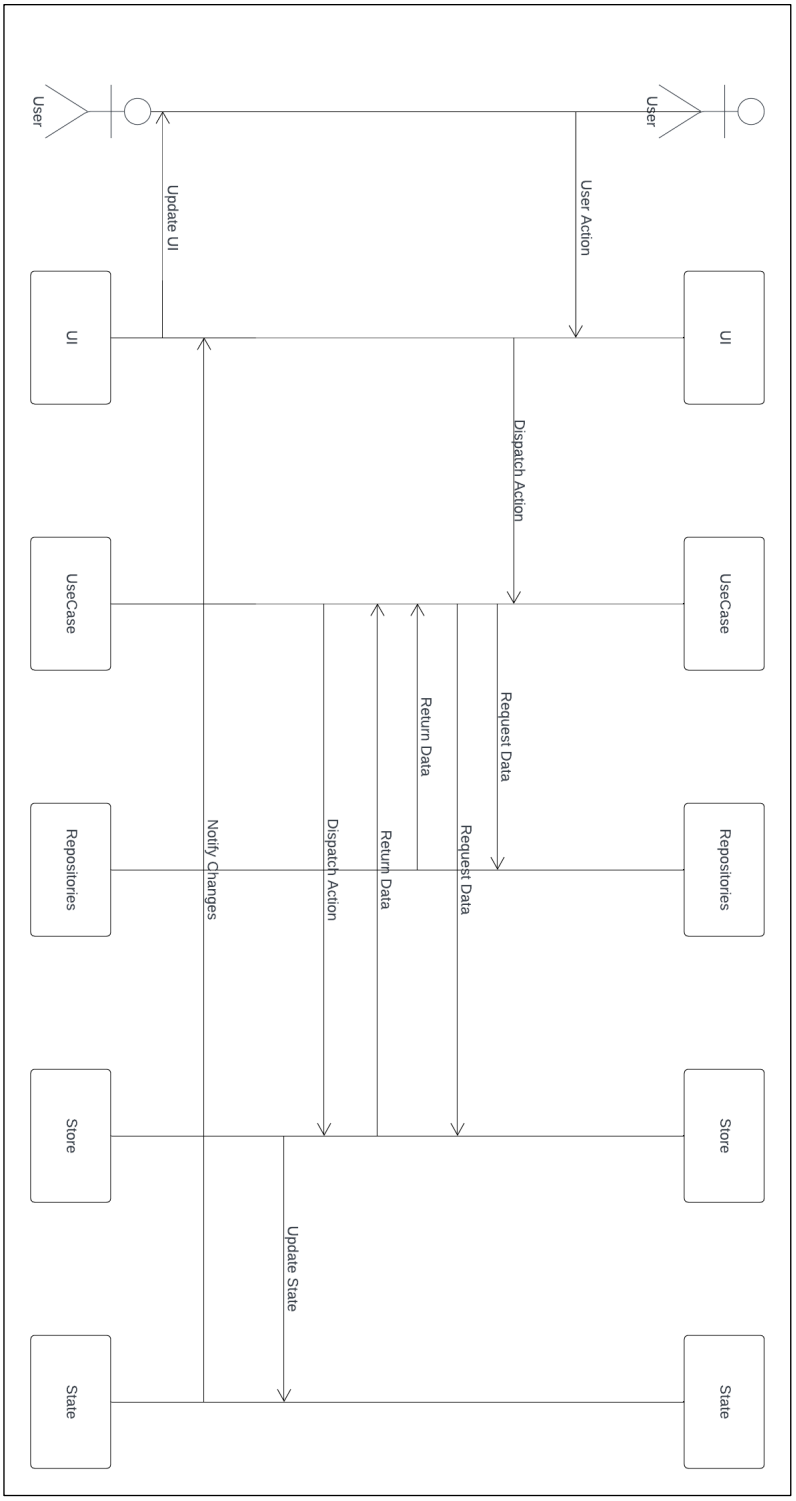

Рисунок 4.5 – Data Flow діаграма застосунку (рисунок виконано самостійно)

Учасники:

– User – користувач, який взаємодіє з інтерфейсом;
– UI – інтерфейс користувача, що отримує дії користувача і передає їх у систему;

- UseCase – реалізація бізнес-логіки для виконання конкретних операцій;
- Repositories – шар для роботи з даними (отримання або збереження);
- Store – компонент для управління станом програми;
- State – поточний стан програми, який оновлюється на основі виконаних дій.

Процес:

- User Action**:** користувач виконує дію (наприклад, натискає кнопку), Ця дія передається до UI;
- Dispatch Action**:** UI передає дію в UseCase, який відповідає за виконання бізнес-логіки;
- Request Data**:** UseCase звертається до Repositories, щоб отримати або записати необхідні дані;
- Request & Return Data**:** Repositories виконує запит до джерела даних (наприклад, бази даних або API) і повертає отримані дані назад у UseCase;
- Dispatch Action: після отримання або обробки даних **UseCase** ініціює зміну стану, передаючи цю зміну до Store;
- Update State: Store оновлює поточний State на основі дій, отриманих від UseCase;
- Notify Changes: Store повідомляє UI про зміну стану;
- Update UI: UI оновлюється відповідно до нового стану, і результати відображаються користувачеві.

4.4 Створення бази даних у хмарі

Для початку необхідно створити проєкт на сайті сервісу Supabase (див. рис. 4.6), вказавши назву організації, назву проекту, пароль до бази даних та регіон.

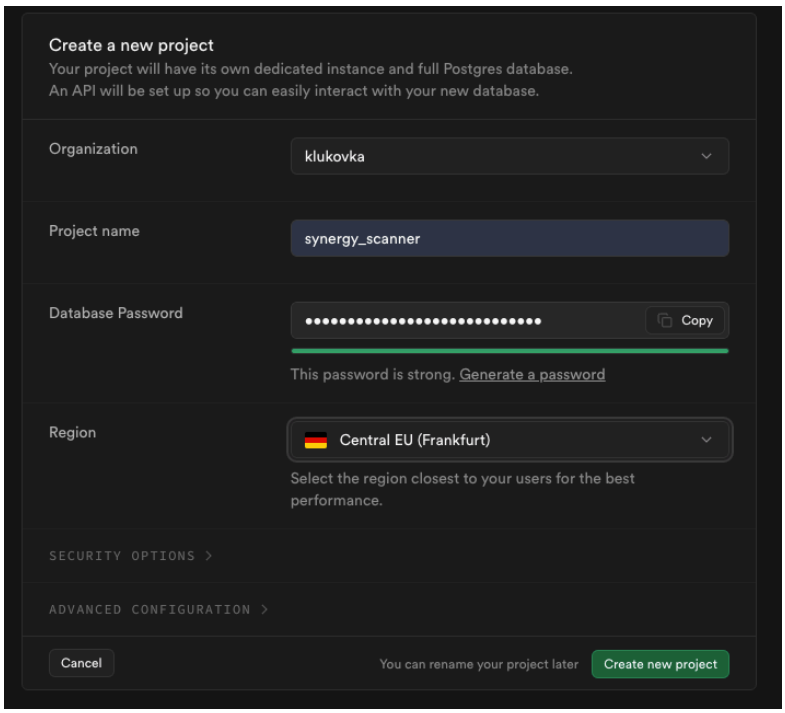

Рисунок 4.6 – Створення нового проекту (рисунок виконано самостійно)

Застосунок буде підтримувати автентифікацію за допомогою електронної пошти та паролю, тому необхідно увімкнути відповідний провайдер у вкладці «Авторизація» (див. рис. 4.7).

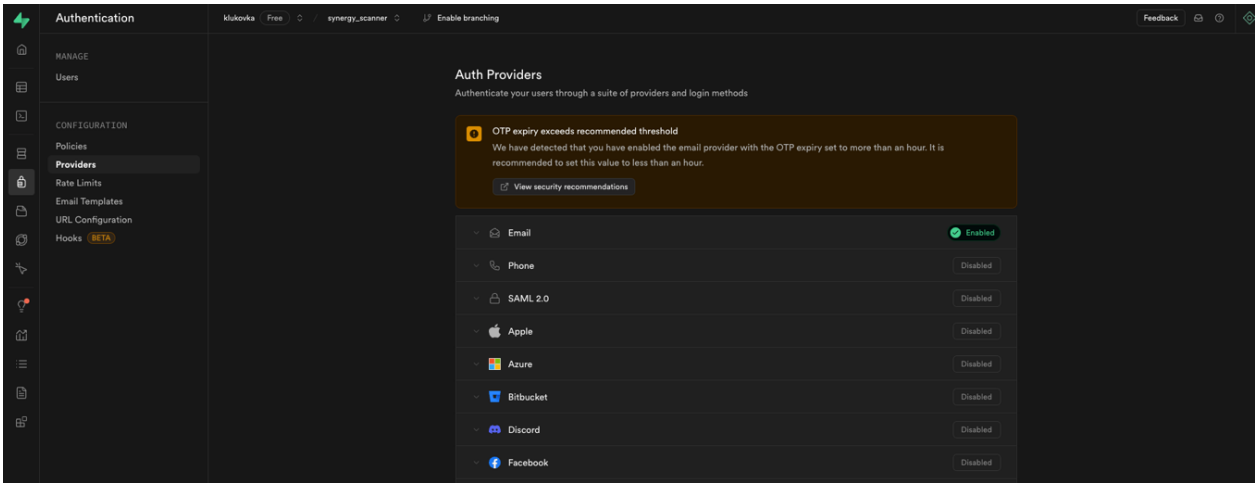

Рисунок 4.7 – Процес встановлення провайдеру автентифікації (рисунок виконано самостійно)

Створювати таблиці у базі даних можна за допомогою SQL-редактора та вручну. Нижче розглянуто обидва способи: таблицю користувачів було створено з використанням редактора, а інші таблиці мануально.

Код створений за допомогою SQL-редактора розміщено у додатку Б.

Код на рисунку А.1 налаштовує базу даних у Supabase для роботи з профілями користувачів та зберіганням аватарок.

Розглянемо основні етапи:

а) створення таблиці для профілів користувачів: код створює таблицю users для збереження даних профілів користувачів:
   1. id: первинний ключ (UUID), що відповідає ідентифікатору користувача в Supabase Auth (auth.users);
   2. email: унікальний email користувача;
   3. full_name: повне ім'я користувача;
   4. avatar_url: URL аватарки користувача;
   5. обмеження: email_length перевіряє, що email має щонайменше 3 символи;

б) налаштування Row Level Security (RLS): активується контроль доступу на рівні рядків, щоб обмежити доступ до даних відповідно до ролі чи ідентифікатора користувача:
   1. політика для select: дозволяє всім бачити дані в таблиці users;
   2. політика для insert: дозволяє користувачам додавати лише свій профіль, перевіряючи, що їхній ідентифікатор відповідає auth.uid();
   3. політика для update: дозволяє користувачам оновлювати тільки свій профіль;

в) тригери для автоматичного створення профілів: створюється функція handle_new_user, яка автоматично додає запис у таблицю users при створенні нового користувача через Supabase Auth:
   1. функція використовує дані із raw_user_meta_data (надані під час реєстрації), щоб заповнити full_name та avatar_url;
   2. тригер виконується після додавання нового користувача в auth.users;

г) налаштування зберігання файлів: код створює bucket (сховище) під назвою avatars для зберігання аватарок:
   1. політика для select: дозволяє всім переглядати аватарки;

2. політика для insert: дозволяє всім завантажувати файли у bucket avatars;
3. політика для update: дозволяє користувачам оновлювати тільки власні файли в цьому bucket.

Розглянемо приклад мануального створення таблиць на прикладі критерій. Для початку у редакторі таблиць створимо відповідну таблицю (див. рис. 4.8), вказавши усі поля та зовнішні ключі.

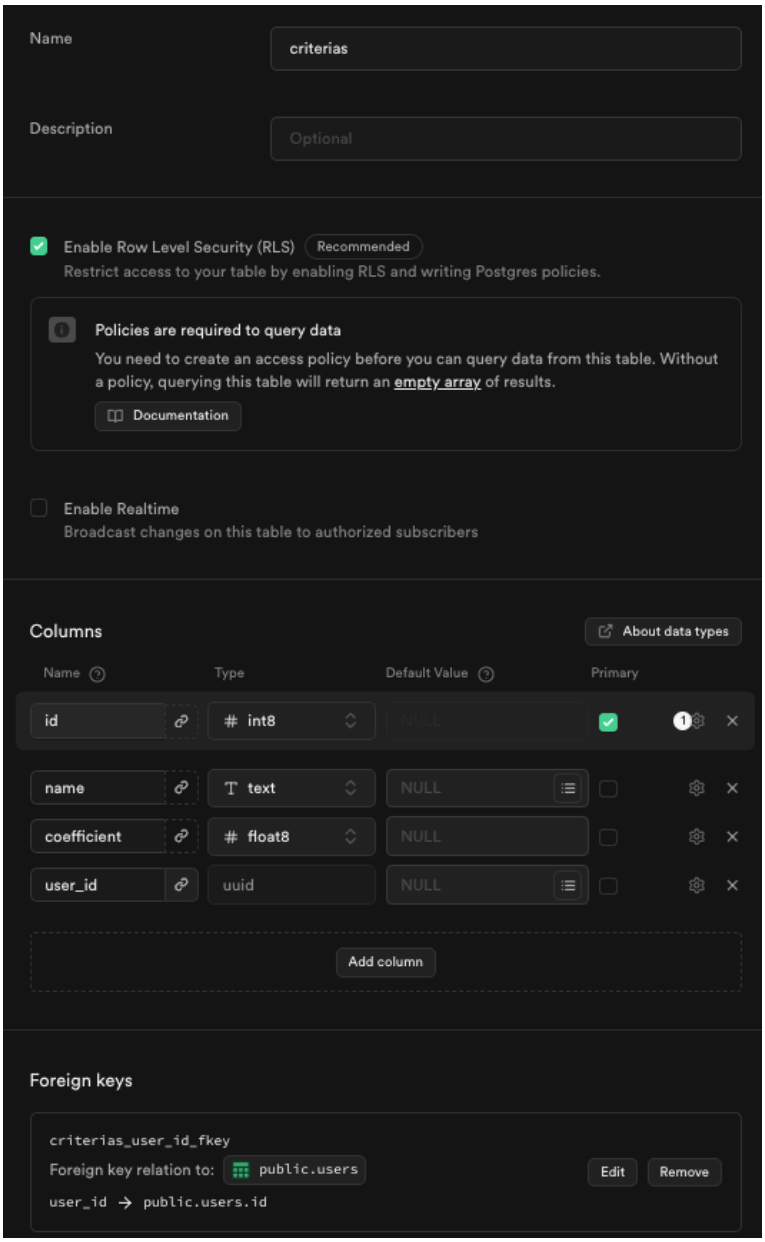

Рисунок 4.8 – Створення таблиці «Критерії» (рисунок виконано самостійно)

Далі у вкладці «Автентифікація» -> «Політики» налаштуємо політики CRUD операцій для цієї таблиці. На рисунку 4.9. розглянуто приклад додавання

політики на оновлення даних (користувач може оновлювати тільки власні дані і має бути при цьому авторизованим).

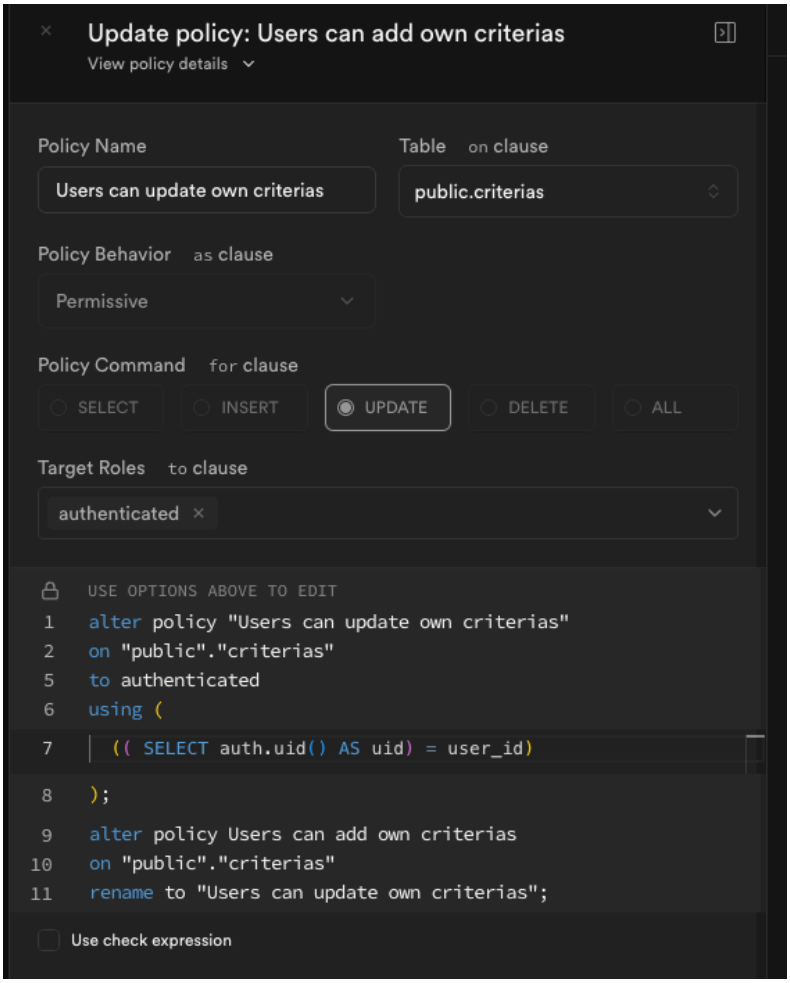

Рисунок 4.9 – Встановлення політики оновлення таблиці «Критерії» (рисунок виконано самостійно)

Аналогічно налаштовуються таблиці «Партнери» та «Оцінки».

