

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ
ПОКРАЩЕННЯ ВІДНОСИН ІЗ СИСТЕМОЮ ГЕЙМІФІКАЦІЇ
DEVELOPMENT OF A MOBILE APPLICATION TO IMPROVE
RELATIONS WITH THE GAMIFICATION SYSTEM

Виконав(ла): студент(ка) 4 курсу, групи КНТ-151
Спеціальності 121 Інженерія програмного
(код і найменування спеціальності)
забезпечення

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

СЕВЕРЕНКО М.В.

(ПРІЗВИЩЕ та ініціали)

Керівник ГЛАДКОВА О.М.

(ПРІЗВИЩЕ та ініціали)

Рецензент ДЬЯЧУК Т.С.

(ПРІЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2025 року

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

СЕВЕРЕНКА Максима Валерійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1.Тема проєкту (роботи) Розробка мобільного застосунку для покращення відносин із системою гейміфікації. Development of a Mobile Application to Improve Relations with the Gamification System

керівник проєкту (роботи) к.т.н., ГЛАДКОВА Ольга Миколаївна,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої від “28” квітня 2025 року № 209

2. Строк подання студентом проєкту (роботи) 02 червня 2025 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2.Аналіз і вибір методів та засобів розробки 3. Проектування та реалізація системи

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

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-3 Основна частина	ГЛАДКОВА О.М., доцент		
Нормоконтроль	БЄЛОВА А.В., асистент		

7. Дата видачі завдання “ 15 ” квітня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 3
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

_____ Максим СЕВЕРЕНКО
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)
(підпис) (Ім'я ПРІЗВИЩЕ)

_____ Ольга ГЛАДКОВА

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 72 с., 13 табл., 19 рис., 2 дод., 17 джерел.

МОБІЛЬНИЙ ДОДАТОК, FLUTTER, ВЗАЄМОДІЯ В ПАРИ, СИНХРОНІЗАЦІЯ КОРИСТУВАЧІВ, FIREBASE, REST API.

Об'єкт дослідження – процеси взаємодії користувачів орієнтованих на підтримку та розвиток міжособистісних відносин.

Предмет дослідження – засоби реалізації функціоналу мобільного застосунку для парної взаємодії, зокрема механізмів синхронізації даних, гейміфікації, внутрішньої валюти та рекомендаційної системи.

Мета роботи – розробити мобільний застосунок для пар, який сприяє зміцненню стосунків через інтерактивні завдання, поради, систему винагород і спільну взаємодію.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування, архітектура Clean Architecture, фреймворк Flutter, мова програмування Dart, системи Supabase (PostgreSQL) та Firebase Auth, Android-емулятор та смартфон з операційною системою Android 10.

Результати. У результаті виконання роботи було розроблено мобільний застосунок для пар, що забезпечує синхронізацію акаунтів, обмін завданнями, порадами та внутрішню систему винагород.

Галузь використання. Мобільний застосунок може використовуватись у сфері інформаційних технологій, психології та особистісного розвитку. Він орієнтований на широке коло користувачів — пари, які бажають покращити стосунки, підвищити якість спілкування та урізноманітнити спільне дозвілля. Також застосунок може бути корисним фахівцям у сфері сімейного консультування як допоміжний інструмент для рекомендацій.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 72 pages, 13 tables, 19 figures, 2 appendixes, 17 sources.

MOBILE APPLICATION, FLUTTER, PAIR INTERACTION, USER SYNCHRONIZATION, FIREBASE, REST API.

The object of the study is the processes of user interaction in mobile applications focused on supporting and developing interpersonal relationships.

The subject of the study is the technic of the implementing the functionality of a mobile application for pair interaction, in particular, mechanisms for data synchronization, gamification, internal currency and a recommendation system.

The purpose of the work is to develop a mobile application for couples that helps strengthen relationships through interactive tasks, advice, a reward system and joint interaction.