Далі у вкладці «База даних» -> «Функції» необхідно створити функцію, яка буде оновлювати середню оцінку партнера на основі виставлених оцінок і критерій коефіцієнтів (див. рис. Б.2).

Далі у вкладці «Тригери» налаштовуються тригери, які будуть оновлювати викликати попередньо створену функцію (див. рис. Б.3) у разі додавання, оновлення чи видалення критерію чи оцінки.

Для отримання списку партнерів з оцінками було створено наступну функцію, яка має у собі відповідний SQL-запит (див. рис. Б.4).

Аналогічний код для повернення критеріїв з оцінками наведено на рисунку А.5.

Для створення теплової карти кореляції між критеріями було створено функцію (див. рис. Б.6). Цей графік використовується для виявлення, які характеристики зазвичай йдуть разом (наприклад, висока спільність інтересів часто супроводжується емоційною підтримкою).

Налаштування Supabase як сервера для безсерверного Flutter-застосунку завершено. Це означає, що всі необхідні компоненти, такі як база даних, автентифікація користувачів, хмарне зберігання та API-запити, успішно налаштовані та готові до використання. Тепер можна переходити до інтеграції функціоналу застосунку, оптимізації запитів та налаштування додаткових можливостей, таких як обробка реального часу, безпечне управління правами доступу та покращення продуктивності.

4.5 Розробка програмного забезпечення для проведення дослідження

Для проведення дослідження було створено сторінку, на якій буде відображатись список створених партнерів. Дослідження буде проводитись саме у такому середовищі, оскільки списки – це один із найбільш вживаних способів відображення інформації у застосунках.

Код для кожного із способів використання / невикористання визначених вище оптимізаційних аспектів [14] наведено у додатку Б.

Аспект «уникнення непотрібної переробки віджетів» включає у себе уникнення додаткових викликів методів setState і оновлення ViewModel. У поточному випадку розглянемо використання методу ignoreChange, який буде блокувати повторний рендеринг сторінки, коли він не потрібен (див. рис. В.1).

Константні конструктори дозволяють створювати незмінні віджети. Це дає змогу Flutter ефективніше повторно використовувати їх у пам’яті, що знижує витрати на рендеринг та створення об'єктів. У результаті відбувається зменшення навантаження на пам’ять і підвищення продуктивності. Нижче наведено код із використанням та без використання константних віджетів (див. рис. В.2).

Stateful віджети дорожчі за Stateless, оскільки вони мають стан, що потрібно зберігати і оновлювати. Зайва кількість таких віджетів може уповільнити роботу застосунку. Тому нижче розглянуто два випадки рендерингу одних і тих самих компонентів з використанням Stateful або Stateless віджетів (рис. А.3).

Методи build виконуються щоразу, коли віджет перерисовується. Якщо метод великий і складний, це може спричиняти затримки в інтерфейсі. Далі наведено код з використанням довгого та короткого build-методу (див. рис. В.4).

Helper-методи всередині build часто створюють нові об’єкти при кожному виклику, що впливає на продуктивність. Нижче наведено код з використанням побудови списку за допомогою методів та окремих віджетів (див. рис. В.5).

Рендеринг елементів, які не видно користувачу, споживає ресурси пристрою без жодної вигоди. Тому краще використовувати ListView (або SliverList), ніж SingleChildScrollView. Код використання обох наведено нижче (див. рис. В.6).

Асинхронність і ізоляти дозволяють виконувати ресурсоємні завдання (завантаження даних, обчислення) поза основним потоком, який відповідає за рендеринг UI. Нижче наведено клас, який визначає домінантний колір списку зображень з використанням [26] та без використання ізолятів (див. рис. В.7).

Віджет Opacity додає шар для рендерингу, що збільшує навантаження на GPU. Використання альтернатив, як-от Colors.transparent або управління стилями, є ефективнішим. Також віджет Visibility працює на основі віджету Opacity. Нижче наведено код з використанням цього віджета та альтернативний без його використання (див. рис. В.8).

Неналежна обробка запитів, як-от зайві або часті виклики, може перевантажувати мережу і сповільнювати роботу додатка, також деякі незалежні виклики можна робити одночасно. Нижче наведено спосіб використання послідовних та паралельних запитів до серверу (див. рис. В.9).

Кешування зменшує кількість повторних обчислень і завантажень даних із мережі чи бази даних. Нижче наведено реалізацію відображення аватарок за допомогою використання звичайного віджету і віджету, який підтримує кешування даних (див. рис. В.10).

Дотримання цих рекомендацій дозволяє створювати швидкі, стабільні та енергоефективні додатки, які забезпечують позитивний досвід користувача.

4.6 Проведення експериментального дослідження обраних методів оптимізації рендерингу

Етап 1. Аналіз апріорної інформації.

Нехай під час аналізу апріорної інформації про програмне забезпечення стало відомо наступне:

- ПЗ є мобільним застосунком, написаним на мові програмування Dart за допомогою фреймфорку Flutter;
- ПЗ виконується на мобільному пристрої під керуванням операційної системи Android або iOS;
- ПЗ використовує Supabase у якості серверу.

Будемо використовувати математичну модель експеримента у вигляді лінійної моделі без залежностей між факторами. Задача зводиться до знаходження значень коефіцієнтів k_i рівняння регресії [27] (формула 4.5):

$$y = k_0 + k_1 x_1 + \cdots + k_n x_n \tag{4.5}$$

Фактори з найбільшими значеннями будуть найбільше впливати на вихідну характеристику.

Етап 2. Вибір факторів впливу.

Фактори впливу вибираються в результаті аналізу апріорної інформації про програмне забезпечення (див. табл. 4.5).

Таблиця 4.5 – Фактори впливу

Фактор впливу	Опис
x_1	Надмірна переробка віджетів
x_2	Кількість константних віджетів
x_3	Кількість Stateful віджетів

Кінець таблиці 4.5

Фактор впливу	Опис
x_4	Обсяг build-методів (у рядках)
x_5	Наявність helper-методів
x_6	Рендеринг усіх віджетів у дереві
x_7	Кількість віджетів з використанням Opacity
x_8	Використання isolates
x_9	Кількість паралельних запитів
x_{10}	Кешування картинок

Етап 3. Вибір верхнього і нижнього рівня для факторів.

Вибираємо верхній і нижній рівень для кожного фактора (див. табл. 4.6).

Таблиця 4.6 – Верхні та нижні значення факторів впливу.

Фактор впливу	Верхнє значення	Нижнє значення
x_1	on	off
x_2	20	4
x_3	20	2
x_4	120	30
x_5	on	off
x_6	on	off
x_7	20	0
x_8	on	off
x_9	20	0
x_{10}	on	off

Етап 4. Складання матриці планування та проведення експериментів

Y – час завантаження і рендерингу таблиці. Матрицю і результати екперименту наведено у таблиці 4.7.

Таблиця 4.7 – Матриця експериментів та результати

N	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	Y, c
1	1	1	1	1	1	1	1	1	1	1	5,45485113
2	1	1	1	1	1	1	1	1	1	-1	6,31701346
3	1	-1	1	1	1	-1	1	1	-1	1	3,63217529
4	1	1	1	1	1	1	1	1	-1	-1	5,09514963
5	1	-1	1	1	1	1	1	-1	1	1	9,58495677
6	1	1	1	1	1	1	1	-1	1	-1	8,78239709
7	1	1	1	1	1	1	1	-1	-1	1	8,076242
8	1	1	1	1	-1	1	1	-1	-1	-1	8,22757018
9	1	1	-1	1	1	1	-1	1	1	1	4,37131927
10	-1	1	1	1	1	1	-1	1	1	-1	5,61518392
11	-1	1	1	1	1	1	-1	1	-1	1	4,06133378
12	1	1	1	-1	1	1	-1	1	-1	-1	4,4824508
13	-1	1	-1	1	-1	1	-1	-1	1	1	7,04133949
14	1	1	1	1	1	1	-1	-1	1	-1	8,97639319
15	1	1	1	1	1	1	-1	-1	-1	1	8,82875376
16	1	1	1	1	1	1	-1	-1	-1	-1	8,14057991
17	1	1	1	1	1	-1	1	1	1	1	4,91950446
18	1	1	1	1	1	-1	1	1	1	-1	5,38298493
19	1	1	1	-1	1	-1	1	1	-1	1	4,13762554
20	-1	1	1	1	1	-1	1	1	-1	-1	4,02799735
21	1	1	1	1	1	-1	1	-1	1	1	8,33366667
22	1	1	1	1	1	-1	1	-1	1	-1	9,70059992
23	1	1	1	1	1	-1	1	-1	-1	1	7,25327812
24	1	1	1	1	1	-1	1	-1	-1	-1	8,78334562
25	1	1	-1	1	1	-1	-1	1	1	1	4,16704807
26	1	1	1	1	1	-1	-1	1	1	-1	4,38075271
27	1	1	1	1	1	-1	-1	1	-1	1	3,56604028
28	-1	1	1	1	1	-1	-1	1	-1	-1	4,1295623
29	-1	1	1	1	1	-1	-1	-1	1	1	8,38640894
30	1	1	1	1	1	-1	-1	-1	1	-1	9,66137418
31	1	1	1	1	1	-1	-1	-1	-1	1	7,51525943
32	-1	1	1	-1	1	-1	-1	-1	-1	-1	8,26538566

Етап 5. Аналіз результатів.

За допомогою аналізу даних у Microsoft Excel виконаємо регресійний аналіз (див. рис. 4.10).

SUMMARY OUTPUT								
Regression Statistics								
Multiple R	0,974760454							
R Square	0,950157943							
Adjusted R Square	0,932971027							
Standard Error	0,547145948							
Observations	40							
ANOVA								
	df	*SS*	*MS*	*F*	*Significance F*			
Regression	10	165,5023718	16,55023718	55,28379493	3,71564E-16			
Residual	29	8,681691966	0,299368688					
Total	39	174,1840638						
	Coefficients	*Standard Error*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*Lower 95,0%*	*Upper 95,0%*
Intercept	5,505615896	0,336548243	16,35906888	3,51331E-16	4,817297453	6,19393434	4,817297453	6,19393434
X Variable 1	0,077319531	0,128494983	0,601731909	0,55202822	-0,185482216	0,340121278	-0,185482216	0,340121278
X Variable 2	0,336867997	0,214011174	1,57406733	0,126319759	-0,100833999	0,774569993	-0,100833999	0,774569993
X Variable 3	0,445928093	0,201418678	2,213936156	0,034852876	0,033980643	0,857875542	0,033980643	0,857875542
X Variable 4	0,269740417	0,17713534	1,52279278	0,138641729	-0,092542031	0,632022865	-0,092542031	0,632022865
X Variable 5	0,273767726	0,13034176	2,10038384	0,044505649	0,007188895	0,540346557	0,007188895	0,540346557
X Variable 6	0,111107668	0,102727241	1,081579409	0,288351747	-0,098993129	0,321208466	-0,098993129	0,321208466
X Variable 7	0,157021809	0,118115601	1,329390927	0,194082819	-0,08455172	0,398595338	-0,08455172	0,398595338
X Variable 8	-1,912738248	0,088774063	-21,54613849	2,16153E-19	-2,094301592	-1,731174903	-2,094301592	-1,731174903
X Variable 9	0,486456715	0,095395103	5,099388726	1,92428E-05	0,291351822	0,681561608	0,291351822	0,681561608
X Variable 10	-0,122151305	0,095078101	-1,284746995	0,209048371	-0,316607855	0,072305246	-0,316607855	0,072305246

Рисунок 4.10 – Регресійний аналіз результатів дослідження (рисунок виконано самостійно)

Основні висновки:

– Multiple R: 0.9748 – сильна кореляція між незалежними змінними та залежною змінною;
– R Square: 0.9502 – модель пояснює 95.02% варіації залежної змінної;
– Adjusted R Square: 0.9329 – скоригований коефіцієнт враховує кількість змінних і обсяг вибірки;
– Significance F: $3.715 * 10^{-16}$ (дуже низьке значення) – модель є статистично значущою.

Коефіцієнти рівняння регресії можна знайти як (формула 4.6):

$$k_j = \frac{\sum_{i=1}^{N} x_{ij} y_i}{N}, j = 1k \tag{4.5}$$

Таблиця 4.8 – Коефіцієнти рівняння регресії

k_0	5,505615896
k_1	0,077319531
k_2	0,336867997
k_3	0,445928093

Кінець таблиці 4.8

k_4	0,269740417
k_5	0,273767726
k_6	0,111107668
k_7	0,157021809
k_8	-1,912738248
k_9	0,486456715
k_{10}	-0,122151305

Рівняння регресії:

$$Y = 5{,}505615896 + 0{,}077319531x_1 + 0{,}336867997x_2 + 0{,}445928093x_3 + 0{,}269740417x_4 + 0{,}273767726x_5 + 0{,}111107668x_6 + 0{,}157021809x_7 - 1{,}912738248x_8 + 0{,}486456715x_9 - 0{,}122151305x_{10}$$

Візуалізацію результатів прогнозування наведено на рисунку 4.11.

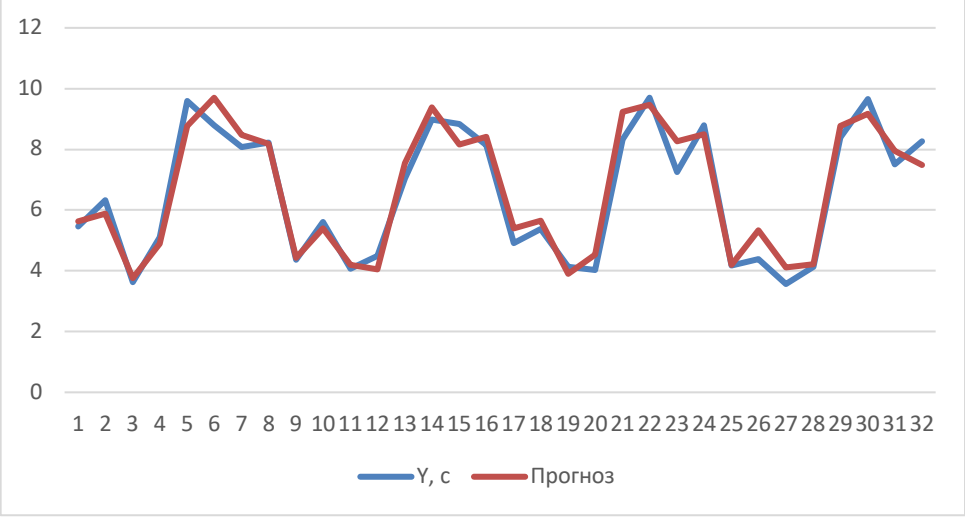

Рисунок 4.11 – Результати прогнозування (рисунок виконано самостійно)

Аналіз значущості змінних (P-value).

Статистично значущі змінні (P-value < 0.05):

x_5 (P $=$ 0.0445);

x_6 (P $=$ $2.09 * 10^{-5}$):

Ці змінні мають значний вплив на залежну змінну Y.

Змінні з високими P-value (P-value > 0.05):

$$x_1, x_2, x_3, x_4, x_6, x_7, x_{10}.$$

Їх вплив менш значущий, і їх можна розглянути для виключення з моделі.

Змінна x_8: (P $= 2.16 * 10^{-19}$) – високий вплив, але коефіцієнт негативний (-1.9127), що свідчить про сильне зниження Y зі збільшенням x_8.

На основі наданих факторів впливу та результатів регресійного аналізу, можна інтерпретувати значення змінних і їхній вплив на залежну змінну Y [28]. Нижче наведено детальніший аналіз:

Ключові фактори впливу:

а) x_5 (наявність helper-методів):

1. коефіцієнт: 0.2737, P-value: 0.0445;
2. висновок: наявність helper-методів негативно впливає на залежну змінну; вплив значущий, тому не рекомендується використовувати helper-методи для оптимізації;

б) x_8 (використання isolates):

1. коефіцієнт: −1.9127, P-value: дуже низьке (<0.001);
2. висновок: використання isolates має дуже сильний позитивний вплив, що дозволяють зменшити час рендерингу;

в) x_9 (кількість паралельних запитів):

1. коефіцієнт: 0.48650, P-value: 0.0005;
2. висновок: збільшення кількості паралельних запитів суттєво й позитивно впливає на результат; це може означати, що паралельність оптимізує час виконання задач.

Фактори без значного впливу:

- x_1 (надмірна переробка віджетів): не має статистично значущого впливу (P=0.5520);
- x_2 (кількість константних віджетів): не значущий (P=0.1263);
- x_3 (кількість Stateful віджетів): не значущий (P=0.8579);
- x_4 (обсяг build-методів): не значущий (P=0.1386);
- x_6 (рендеринг усіх віджетів у дереві): не значущий (P=0.3210);
- x_7 (кількість віджетів із використанням Opacity): не значущий (P=0.1941);
- x_{10} (кешування картинок): не значущий (P=0.2090).

У процесі оптимізації часу рендерингу у Flutter-додатках найсильніший вплив мають змінні x_8 (використання isolates) та x_9 (кількість паралельних запитів). Обидві змінні демонструють статистично значущий позитивний вплив на залежну змінну Y, що свідчить про ефективність застосування паралелізму та розподілу обчислень при побудові високопродуктивних інтерфейсів.

Також змінна x_5 (наявність helper-методів) виявилася значущою, проте її вплив негативний, тобто часте використання таких методів може збільшувати час рендерингу, можливо через додаткові витрати на виклики функцій або ускладнення логіки.

Натомість змінні $x_1, x_2, x_3, x_4, x_6, x_7, x_{10}$ мають високі P-value, що вказує на відсутність статистично значущого впливу на результат. Це означає, що в рамках цієї моделі та вибірки даних їхній внесок у зміну часу рендерингу є мінімальним або випадковим. Їх можна розглядати як кандидатів на виключення з моделі для її спрощення без втрати точності.

Таким чином, для досягнення ефективної оптимізації доцільно зосередити увагу на використанні isolates та паралельних запитів, а також уникати надмірного використання helper-методів у критичних ділянках побудови інтерфейсу.

5 ПРОГРАМНА РЕАЛІЗАЦІЯ

Після першого запуску застосунку Synergy Scanner користувач потрапляє на сторінку входу (див. рис. 5.1). Тема оформлення спочатку відповідає системним налаштуванням пристрою. У застосунку передбачено дві мови: англійську та українську. За замовчуванням вибирається мова системи, а якщо вона недоступна, використовується англійська.

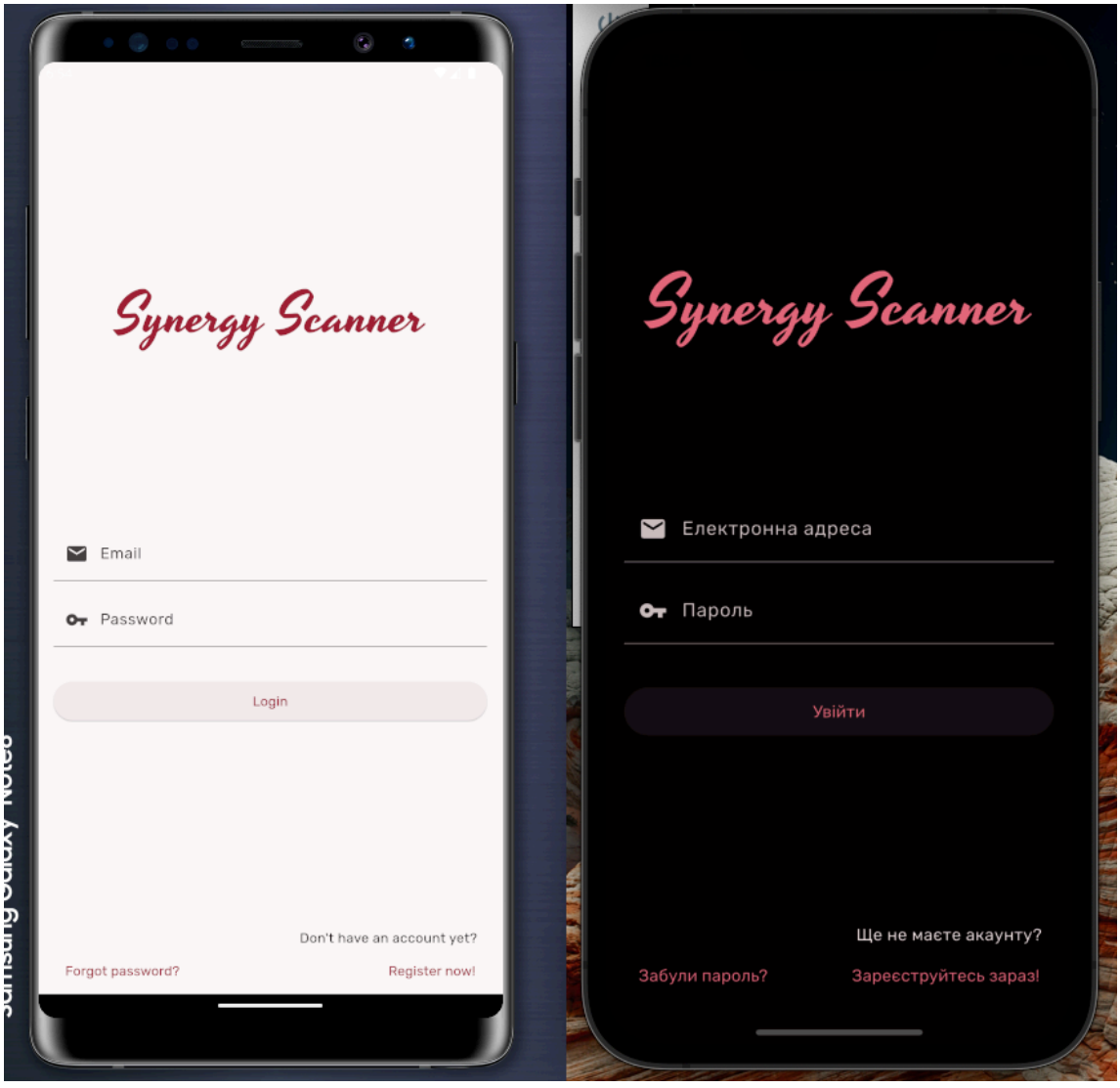

Рисунок 5.1 – Сторінка логіну (рисунок виконано самостійно)

Якщо користувач забув пароль, він може перейти на сторінку відновлення доступу (див. рис. 5.2). Там необхідно ввести адресу електронної пошти, на яку буде надіслано тимчасовий пароль. Після цього користувач повинен змінити його на новий.

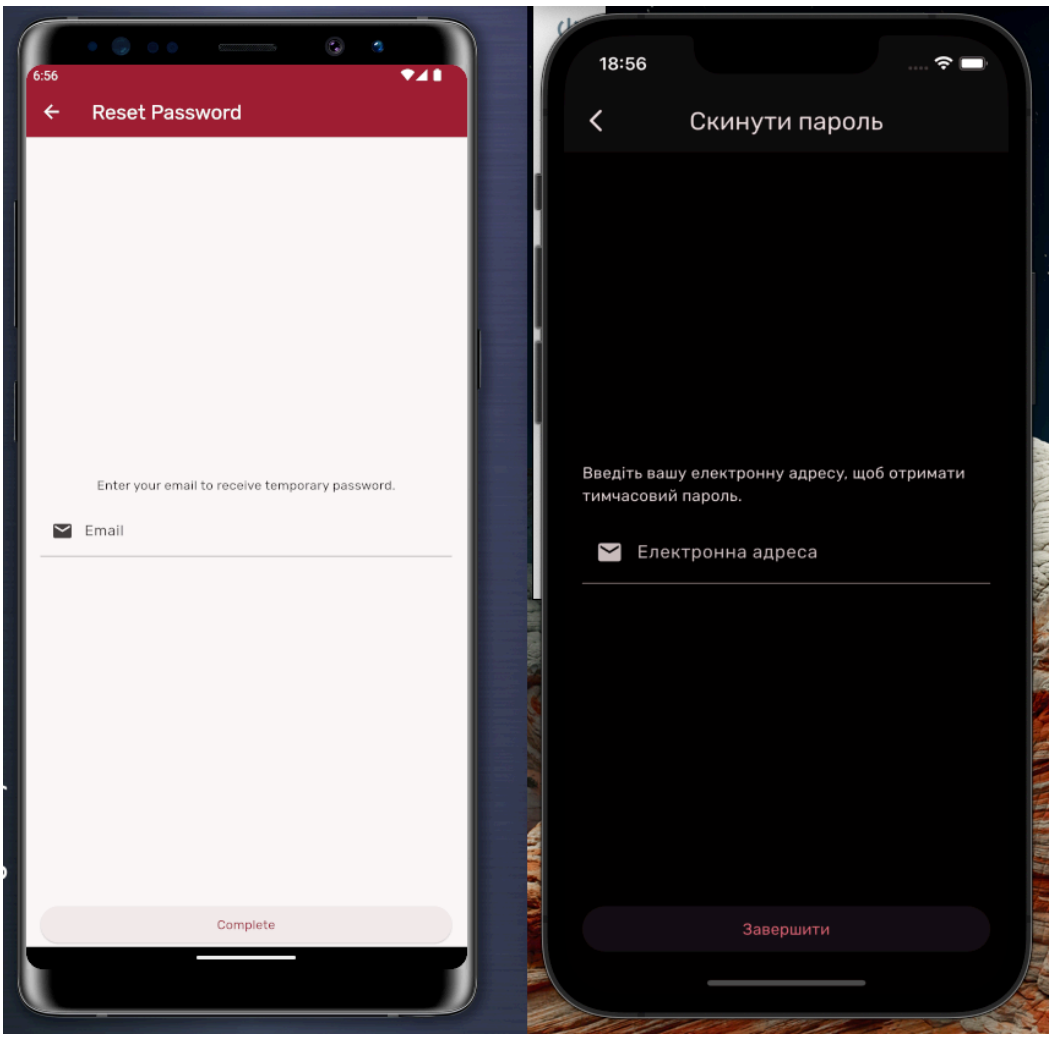

Рисунок 5.2 – Сторінка відновлення паролю (рисунок виконано самостійно)

Якщо користувач не має акаунту, він може створити його на сторінці реєстрації (див. рис. 5.3). Для цього потрібно вказати ім’я, електронну пошту та пароль, які будуть використовуватися для подальшої автентифікації в системі. Пароль повинен відповідати вимогам безпеки, зокрема містити мінімальну кількість символів та включати комбінацію літер і цифр.

Крім обов’язкових полів, користувач має можливість додати власне фото профілю, що сприяє персоналізації облікового запису. Завантаження фото є необов’язковим і може бути пропущене або здійснене пізніше через розділ редагування профілю.

Після заповнення всіх необхідних даних та підтвердження форми реєстрації, система надсилає лист підтвердження на вказану електронну пошту. Це дозволяє перевірити достовірність адреси та запобігти реєстрації фіктивних

або несанкціонованих облікових записів. Лише після активації через посилання в електронному листі користувач отримує повний доступ до функціональності системи.

Таким чином, процес реєстрації поєднує зручність для користувача та базові вимоги безпеки, забезпечуючи надійну ідентифікацію та контроль доступу до сервісу.

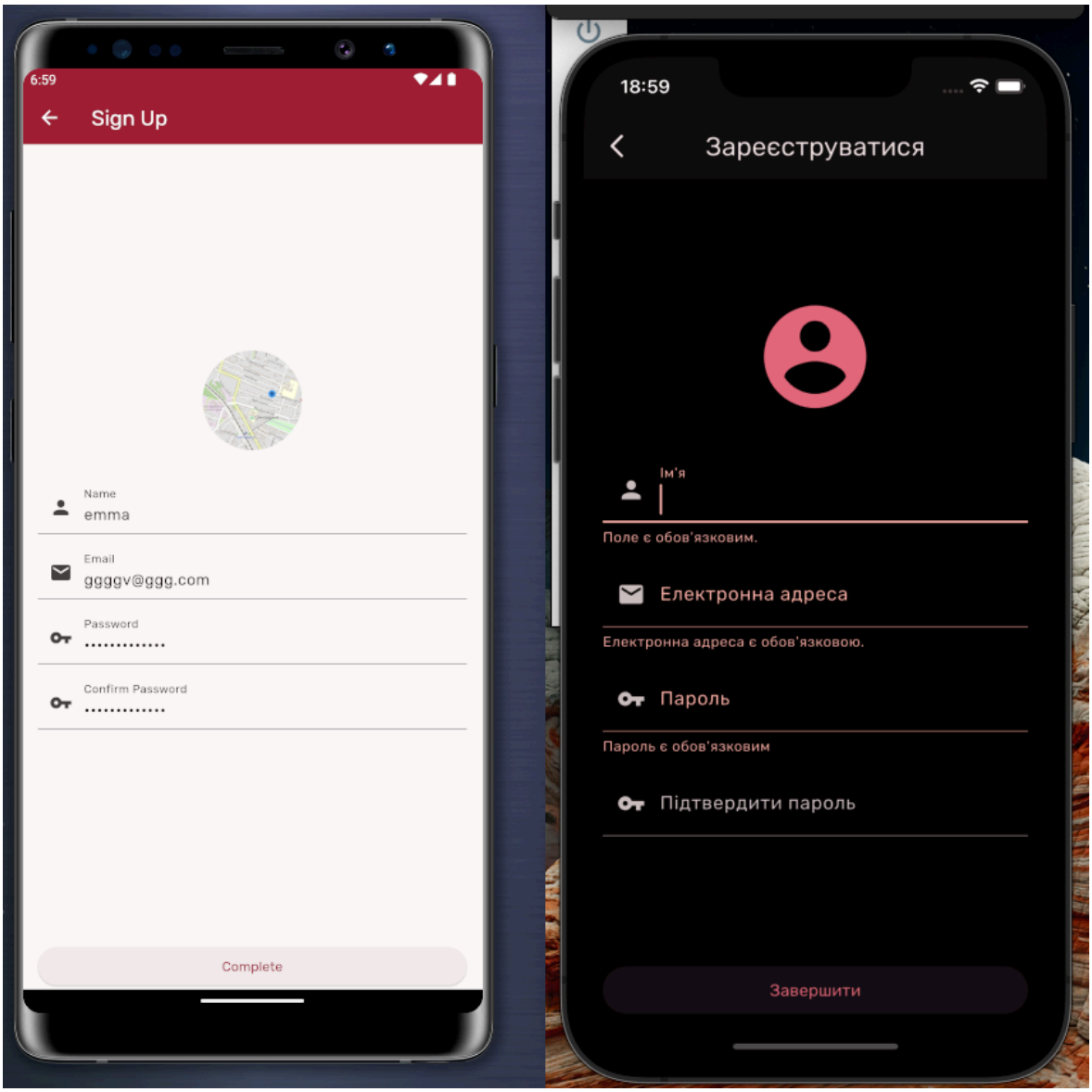

Рисунок 5.3 – Сторінка реєстрації (рисунок виконано самостійно)

Після авторизації користувач потрапляє на головну сторінку, яка містить чотири вкладки. Перша – вкладка з партнерами (див. рис. 5.4). Тут можна переглядати доданих парнерів, переходити до їхніх деталей, а також здійснювати пошук за назвою.