Materials, methods and technical means: object-oriented programming, Clean Architecture, Flutter framework, Dart programming language, Supabase (PostgreSQL) and Firebase Auth systems, an Android emulator and a smartphone with the Android 10 operating system.

Results. As a result of the work, a mobile application for couples was developed that provides account synchronization, task exchange, advice and an internal reward system.

Field of application. The developed mobile application can be used in the field of information technology, psychology and personal development. It is aimed at a wide range of users - couples who want to improve their relationships, increase the quality of communication, and diversify their joint leisure time. The application can also be useful to family counseling professionals as an auxiliary tool for recommendations.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Аналіз предметної області.....	10
1.1 Типи міжособистісних відносин	10
1.2 Типові проблеми у взаєминах.....	11
1.3 Аналіз переваг і недоліків електронних застосунків для підтримки стосунків	12
1.4 Аналіз існуючих додатків для пар.....	13
1.5 Оцінка переваг і недоліків аналогів	15
1.6 Постановка задачі.....	16
1.6.1 Основні функціональні вимоги до додатку.....	16
1.6.2 Нефункціональні вимоги.....	17
1.6.3 Визначення вхідних і вихідних даних	18
1.6.4 Модель користувацької взаємодії (User Flow).....	18
1.6.5 Сценарії використання	19
1.7 Висновки за розділом	19
2 Аналіз і вибір методів та засобів розробки	20
2.1 Вибір мови програмування та середовища розробки.....	20
2.2 Архітектурний підхід до проєкту	22
2.3 Сторонні бібліотеки та сервіси	24
2.3.1 Firebase Authentication	25
2.3.2 Supabase – база даних та бекенд-сервіс	25
2.3.3 Flutter Bloc (Cubit) – керування станом	25
2.4 Інструменти для управління проєктом	27
2.5 Висновки за розділом	28
3 Проєктування та реалізація системи	30
3.1 Загальна структура застосунку	30

	7
3.2 UI/UX-дизайн та ескізи екранів	31
3.3 Проектування бази даних	33
3.4 Організація авторизації користувачів	36
3.4.1 Підтримувані способи входу	38
3.4.2 Архітектурна реалізація	38
3.4.3 Основні функції авторизації	38
3.4.4 Складності реалізації	39
3.5 Модель синхронізації партнерів	40
3.5.1 Принцип роботи	41
3.6 Реалізація завдань та порад	43
3.6.1 Щоденні поради (Daily Tips)	44
3.6.2 Завдання для партнерів (Tasks)	44
3.7 Функція “Say Hello” та внутрішня валюта	45
3.7.1 Функція “Say Hello”: концепція і соціальний вплив	45
3.7.2 Внутрішня валюта: мотивація через гейміфікацію	45
3.7.3 Потенційне використання валюти	46
3.7.4 Технічна реалізація (поточний стан)	46
3.7.5 Примітки щодо подальшого розвитку	47
3.8 Тестування MVP	47
3.9 Інструкція користувача	49
3.9.1 Встановлення застосунку	49
3.9.2 Реєстрація та вхід	49
3.9.3 Основні функції застосунку	50
Висновки	51
Перелік джерел посилання	52
Додаток А Текст програми	54
Додаток Б Слайди презентації	67

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface– інтерфейс прикладного програмування;
Auth	– Authentication – аутентифікація;
Clean	– підхід до архітектури ПЗ, що розділяє
Architecture	відповідальності між рівнями;
Flutter	– фреймворк для створення мобільних застосунків на мові Dart;
ID	– Identifier – унікальний ідентифікатор;
JSON	– JavaScript Object Notation – текстовий формат обміну даними;
MVP	– Minimum Viable Product – мінімально життєздатний продукт;
Supabase	– платформа з відкритим кодом для створення бекенду;
UI	– User Interface – користувацький інтерфейс;
UX	– User Experience – користувацький досвід;
БД	– база даних.