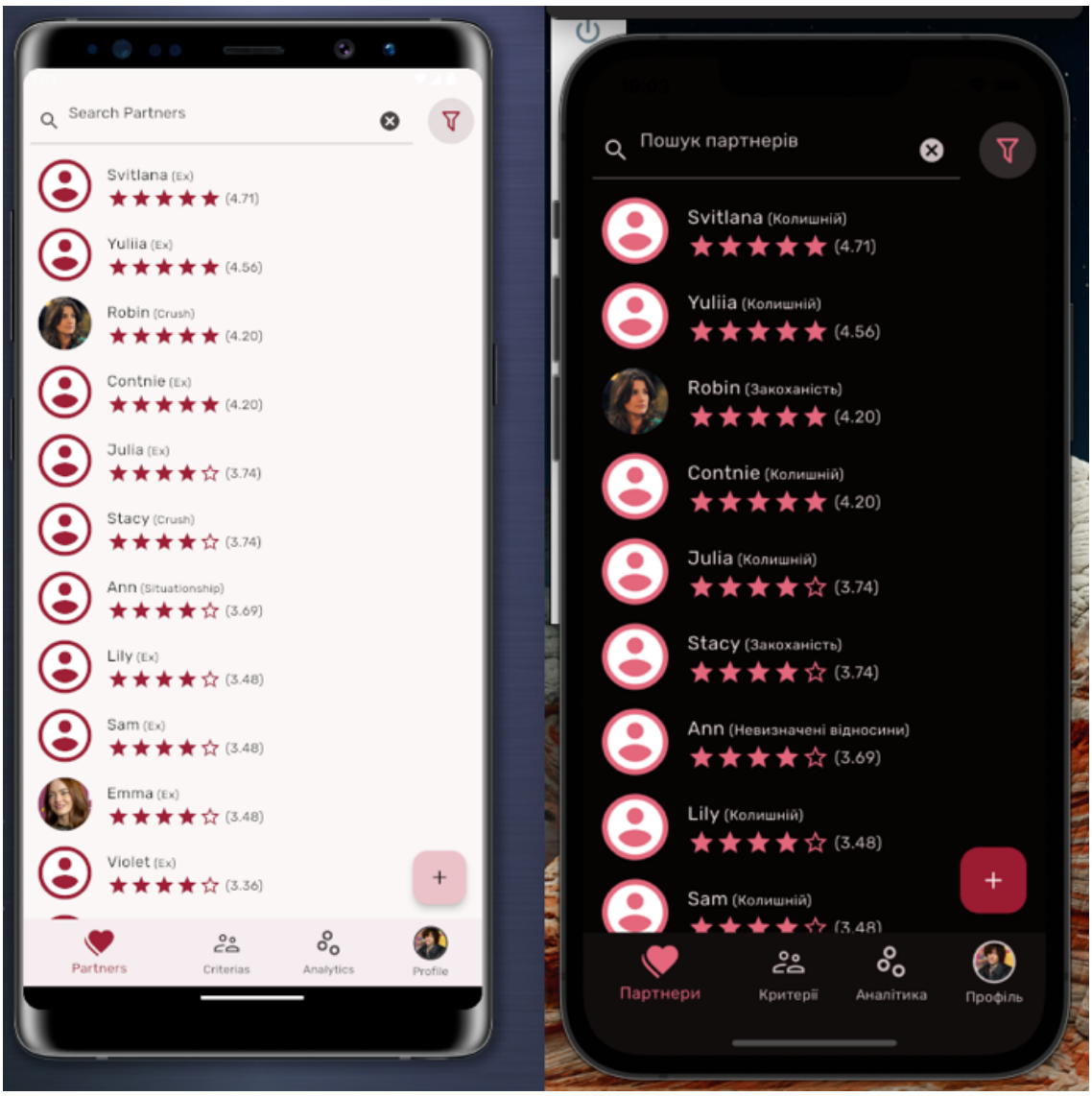

Рисунок 5.4 – Вкладка «Партнери» (рисунок виконано самостійно)

Натиснувши кнопку «Фільтр» на вкладці з партнерами, користувач переходить на відповідну сторінку фільтрації (див. рис. 5.5), яка надає гнучкі інструменти для сортування та відбору партнерів згідно з індивідуальними вподобаннями або критеріями пошуку.

На цій сторінці доступна функція сортування, яка дозволяє впорядкувати список партнерів за такими параметрами, як:

– середня оцінка – дозволяє вивести на початок списку найвищооцінених користувачів або навпаки;
– ім’я – сортування за алфавітом (від A до Z або Z до A);
– тип партнера.

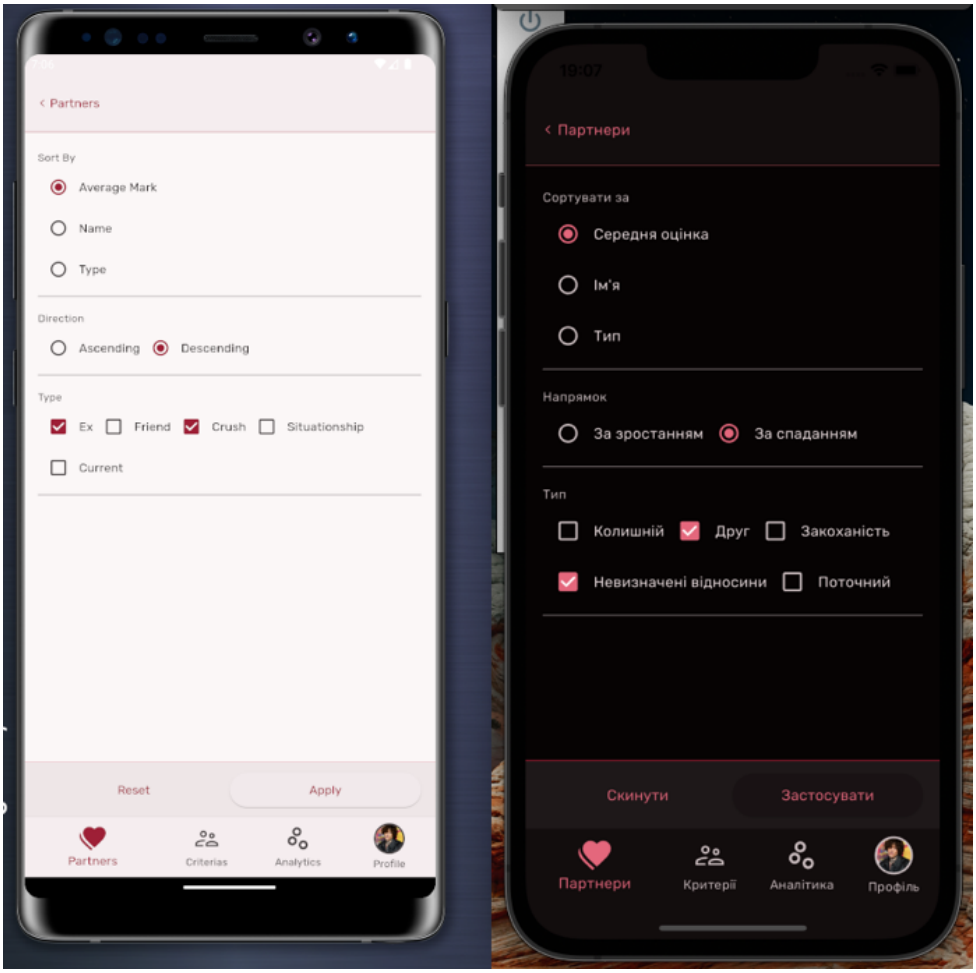

Рисунок 5.5 – Сторінка фільтрів партнерів (рисунок виконано самостійно)

Для кожного критерію сортування користувач може вказати напрямок сортування: за зростанням (в порядку зростання значень) або за спаданням.

Крім сортування, доступна також функція фільтрації, яка дозволяє обмежити список партнерів лише тими, що належать до певного типу.

Інтерфейс сторінки реалізований інтуїтивно зрозумілим способом: фільтри представлені у вигляді випадаючих списків та перемикачів, що робить процес налаштування зручним навіть для недосвідчених користувачів. Після застосування обраних параметрів результати оновлюються автоматично, що дозволяє миттєво оцінити ефективність фільтрації.

Таким чином, функціонал сортування і фільтрації значно покращує користувацький досвід, дозволяючи швидко орієнтуватися у великій кількості доступних партнерів і знаходити найбільш релевантних відповідно до потреб користувача.

Натиснувши на кнопку «+» на вкладці «Партнери» (див. рис. 5.6), користувач опиняється на сторінці додавання. Тут можна додати партнера, вказавши його ім’я та тип, а також додати фото (необов’язкове).

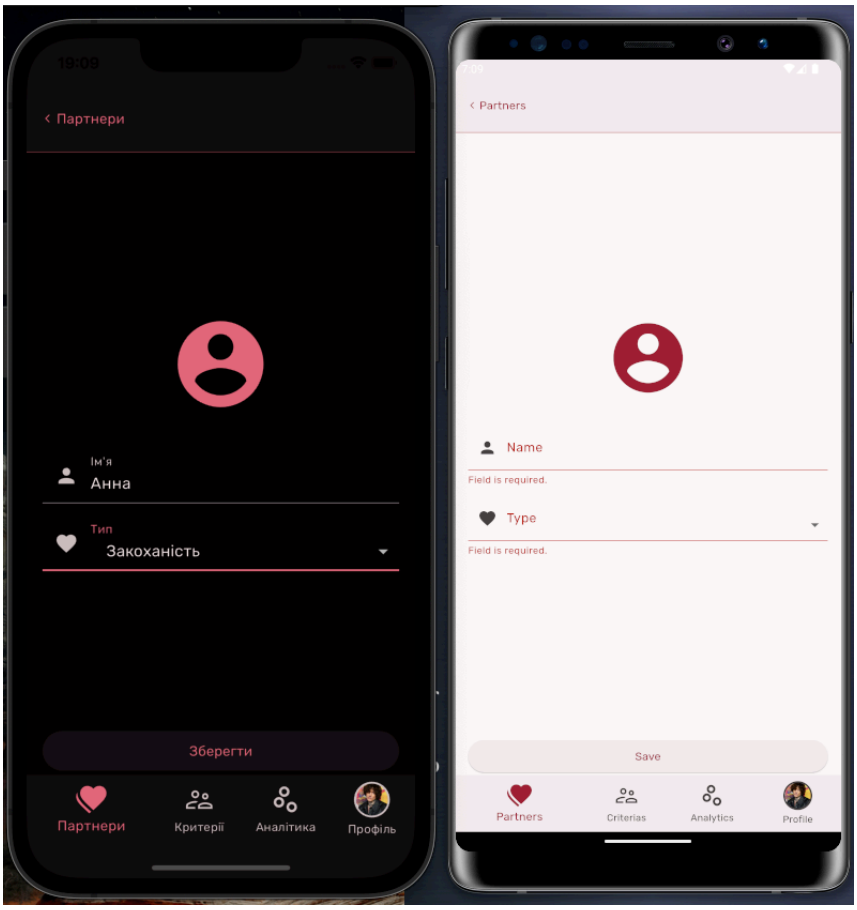

Рисунок 5.6 – Сторінка додавання партнера (рисунок виконано самостійно)

Натиснувши на картку з партнером, користувач переходить на сторінку з деталями партнера (див. рис. 5.7). Ця сторінка надає розгорнуту інформацію про обраного партнера та дозволяє оцінити його за різними критеріями, що формують загальне враження та допомагають іншим користувачам приймати зважені рішення.

На сторінці відображаються такі елементи:

– ім’я партнера та його основна інформація (тип, фото, короткий опис або спеціалізація);
– список критеріїв оцінювання, що охоплюють ключові аспекти взаємодії (наприклад, комунікабельність, технічна обізнаність, пунктуальність тощо);

– поточні оцінки, виставлені іншими користувачами, для кожного з критеріїв;
– можливість власного оцінювання – користувач може обрати бажані критерії та виставити свою оцінку (наприклад, за шкалою від 1 до 5).

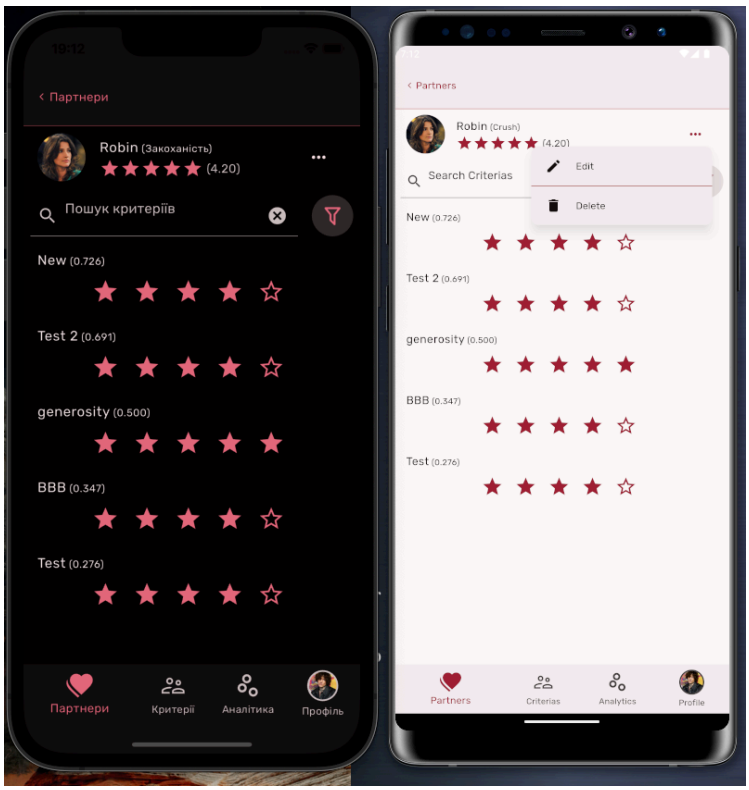

Рисунок 5.7 – Сторінка деталей партнера (рисунок виконано самостійно)

Крім того, доступна функція пошуку критеріїв за назвою, що значно спрощує навігацію, особливо коли критеріїв багато. Пошук працює в режимі реального часу – достатньо ввести частину назви, і список автоматично фільтрується.

Інтерфейс сторінки розроблений з урахуванням зручності: усі дії – від перегляду до оцінювання – виконуються швидко, без необхідності переходити на інші екрани або перезавантажувати сторінку. Завдяки цьому взаємодія з платформою стає ефективнішою та інтуїтивно зрозумілою.

Таким чином, детальна сторінка партнера виконує не лише інформаційну функцію, а й дозволяє формувати якісні відгуки, сприяючи побудові прозорої та об’єктивної системи оцінювання всередині застосунку.

Натиснувши на кнопку «Редагувати», користувач переходить на сторінку редагування партнера (див. рис. 5.8). Ця сторінка за своїм виглядом і функціональністю є майже ідентичною до сторінки створення нового партнера, що забезпечує послідовний користувацький досвід і спрощує процес взаємодії з інтерфейсом.

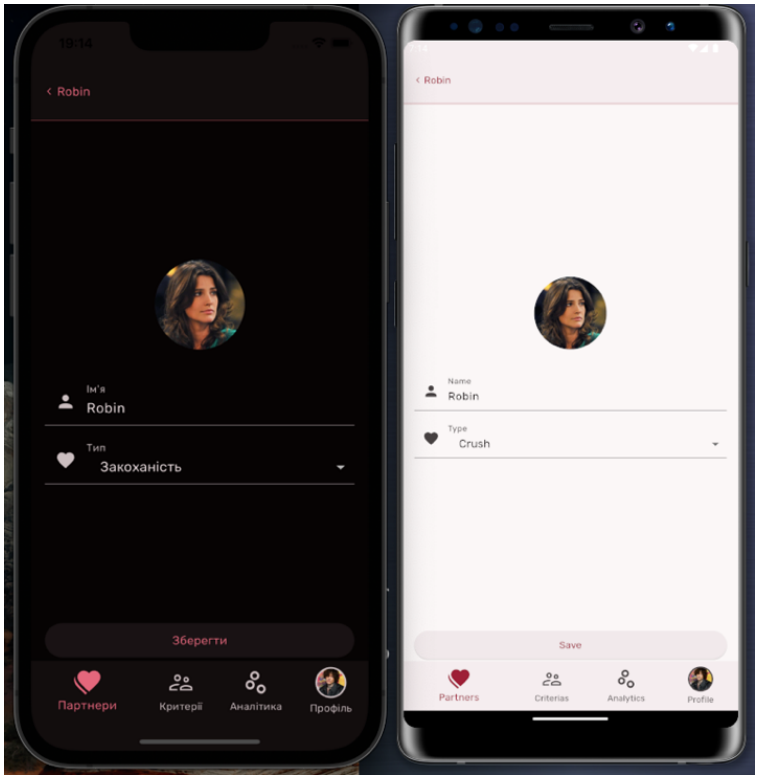

Рисунок 5.8 – Сторінка редагування партнера (рисунок виконано самостійно)

На сторінці редагування користувач може:

- змінити ім’я партнера, ввівши нове значення у відповідне текстове поле;
- завантажити або змінити фото профілю, обравши зображення з пристрою (це дозволяє оновлювати візуальну інформацію про партнера відповідно до потреб або змін у його статусі);
- обрати або змінити тип партнера зі списку доступних варіантів (наприклад, ментор, співрозмовник, наставник тощо). Це поле реалізоване у вигляді випадаючого списку для зручності вибору.