ВСТУП

Людські взаємини – фундаментальна частина існування суспільства. Вони охоплюють широкий спектр форм: від дружби та родинних зв'язків до романтичних стосунків і взаємодії між батьками та дітьми. Кожен тип стосунків потребує підтримки, взаєморозуміння, турботи та спільної роботи над комунікацією. В умовах сучасного ритму життя, постійного стресу, дистанційної роботи та цифрової ізоляваності – навіть найміцніші зв'язки можуть зазнавати труднощів.

Особливо помітним це стає у романтичних парах. Недостаток уваги, рутинність, відсутність якісної комунікації – це лише частина типових проблем, які можуть призводити до віддалення партнерів одне від одного. При цьому багато пар прагнуть зміцнити свої стосунки, проте не завжди мають ефективні інструменти для цього.

Останні роки продемонстрували зростання інтересу до електронних застосунків, що допомагають парам покращити спілкування, вносити новизну у взаємодію та спільно проводити час. Саме створення такого застосунку – інтерактивного, гейміфікованого і спрямованого на щоденне зміцнення стосунків – є темою даного дипломного проєкту.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Типи міжособистісних відносин

Міжособистісні відносини – це сукупність соціальних зв'язків, які формуються між людьми впродовж їхнього життя. Ці зв'язки базуються на емоційній близькості, взаємній довірі, відповідальності, спільних інтересах і досвіді. Умовно міжособистісні стосунки можна поділити на кілька основних типів:

Романтичні стосунки – це взаємозв'язки, які виникають між партнерами на основі емоційної та фізичної близькості, прихильності та прагнення до спільного майбутнього. У таких стосунках особливо важливими є щоденне спілкування, спільне проведення часу, підтримка та розвиток довіри [1].

Дружні стосунки – дружба ґрунтується на взаємній симпатії, спільних інтересах та готовності підтримати одне одного в складних життєвих ситуаціях. Це неформальні, добровільні зв'язки, які, втім, можуть бути дуже міцними та довготривалими [1].

Сімейні стосунки – включають відносини між батьками та дітьми, братами та сестрами, подружжям, а також іншими родичами. Такі зв'язки формуються з дитинства і часто залишаються важливими протягом усього життя. Характерною особливістю сімейних стосунків є глибока емоційна прив'язаність, відповідальність і необхідність вирішення конфліктів через компроміс [1].

Виховні стосунки (батьки – діти) – цей тип відносин базується на турботі, авторитеті, навчанні та забезпеченні базових потреб. У таких взаємодіях ключовими є довіра, послідовність у діях і здатність слухати одне одного [1].

Професійно-соціальні зв'язки – хоча вони менш емоційно забарвлені, ці стосунки часто мають велике значення для психологічного комфорту

людини. До них належать зв'язки з колегами, керівниками, сусідами або знайомими в спільнотах за інтересами [1].

Кожен тип стосунків має свої унікальні виклики. Водночас існують універсальні труднощі, характерні для багатьох взаємин: відсутність відкритої комунікації, нерівномірний розподіл ролей, конфлікти цінностей, брак спільного часу тощо. Саме ці проблеми стають об'єктом подальшого аналізу в наступному підрозділі.

1.2 Типові проблеми у взаєминах

Незалежно від типу міжособистісних відносин, у них періодично виникають труднощі. Причини можуть бути як зовнішніми (стрес, умови життя, робота), так і внутрішніми (особистісні відмінності, емоційна нестабільність, невміння слухати). Нижче наведено найбільш поширені проблеми, з якими стикаються пари, друзі та члени сім'ї.

Відсутність ефективної комунікації. Часто сторони не висловлюють свої потреби або очікування, не вміють слухати одне одного або не мають звички говорити відкрито. Це призводить до накопичення непорозумінь, образ і конфліктів.

Різниця у цінностях та життєвих цілях. Партнери або члени родини можуть мати різні уявлення про те, що є важливим у житті. Без взаємного узгодження пріоритетів виникає напруження, що ускладнює досягнення згоди.

Нерівномірний розподіл відповідальності. У стосунках – зокрема в романтичних і сімейних – одна зі сторін може почуватися перевантаженою через те, що бере на себе більше обов'язків (емоційних, фінансових, побутових).

Втрата емоційного зв'язку. У довготривалих відносинах партнери можуть перестати ділитися почуттями, спільно переживати нові емоції або просто проводити час разом. Це часто призводить до охолодження стосунків.

Вплив цифрових технологій. Надмірне використання гаджетів і соціальних мереж іноді витісняє живе спілкування, що послаблює зв'язок між людьми, знижує рівень довіри та залученості у взаємини.

Відсутність спільної активності. Коли люди не мають спільних цілей, розваг чи захоплень, їхнє емоційне зближення послаблюється. Відсутність нових вражень і досвіду призводить до рутини та нудьги.

Розуміння цих проблем – перший крок до їх вирішення. Саме на цьому базується ідея створення цифрового інструменту, який допоможе людям краще пізнавати себе й одне одного, покращувати комунікацію та формувати позитивні звички взаємодії.

1.3 Аналіз переваг і недоліків електронних застосунків для підтримки стосунків

Цифрові інструменти все частіше використовуються для підтримки і розвитку стосунків – від месенджерів до спеціалізованих мобільних застосунків для пар, родин або друзів. Вони можуть бути ігровими, терапевтичними, мотиваційними чи організаційними за своєю суттю. У цьому підрозділі розглянемо ключові переваги та недоліки таких рішень. Спочатку наведемо переваги.

Доступність – більшість таких застосунків працює на мобільних пристроях і не вимагає спеціальних знань або додаткового обладнання.

Мотивація через ігровізацію – використання балів, завдань, досягнень або систем нагород стимулює регулярну взаємодію між користувачами, допомагає в подоланні емоційної відстані.

Покращення комунікації – деякі застосунки пропонують готові питання, вправи для діалогу або інструменти для розв'язання конфліктів, що особливо корисно для пар з труднощами в комунікації.

Розвиток спільних звичок – застосунки часто пропонують щоденні виклики, спільне планування дозвілля або звички вдячності, що покращує якість стосунків.

Недоліки наведені нижче.

Обмежена індивідуалізація. Більшість застосунків мають шаблонні завдання, які не враховують особливостей конкретної пари або типу відносин (наприклад, романтичні пари проти родичів або друзів).

Стимул лише на короткий термін. Без глибокої мотивації з боку користувачів, інтерес до програми часто згасає після перших тижнів використання.

Заміна живого спілкування. Існує ризик того, що спілкування через застосунок стане заміником реальної розмови, а не її доповненням.

Небезпека для конфіденційності. При недостатньому захисті персональних даних існує загроза витоку чутливої інформації, що особливо важливо у сфері особистих стосунків.

Отже, хоча цифрові застосунки мають потенціал позитивного впливу на якість стосунків, їх ефективність залежить від дизайну, рівня персоналізації, зручності використання та зацікавленості самих користувачів. Саме ці аспекти мають враховуватись під час створення нового рішення, орієнтованого на підтримку взаєморозуміння у стосунках.

1.4 Аналіз існуючих додатків для пар

На ринку мобільних застосунків існує чимало рішень, що покликані покращити комунікацію у стосунках, зміцнити емоційний зв'язок і внести у відносини більше інтерактивності та гри. Нижче наведено огляд найпоширеніших додатків для романтичних пар, з розкриттям механік, які вони використовують для підтримки взаємодії.

Between – це один із найпопулярніших додатків для пар. Його основна функція — створення приватного простору для двох людей. Додаток містить

чат, фотогалерею, щоденник, календар важливих подій. Комунікація переважно відбувається у вигляді звичайного листування, без ігрових елементів. Натомість Between створює атмосферу емоційної інтимності, де пара може зберігати спогади, ділитися емоціями, планувати спільне дозвілля [2].