Після внесення змін користувач може зберегти оновлення, натиснувши відповідну кнопку. Усі зміни миттєво відображаються в системі.

Якщо користувач натисне на кнопку видалення, то з’явиться діалог, у якому потрібно підтвердити свою дію (див. рис. 5.9).

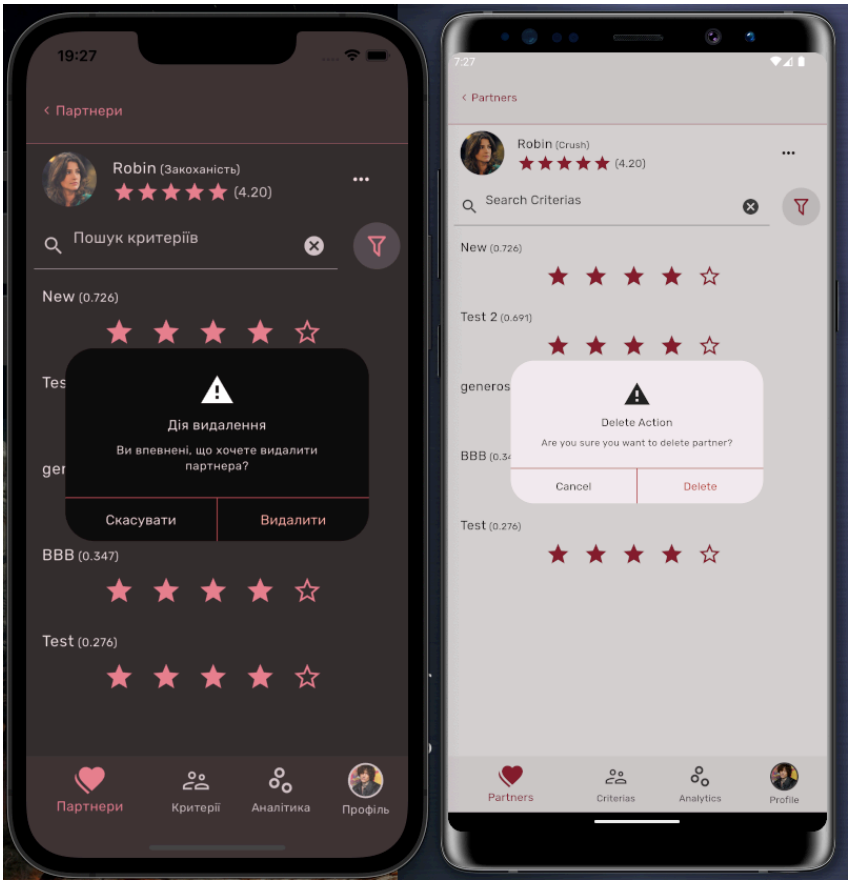

Рисунок 5.9 – Діалог підтвердження видалення партнера (рисунок виконано самостійно)

Якщо натиснути на кнопку фільтрів, відкриється спеціальне вікно сортування та фільтрації критеріїв (див. рис. 5.10), яке надає користувачеві розширені можливості для організації відображення критеріїв відповідно до конкретних потреб або вподобань.

У цьому вікні можна:

– відсортувати критерії за назвою – в алфавітному порядку (А–Я або Я-А), що зручно при пошуку конкретних критеріїв у великому списку;
– сортувати за коефіцієнтом – від найменш значущих до найбільш значущих або навпаки, це дає змогу визначити, які з критеріїв мають найбільший вплив у загальній оцінці;

– сортувати за оцінкою – за середнім значенням, виставленим іншими користувачами, що дозволяє легко виділити найбільш високо або низько оцінені аспекти взаємодії з партнером.

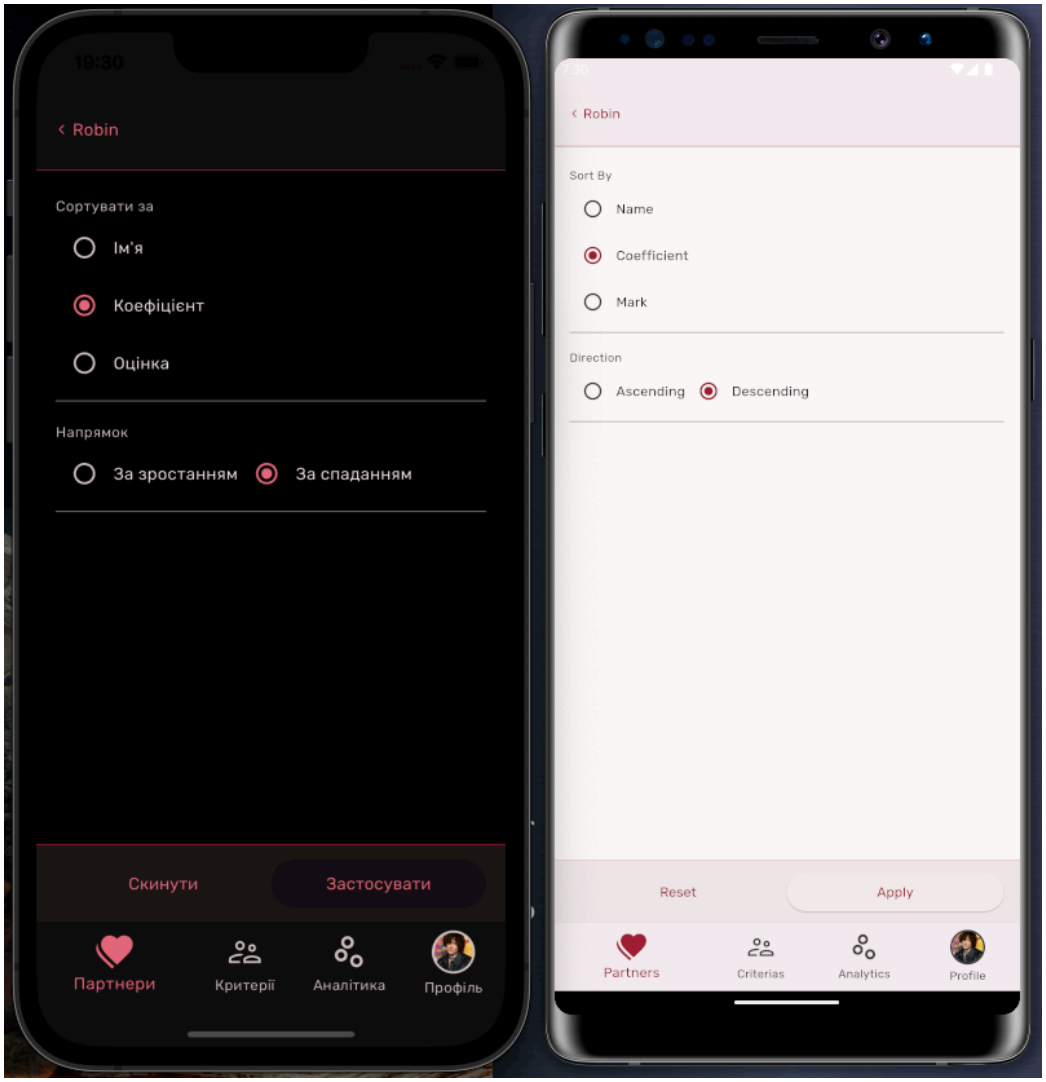

Рисунок 5.10 – Сторінка фільтрації критеріїв партнера (рисунок виконано самостійно)

Для кожного з параметрів сортування користувач також може вказати напрямок сортування – за зростанням або за спаданням – за допомогою інтуїтивно зрозумілих перемикачів або стрілок.

Інтерфейс вікна фільтрації побудований таким чином, щоб забезпечити максимальну зручність і швидкість взаємодії. Обрані параметри застосовуються одразу після підтвердження, оновлюючи список критеріїв без потреби перезавантаження сторінки.

Таким чином, ця функціональність допомагає користувачам ефективно аналізувати та порівнювати критерії, забезпечуючи глибше розуміння якості взаємодії з партнерами та сприяючи прийняттю обґрунтованих рішень.

Другою вкладкою на головній сторінці є «Критерії» (див. рис. 5.11). Тут користувач може переглядати всі створені критерії, шукати їх за назвою, переходити до деталей, а також перейти на сторінку створення нового критерію.

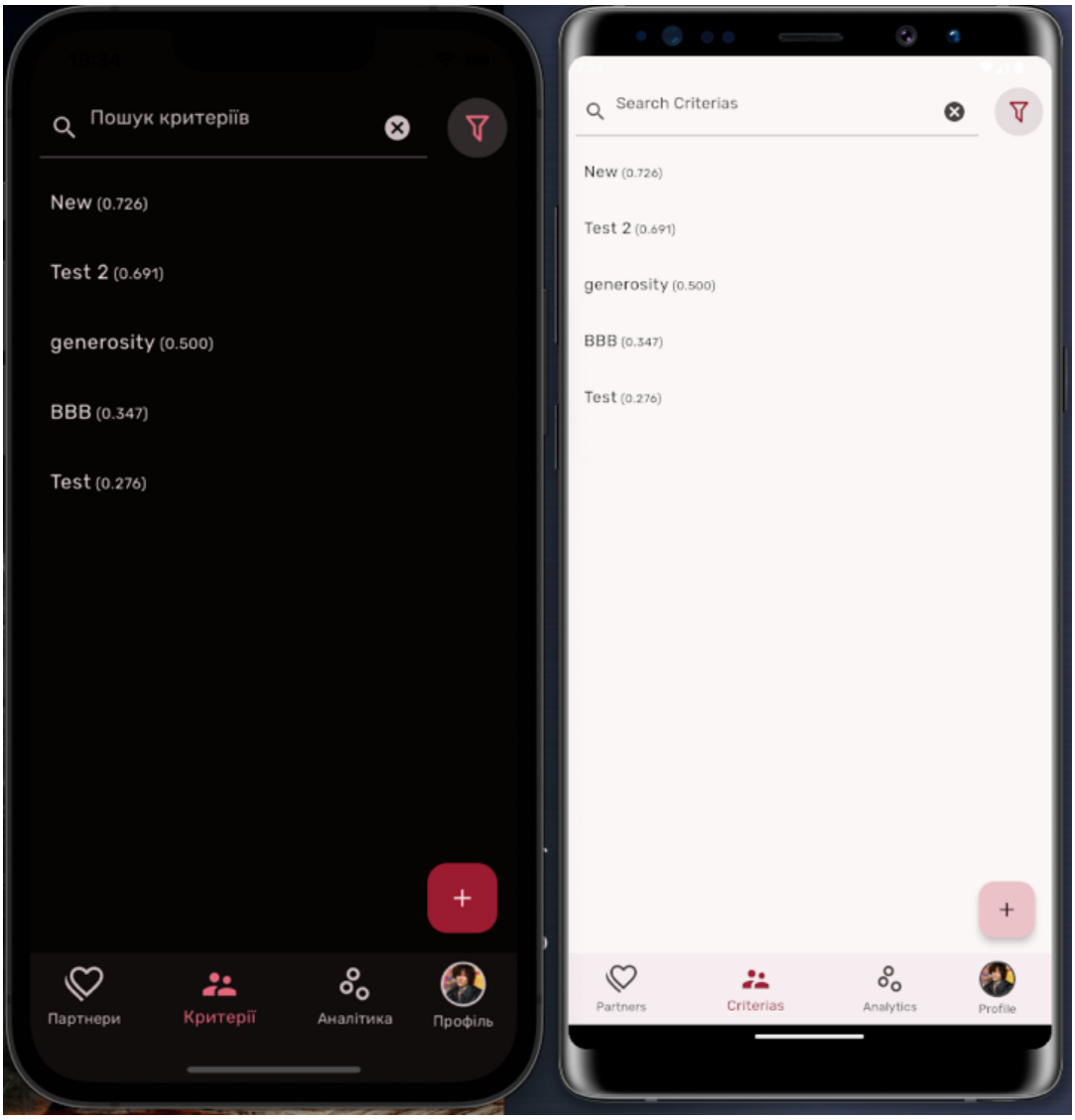

Рисунок 5.11 – Вкладка «Критерії» (рисунок виконано самостійно)

Натиснувши кнопку «Фільтр», користувач переходить на сторінку фільтрації критеріїв (див. рис. 5.12). Ця сторінка призначена для зручного впорядкування та перегляду наявних критеріїв оцінювання, що дозволяє швидко знайти потрібні або найважливіші з них.

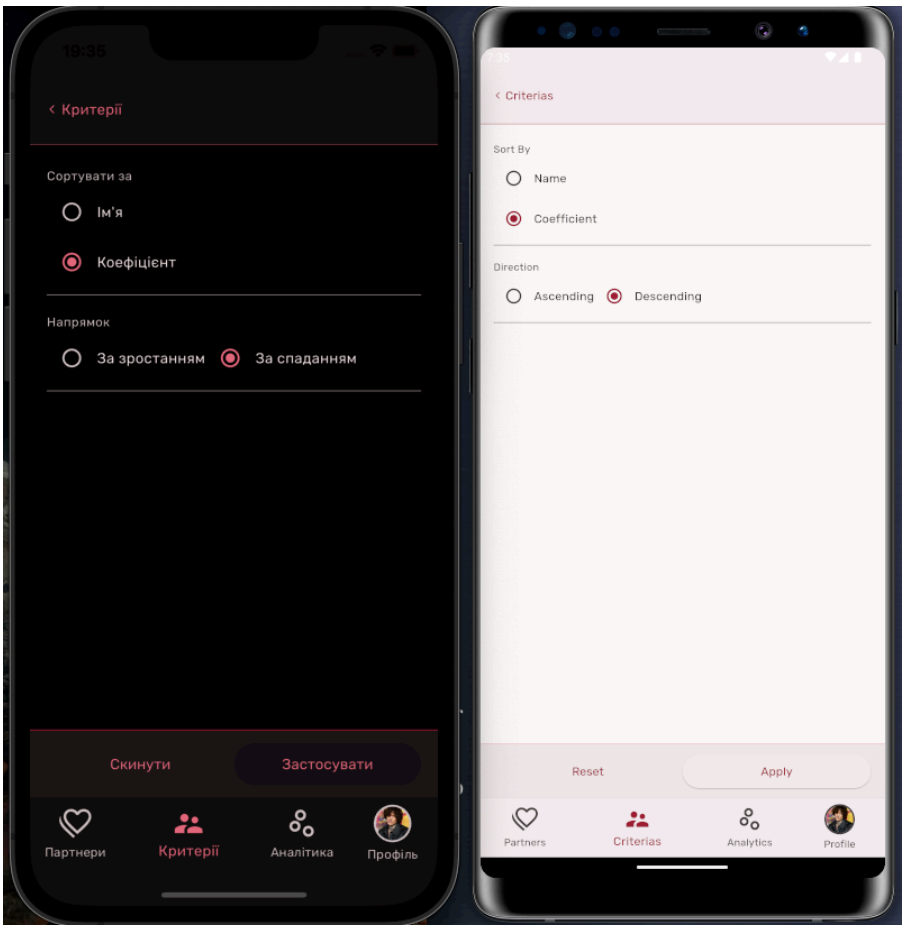

Рисунок 5.12 – Сторінка фільтрів критеріїв (рисунок виконано самостійно)

На сторінці доступні наступні можливості сортування:

- за назвою – дозволяє впорядкувати критерії в алфавітному порядку (А–Я або Я–А), що особливо корисно при великій кількості критеріїв;
- за коефіцієнтом – критерії можуть бути відсортовані за їхньою вагомістю або впливом на загальну оцінку, що допомагає визначити найбільш значущі параметри.

Для кожного типу сортування можна окремо вказати напрямок сортування – за зростанням або за спаданням. Це здійснюється за допомогою зрозумілого елемента керування, наприклад перемикача або стрілки.

Інтерфейс реалізовано таким чином, щоб забезпечити швидкий і зручний доступ до необхідної інформації без зайвих дій. Усі зміни відображаються динамічно, без потреби перезавантаження сторінки, що покращує загальний користувацький досвід.

Таким чином, сторінка фільтрації критеріїв забезпечує користувачу інструменти для ефективного аналізу, полегшуючи навігацію та вибір найбільш релевантних параметрів для подальшої оцінки або перегляду.

Якщо натиснути на кнопку «+», то відкриється сторінка додавання критерію (див. рис. 5.13). Необхідно вказати назву та коефіцієнт, обидва поля обов’язкові.

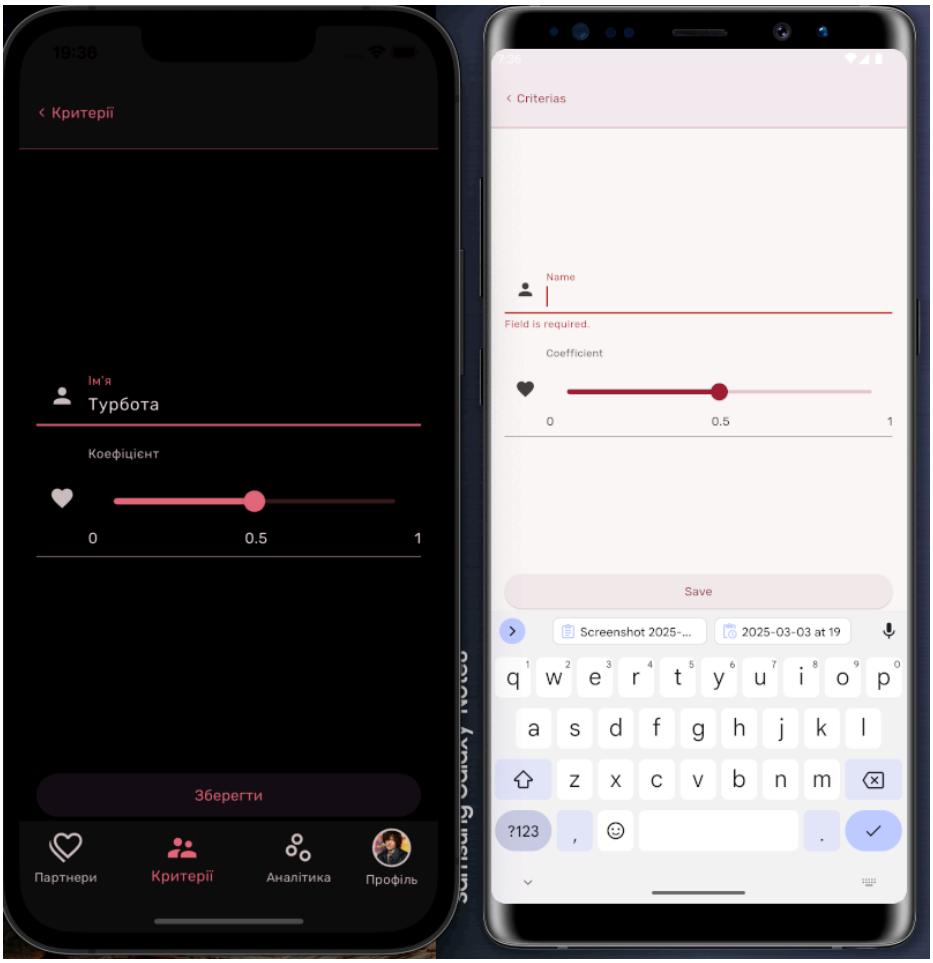

Рисунок 5.13 – Сторінка створення критерію (рисунок виконано самостійно)

Натиснувши на картку з критерієм, користувач переходить на сторінку з деталями критерію (див. рис. 5.14). Ця сторінка містить розгорнуту інформацію про обраний критерій оцінювання та дозволяє взаємодіяти з ним у контексті конкретних партнерів.

На сторінці відображаються:

– назва критерію та його короткий опис (за наявності), що допомагає краще зрозуміти, які саме аспекти взаємодії він охоплює;

– список партнерів, які вже були оцінені за цим критерієм. Поруч з кожним партнером показується поточна оцінка, виставлена користувачами;
– можливість поставити власну оцінку – користувач може самостійно оцінити кожного з партнерів за даним критерієм, що дозволяє сформувати більш об’єктивну та збалансовану картину.

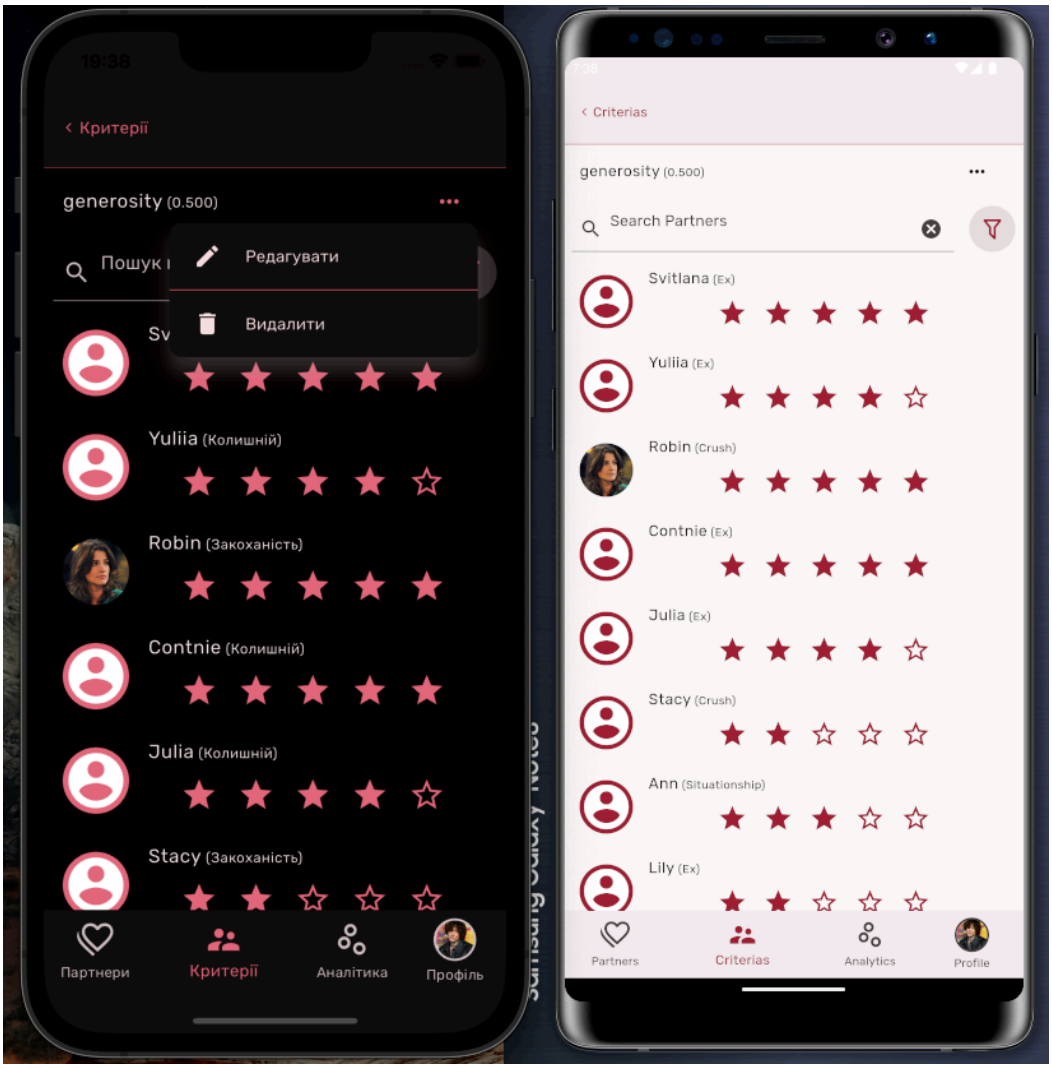

Рисунок 5.14 – Сторінка деталей критерію (рисунок виконано самостійно)

Також реалізовано пошук партнерів за ім’ям, що значно спрощує навігацію, особливо коли кількість оцінених партнерів велика. Пошук працює миттєво: результати оновлюються в реальному часі під час введення тексту.

Інтерфейс побудований таким чином, щоб забезпечити максимальну зручність у виставленні та перегляді оцінок: усі елементи розміщені логічно та інтуїтивно, а сам процес оцінювання займає мінімум часу.

Таким чином, сторінка деталізації критерію слугує важливим інструментом для глибшого аналізу якості партнерських взаємодій та активного формування рейтингової системи на платформі.

Натиснувши на кнопку «Редагувати», користувач переходить на сторінку редагування критерію (див. рис. 5.15). Ця сторінка за зовнішнім виглядом і функціоналом дуже схожа на сторінку створення нового критерію, що дозволяє забезпечити послідовність інтерфейсу та зручність користування. На сторінці редагування користувач має можливість змінити назву існуючого критерію, що допомагає більш точно відображати його зміст або призначення.

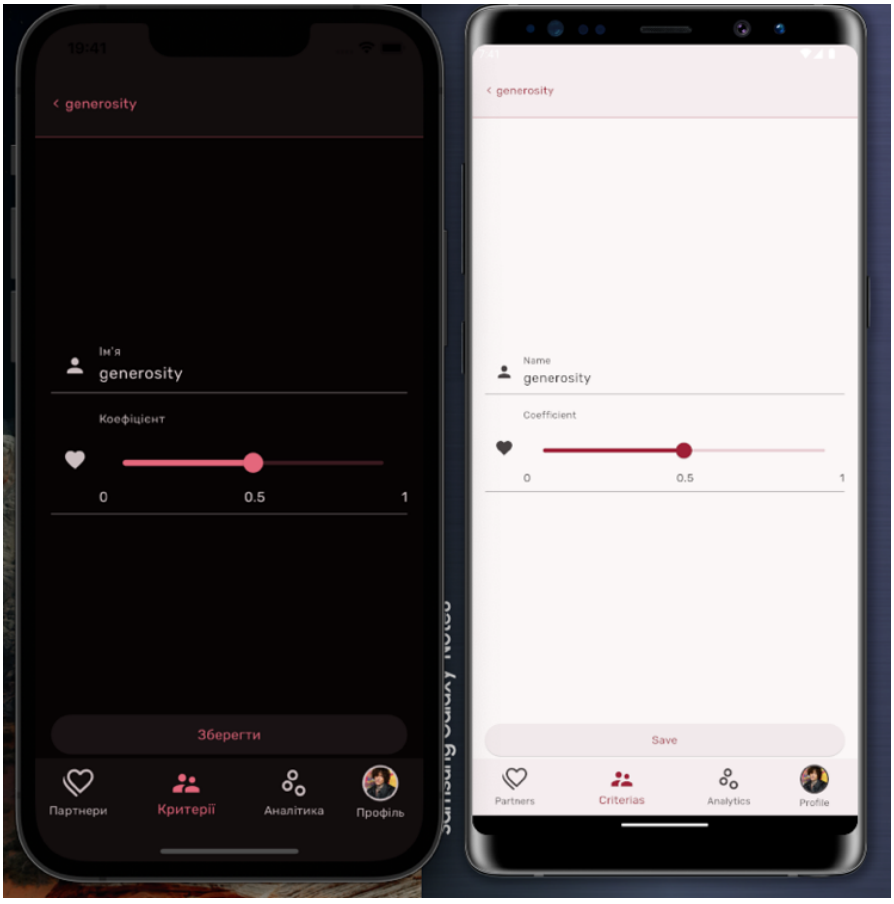

Рисунок 5.15 – Сторінка редагування критерію (рисунок виконано самостійно)

Окрім назви, можна також змінити коефіцієнт критерію – числове значення, яке використовується для визначення ваги або важливості цього критерію у подальших обчисленнях чи оцінках. Інтерфейс сторінки містить відповідні поля введення для обох параметрів, а також кнопки для підтвердження змін або скасування редагування.

Після внесення необхідних змін користувач може зберегти оновлені дані, натиснувши кнопку «Зберегти», після чого система оновить інформацію про критерій у базі даних. Якщо користувач бажає відмінити редагування, він може натиснути кнопку «Скасувати», і повернеться до попереднього екрану без збереження змін.

Таким чином, сторінка редагування критерію дозволяє швидко та зручно вносити необхідні корективи до існуючих параметрів, забезпечуючи гнучкість і точність у роботі з критеріями.

Якщо користувач натисне на кнопку видалення, то з’явиться діалог, у якому потрібно підтвердити свою дію (див. рис. 5.16).

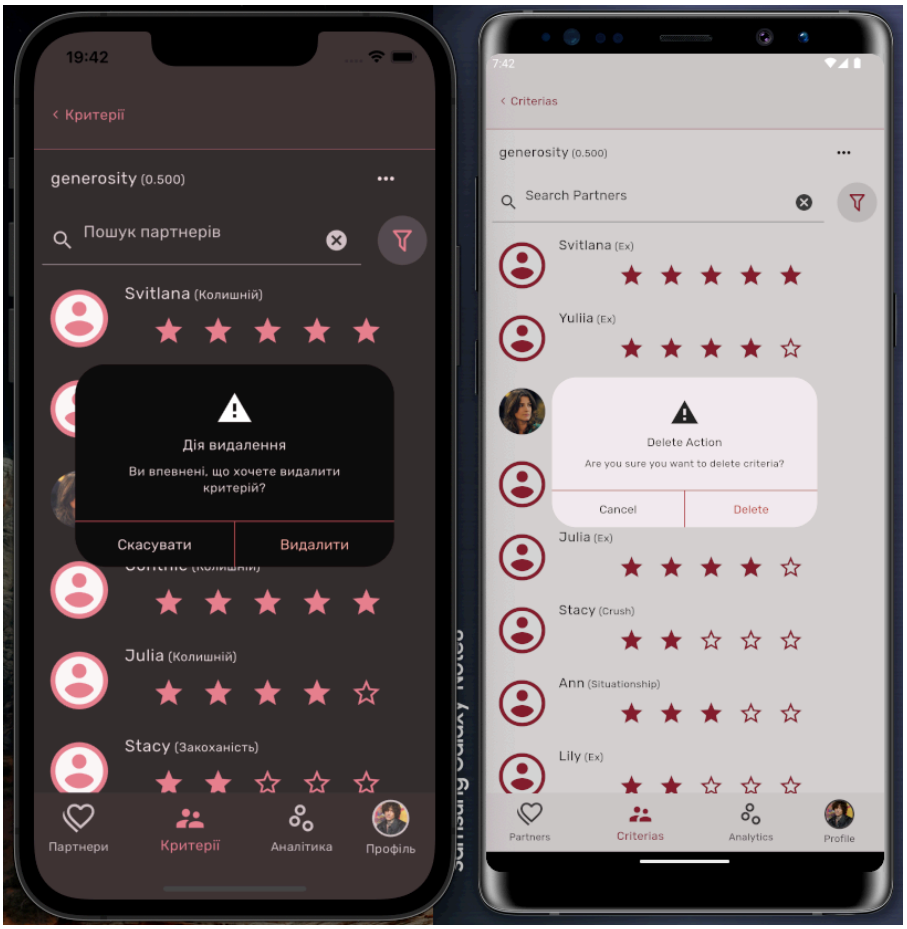

Рисунок 5.16 – Підтвердження видалення критерію (рисунок виконано самостійно)

Якщо натиснути на кнопку фільтрів, відкриється спеціальне вікно з можливістю налаштування сортування та фільтрації партнерів (див. рис. 5.17). У

цьому вікні користувач може вибрати параметр для сортування списку партнерів за такими критеріями, як ім’я, тип партнера або оцінка.

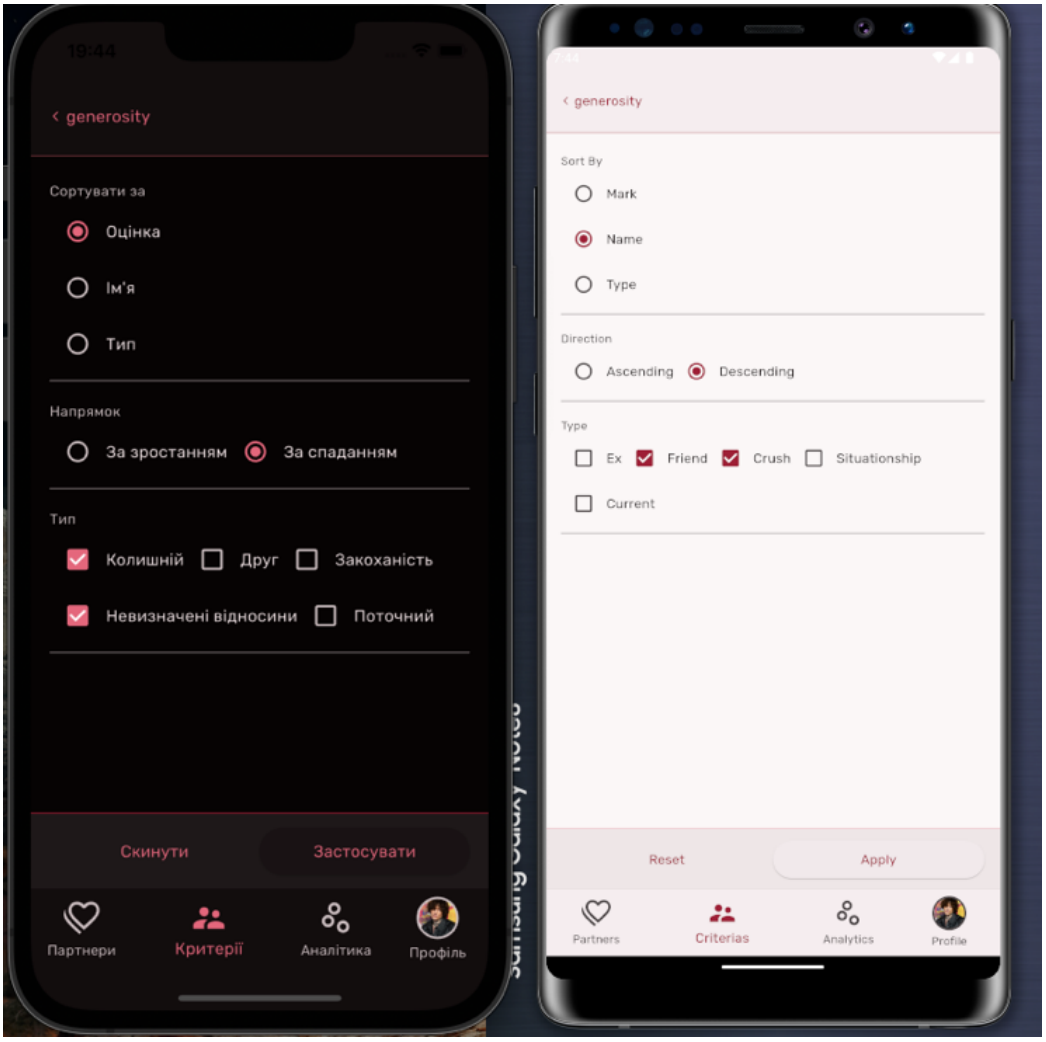

Рисунок 5.17 – Сторінка фільтрів партнерів за критерієм (рисунок виконано самостійно)

Крім того, передбачена опція вибору напрямку сортування – за зростанням або за спаданням, що дозволяє більш гнучко формувати порядок відображення партнерів відповідно до потреб користувача.

Також можна застосувати додатковий фільтр за типом партнера, що дає змогу обмежити список лише тими партнерами, які відповідають обраному типу. Це особливо корисно при роботі з великим масивом даних, коли потрібно швидко знайти потрібні записи.

Інтерфейс фільтрів інтуїтивно зрозумілий і дозволяє швидко налаштувати відображення партнерів, підвищуючи ефективність роботи з системою.

Якщо користувач переходить на вкладку «Аналітика», йому відкривається можливість переглянути інтерактивні графіки, які допомагають глибше проаналізувати дані (див. рис. 5.18).

Рисунок 5.18 – Вкладка «Аналітика» (рисунок виконано самостійно)

Перший графік – це рейтинг партнерів, представлений у вигляді гістограми з упорядкованими за значенням рейтингами. Ця гістограма дозволяє користувачу візуально оцінити розподіл рейтингів серед партнерів. Для зручності навігації по графіку доступні функції гортання, які дають змогу переглядати різні частини гістограми, а також зміни масштабу, що допомагає зосередитися на потрібному діапазоні значень.

Другий графік – кореляція критеріїв, яка подана у вигляді теплової діаграми. Вона демонструє коефіцієнти кореляції між різними критеріями, що дозволяє швидко виявити взаємозв’язки та залежності між ними. За допомогою

цієї діаграми користувач може оцінити, наскільки сильно пов’язані різні критерії, що є корисним для прийняття обґрунтованих рішень на основі даних.

Обидва графіки є інтерактивними, що забезпечує зручний і гнучкий спосіб роботи з аналітикою без потреби у додаткових інструментах.

Останньою вкладкою на головній сторінці є «Профіль» (див. рис. 5.19). У цьому розділі користувач може переглянути основну інформацію про себе, а також отримати доступ до налаштувань і додаткових функцій.

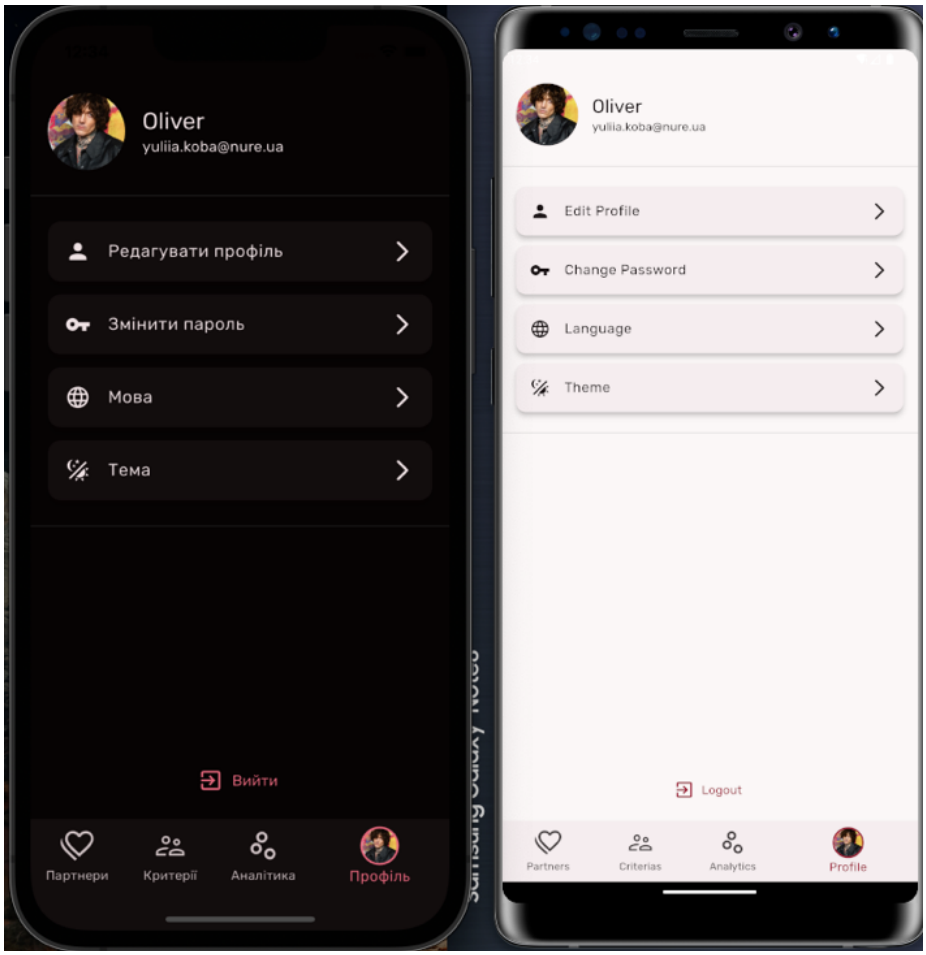

Рисунок 5.19 – Вкладка «Профіль» (рисунок виконано самостійно)

При натисканні кнопки «Редагувати профіль» відкривається відповідна сторінка (див. рис. 5.20), яка дозволяє користувачу внести зміни до персональної інформації. На цій сторінці користувач може оновити своє фото профілю, вибравши нове з пристрою або зробивши знімок. Також доступні поля для редагування імені, що дає змогу оновити відображуване ім’я в системі, а також електронної пошти, що використовується для авторизації та сповіщень.

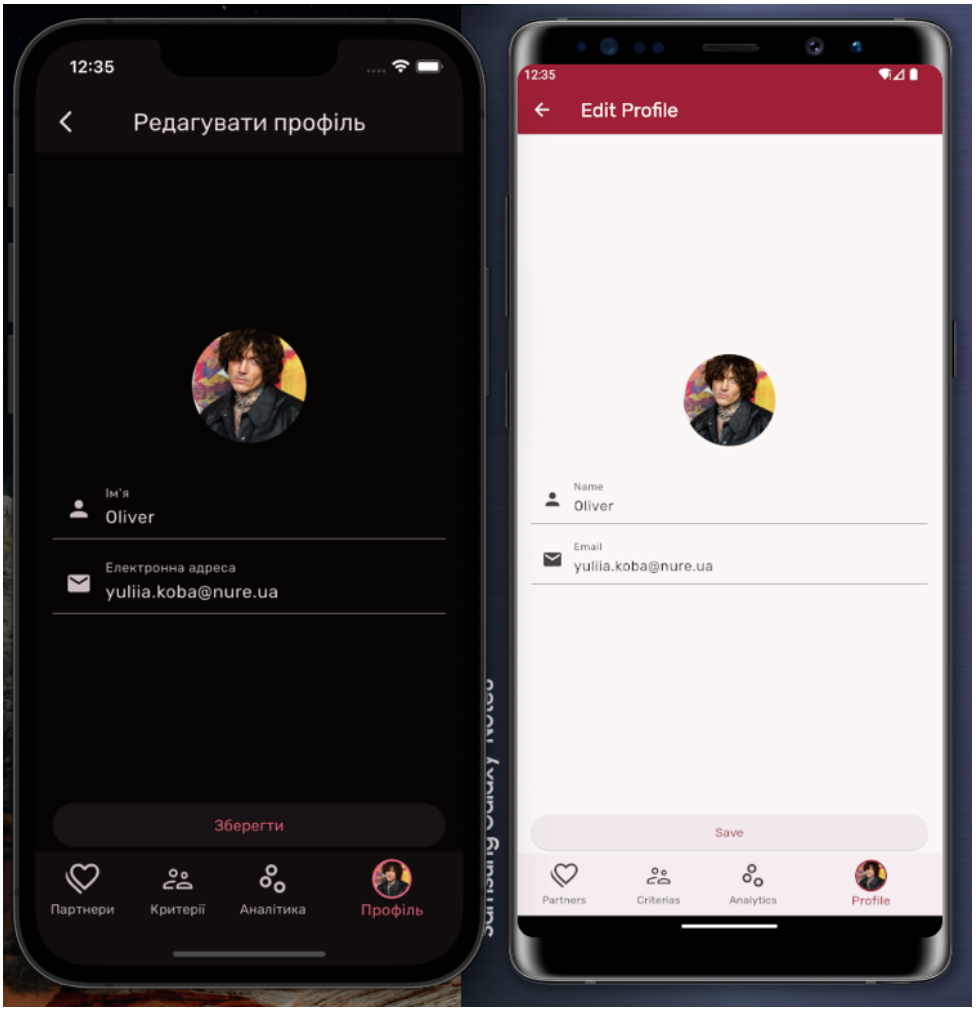

Рисунок 5.20 – Сторінка редагування профілю (рисунок виконано самостійно)

Інтерфейс сторінки простий та зручний, містить кнопки для збереження внесених змін або скасування редагування. Після збереження оновлені дані миттєво відображаються у профілі користувача, що забезпечує актуальність і точність інформації.

При натисканні кнопки «Змінити пароль» відкривається форма для зміни паролю (див. рис. 5.21). У цій формі користувач повинен ввести свій поточний пароль для підтвердження особи, а також новий пароль і повторити його у відповідному полі для підтвердження правильності введення.

Інтерфейс форми забезпечує надійність і зручність: поля для введення паролів мають можливість показати або приховати символи для зручності користувача, а також передбачена валідація нових паролів на відповідність

вимогам безпеки (наприклад, мінімальна довжина, наявність цифр і спеціальних символів).

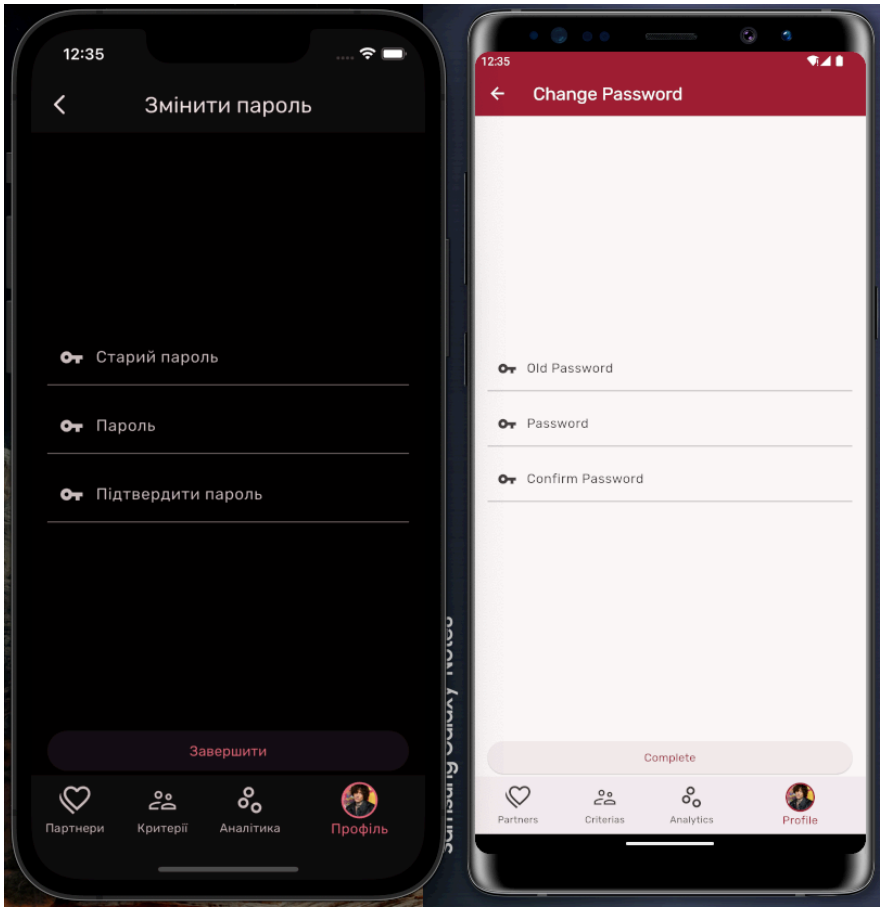

Рисунок 5.21 – Сторінка зміни паролю (рисунок виконано самостійно)

Після введення всіх даних користувач може підтвердити зміну паролю кнопкою «Зберегти» або відмінити операцію кнопкою «Скасувати». У разі успішної зміни паролю система відображає відповідне повідомлення про успіх.

Щоб змінити мову інтерфейсу, користувачеві необхідно натиснути кнопку «Мова». Після цього відкривається діалогове вікно (див. рис. 5.22), у якому представлено список доступних мов для вибору. Користувач може обрати потрібну мову, після чого інтерфейс системи автоматично оновиться відповідно до обраної мови.

Таким чином, система підтримує багатомовність і забезпечує комфортне використання для користувачів з різними мовними уподобаннями. Наразі застосунок підтримує українську та англійську мови. Інші варіанти локалізації можуть бути додані за потреби.

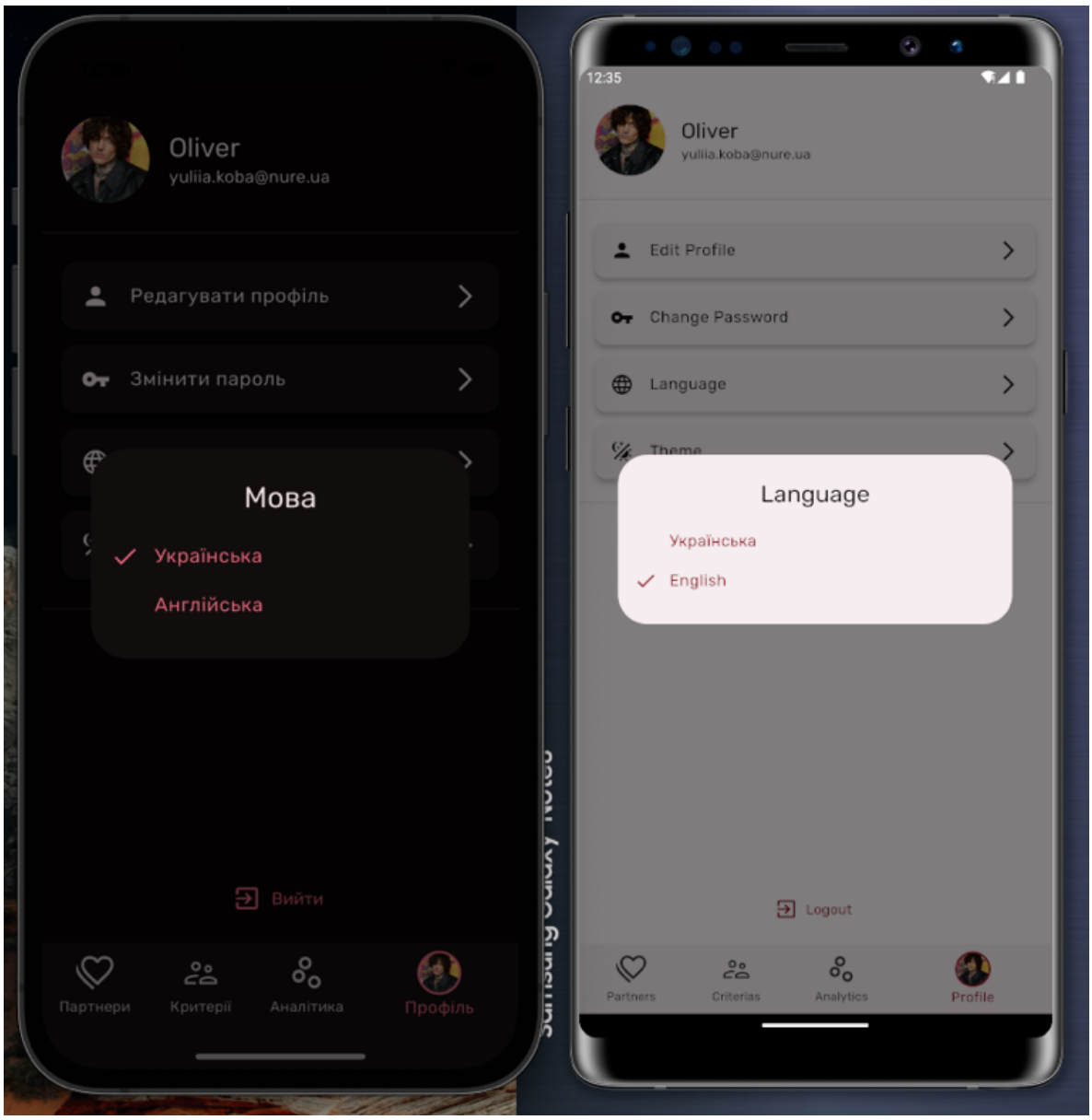

Рисунок 5.22 – Діалогове вікно зміни мови (рисунок виконано самостійно)

Щоб змінити тему оформлення застосунку, користувачеві потрібно натиснути кнопку «Тема». Після цього відкриється діалогове вікно (див. рис. 5.23), у якому буде представлено перелік доступних тем оформлення. Користувач може обрати одну з тем відповідно до своїх вподобань – наприклад, світлу, темну або інші варіанти, якщо такі передбачені.

Вибір теми миттєво застосовується до інтерфейсу застосунку, забезпечуючи комфортне відображення та зручність використання у різних умовах освітлення.

Таким чином, користувач має можливість персоналізувати зовнішній вигляд застосунку під свої вподобання.

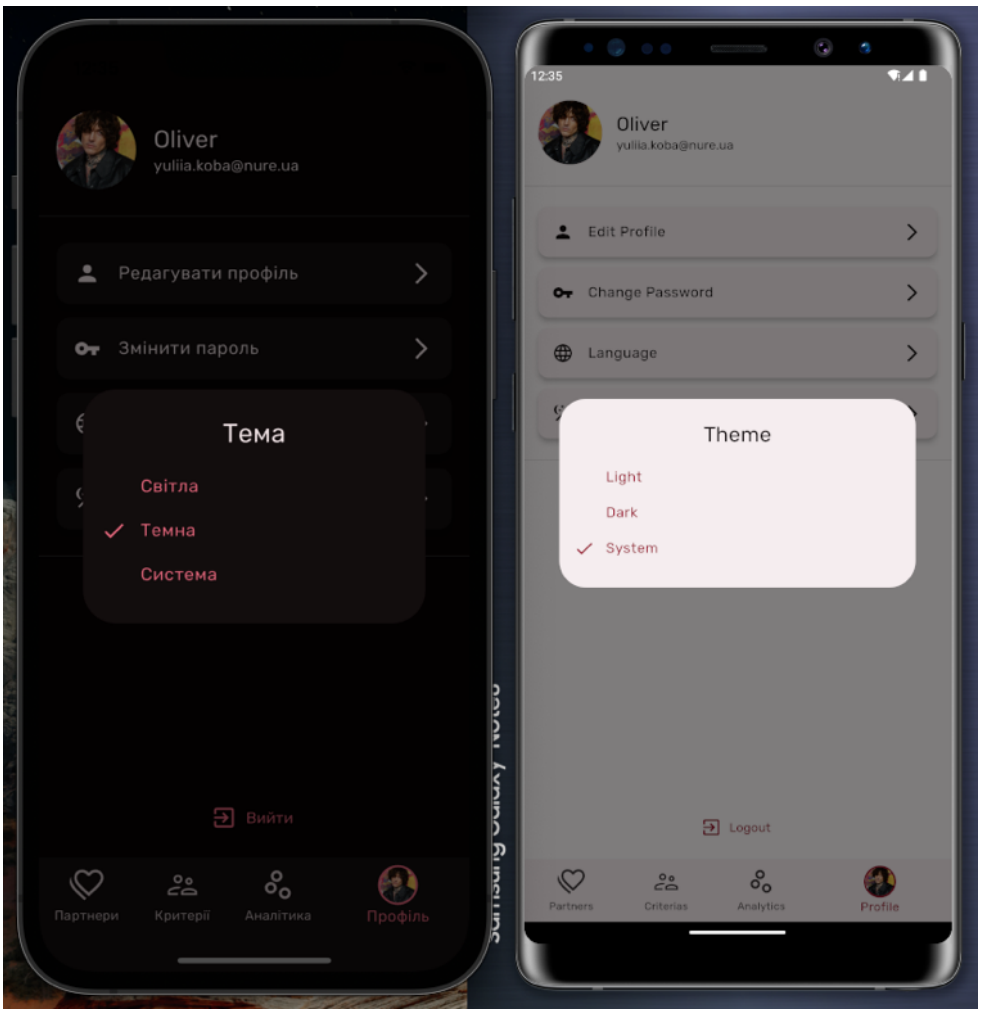

Рисунок 5.23 – Діалогове вікно зміни мови (рисунок виконано самостійно)

Щоб вийти з акаунту, потрібно натиснути кнопку «Вийти».

Розроблений застосунок відповідає принципам UI/UX-дизайну: всі елементи розташовані логічно та зручно, що робить використання інтуїтивно зрозумілим для користувача [29].

Повний код розробленого застосунку можна знайти на посиланням [30] у списку джерел.

ВИСНОВКИ

Отже, у ході роботи було проведено аналіз предметної галузі та наукових джерел і досліджено безсерверні Flutter застосунки та способи їх оптимізації. Також було розглянуто різні методи програмної реалізації безсерверних архітектур за допомогою хмарних сервісів. У процесі роботи було застосовано багатокритеріальний підхід для вибору оптимального хмарного сервісу.

У роботі було обрано оптимальне безсерверне рішення для розробки Flutter-застосунку, враховуючи різні критерії, які впливають на зручність використання, функціональні можливості, масштабованість і витрати, спроектовано структуру бази даних – визначено основні сутності, їх властивості та зв'язки між ними, розроблено схему архітектури застосунку – визначено компоненти системи, їх взаємодію та основні вимоги до реалізації серверної та клієнтської частин, створено базу даних у хмарі – налаштовано базу даних для інтеграції з безсерверною логікою застосунку у обраному хмарному сервісі, спроектовано та розроблено програмне забезпечення для дослідження – створено прототип застосунку, який дозволить тестувати різні методи оптимізації рендерингу інтерфейсу, проведено експериментальне дослідження обраних методів оптимізації рендерингу – використано лінійну регресійну модель для аналізу продуктивності різних методів, зокрема, вимірюючи час рендерингу інтерфейсу з застосуванням і без застосування оптимізаційних технік.

У результаті виконаного дослідження було розроблено оптимізований Flutter застосунок з використанням хмарної платформи Supabase, що демонструє підвищену продуктивність рендерингу користувацького інтерфейсу. Систематизовано методи оптимізації UI потоку та створено практичні рекомендації щодо їх застосування. Результати роботи можуть бути використані розробниками мобільних застосунків для підвищення продуктивності своїх проектів та оптимізації процесу розробки.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Дудар З., Кобзєв В., Семенець В. Історія, стан та перспективи розвитку загальнодоступних електронних послуг. *Міжнародна науково-практична конференція «Застосування інформаційних технологій у підготовці та діяльності сил охорони правопорядку»* : Зб. тез доп., м. Харків. 2023. С. 226–228.

2. Luo, S., Li, K., Xing, H., & Fan, P. (2024). Efficient and Flexible Component Placement for Serverless Computing. *IEEE Systems Journal*, 18, 1104-1114. https://doi.org/10.1109/JSYST.2024.3381590.

3. Raza, A., Akhtar, N., Isahagian, V., Matta, I., & Huang, L. (2023). Configuration and Placement of Serverless Applications Using Statistical Learning. *IEEE Transactions on Network and Service Management*, 20, 1065-1077. https://doi.org/10.1109/TNSM.2023.3254437.

4. Lin, C., & Khazaei, H. (2021). Modeling and Optimization of Performance and Cost of Serverless Applications. *IEEE Transactions on Parallel and Distributed Systems*, 32, 615-632. https://doi.org/10.1109/TPDS.2020.3028841.

5. Eismann, S., Scheuner, J., Van Eyk, E., Schwinger, M., Grohmann, J., Abad, C., & Iosup, A. (2020). Serverless Applications: Why, When, and How?. *IEEE Software*, 38, 32-39. https://doi.org/10.1109/MS.2020.3023302.

6. Rajan, A. (2024). Optimizing Serverless Function Orchestration for Complex Workflows in Cloud Platforms. *American Journal of Computing and Engineering*. https://doi.org/10.47672/ajce.2458.

7. Ship, H., Shindin, E., Wang, C., Arroyo, D., & Tantawi, A. (2023). Optimizing Simultaneous Autoscaling for Serverless Cloud Computing. *2024 IEEE 17th International Conference on Cloud Computing (CLOUD)*, 105-114. https://doi.org/10.1109/CLOUD62652.2024.00022.

8. Mampage, A., Karunasekera, S., & Buyya, R. (2023). A Deep Reinforcement Learning based Algorithm for Time and Cost Optimized Scaling of Serverless Applications. *ArXiv*, abs/2308.11209. https://doi.org/10.48550/arXiv.2308.11209.

9. Winzinger, S., & Wirtz, G. (2019). Model-Based Analysis of Serverless Applications. *2019 IEEE/ACM 11th International Workshop on Modelling in Software Engineering (MiSE)*, 82-88. https://doi.org/10.1109/MiSE.2019.00020.

10. Kallas, K., Zhang, H., Alur, R., Angel, S., & Liu, V. (2023). Executing Microservice Applications on Serverless, Correctly. *Proceedings of the ACM on Programming Languages*, 7, 367 - 395. https://doi.org/10.1145/3571206.

11. Rausch, T., Rashed, A., & Dustdar, S. (2021). Optimized container scheduling for data-intensive serverless edge computing. *Future Gener. Comput. Syst.*, 114, 259-271. https://doi.org/10.1016/j.future.2020.07.017.

12. De Silva, D., & Hewawasam, L. (2024). The Impact of Software Testing on Serverless Applications. *IEEE Access*, 12, 51086-51099. https://doi.org/10.1109/ACCESS.2024.3384459.

13. Ali, A., Pinciroli, R., Yan, F., & Smirni, E. (2022). Optimizing Inference Serving on Serverless Platforms. *Proc. VLDB Endow.*, 15, 2071-2084. https://doi.org/10.14778/3547305.3547313.

14. Nanavati, J., Patel, S., Patel, U., & Patel, A. (2024). Critical Review and Fine-Tuning Performance of Flutter Applications. *2024 5th International Conference on Mobile Computing and Sustainable Informatics (ICMCSI)*, 838-841. https://doi.org/10.1109/ICMCSI61536.2024.00131.

15. Szczepanik, M., & Kedziora, M. (2020). State Management and Software Architecture Approaches in Cross-platform Flutter Applications. , 407-414. https://doi.org/10.5220/0009411604070414.

16. Boukhary, S., & Colmenares, E. (2019). A Clean Approach to Flutter Development through the Flutter Clean Architecture Package. *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, 1115-1120. https://doi.org/10.1109/CSCI49370.2019.00211.

17. Гребеннік І. В. Методи підтримки прийняття рішень : навч. посібник / І. В. Гребеннік, Т. Є. Романова, А. Д. Тевяшев, Г. М Яськов ; МОН України, Харк. нац. ун-т радіоелектроніки. – Харків : ХНУРЕ, 2010. – 127 с.

18. Flutter Selverless Services Questionnaire. *Stack Overflow*. URL: https://stackoverflow.com/flutter-selverless-services-questionnaire (дата звернення: 19.11.2024).

19. Pricing | Firestore | Google Cloud. *Google Cloud*. URL: https://cloud.google.com/firestore/pricing (дата звернення: 14.12.2024).

20. AWS Amplify Pricing | Front-End Web & Mobile | Amazon Web Services. *Amazon Web Services, Inc.* URL: https://aws.amazon.com/en/amplify/pricing/ (дата звернення: 14.12.2024).

21. Azure Functions pricing. *Microsoft*. URL: https://azure.microsoft.com/en-us/pricing/details/functions/ (дата звернення: 14.12.2024).

22. Pricing & Fees | Supabase. *Supabase*. URL: https://supabase.com/pricing (date of access: 14.12.2024).

23. Contributors to Вікі Як я зустрів вашу маму. Matchmaker. *Вікі Як я зустрів вашу маму*. URL: https://how-i-met-your-mother.fandom.com/uk/wiki/Matchmaker (дата звернення: 14.12.2024).

24. Мартін Р. С. Чиста архітектура. Фабула, 2019. 368 с.

25. Dudar Z., Kozyriev A. Finite predicate-driven logic networks method for enhanced education data analysis. *Radioelectronic and Computer Systems*. 2024. Т. 2024, № 3. С. 205–215. URL: https://doi.org/10.32620/reks.2024.3.14 (дата звернення: 30.05.2025).

26. Koba Y., Kravets N. CONCURRENCY IMPLEMENTATION FEATURES IN THE PROGRAMMING LANGUAGE DART. *Збірник наукових праць SCIENTIA*. 2021.

27. Dudar Z., Liashyk V. Method for solving quantifier linear equations based on the algebra of linear predicate operations. *Radioelectronic and Computer Systems*. 2025. Т. 2025, № 1. С. 102–112. URL: https://doi.org/10.32620/reks.2025.1.07 (дата звернення: 30.05.2025).

28. Dudar Z., Litvin S. Ontological description method for building service-oriented distributed learning systems. *INNOVATIVE TECHNOLOGIES AND*

SCIENTIFIC SOLUTIONS FOR INDUSTRIES. 2024. № 1 (27). С. 39–53. URL: https://doi.org/10.30837/itssi.2024.27.039 (дата звернення: 30.05.2025).

29. Calonaci D. Designing User Interfaces: Exploring User Interfaces, UI Elements, Design Prototypes and the Figma UI Design Tool (English Edition). BPB Publications, 2021. 230 с.

30. GitHub - klukovka/synergy_scanner. *GitHub*. URL: https://github.com/klukovka/synergy_scanner (дата звернення: 27.05.2025).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ЗА НАУКОВИМИ НАПРЯМАМИ КЕРІВНИКА ТА НАУКОВЦІВ КАФЕДРИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

1. Дудар З., Кобзєв В., Семенець В. Історія, стан та перспективи розвитку загальнодоступних електронних послуг. *Міжнародна науково-практична конференція «Застосування інформаційних технологій у підготовці та діяльності сил охорони правопорядку»* : Зб. тез доп., м. Харків. 2023. С. 226–228.

25. Dudar Z., Kozyriev A. Finite predicate-driven logic networks method for enhanced education data analysis. *Radioelectronic and Computer Systems*. 2024. Т. 2024, № 3. С. 205–215. URL: https://doi.org/10.32620/reks.2024.3.14 (дата звернення: 30.05.2025).

26. Koba Y., Kravets N. CONCURRENCY IMPLEMENTATION FEATURES IN THE PROGRAMMING LANGUAGE DART. *Збірник наукових праць SCIENTIA*. 2021.

27. Dudar Z., Liashyk V. Method for solving quantifier linear equations based on the algebra of linear predicate operations. *Radioelectronic and Computer Systems*. 2025. Т. 2025, № 1. С. 102–112. URL: https://doi.org/10.32620/reks.2025.1.07 (дата звернення: 30.05.2025).

28. Dudar Z., Litvin S. Ontological description method for building service-oriented distributed learning systems. *INNOVATIVE TECHNOLOGIES AND SCIENTIFIC SOLUTIONS FOR INDUSTRIES*. 2024. № 1 (27). С. 39–53. URL: https://doi.org/10.30837/itssi.2024.27.039 (дата звернення: 30.05.2025).