Couple також орієнтований на приватне листування, однак включає функцію «Thumbkiss», яка дозволяє двом користувачам одночасно доторкнутися до екрана й «відчути» дотик партнера. Присутні нотатки, завдання, можливість надсилати нагадування. Гейміфікація мінімальна, проте є базові інструменти планування та емоційного зв'язку [3].

Love Nudge базується на концепції «п'яти мов любові» Гері Чепмена. Застосунок пропонує завдання відповідно до мов любові партнера (подарунки, дотики, слова, дії, час разом). Користувачі отримують «підказки» для щоденних дій, можуть виставляти нагадування, ставити галочки за виконанні завдання. Є система звітів та оцінки прогресу, однак балів або рейтингових таблиць немає. Основна мета – допомогти краще зрозуміти потреби партнера [4].

Agape – це застосунок, орієнтований на діалоги. Кожного дня пара отримує одне питання для спільного обговорення (наприклад: «Що ти найбільше цінуєш у наших стосунках?»). Мета – сприяти глибшому емоційному розкриттю. Присутня елемент гейміфікації – за щоденні відповіді користувачі отримують віртуальні нагороди, зростає рівень «зв'язку». Застосунок ідеально підходить для розвитку емоційної відкритості, але майже не включає ігрових елементів або завдань [5].

Coral спеціалізується на сексуальній близькості у стосунках. Він містить інтерактивні вправи, статті та щоденні завдання для пар. Користувачі можуть проходити міні-курси разом, обговорювати інтимні теми у закритому просторі. Додаток має структуру, схожу на навчальний курс із модулями та прогрес-барами. Це приклад застосунку, де ігрові елементи поєднані з освітніми матеріалами [6].

Paired – один з найбільш універсальних додатків для розвитку стосунків. Він містить:

- щоденні питання;
- інтерактивні тести (з результатами для обох партнерів);
- завдання на спільне виконання;
- статті та подкасти;
- ігрові елементи (балів як таких немає, але є прогрес-бар і досягнення).

Система питань адаптується до етапу стосунків, а відповіді партнери бачать лише після власної відповіді. Це підвищує чесність і сприяє розкриттю [7].

1.5 Оцінка переваг і недоліків аналогів

На основі аналізу існуючих мобільних додатків для пар можна виокремити спільні риси, сильні сторони, а також обмеження, які варто врахувати при розробці власного продукту.

Переваги аналогів:

- приватність взаємодії – усі розглянуті застосунки (Between, Couple, Paired) забезпечують закриту комунікацію лише між партнерами, що створює відчуття емоційної безпеки;
- інтерфейс, орієнтований на пару – спільні календарі, чат, фотогалереї, завдання – усе адаптовано під формат "двоє людей";
- емоційна взаємодія – додатки на зразок Agape і Paired пропонують щоденні питання, які стимулюють розмову, розкривають партнера, поглиблюють зв'язок;
- психологічна складова – Love Nudge використовує концепцію п'яти мов любові, Coral пропонує вправи для покращення сексуальної близькості – це збагачує додатки змістом;

- простота використання – у більшості випадків не потрібне складне навчання, усі функції доступні інтуїтивно.

Недоліки аналогів:

- низький рівень гейміфікації – у більшості застосунків відсутні повноцінні ігрові механіки (системи балів, рівнів, квестів), які могли б зробити взаємодію більш захопливою;

- обмеження лише на романтичні стосунки – жоден з проаналізованих додатків не підтримує інші типи стосунків (дружні, родинні, батьки–діти), хоча потреба у покращенні комунікації існує і там;

- відсутність гнучкості – більшість інструментів є закритими, і користувачі не можуть адаптувати типи завдань або створювати власні;

- обмеженість функціоналу без підписки – у багатьох випадках ключові функції (наприклад, повний доступ до завдань або аналітики) доступні лише у преміум-версії;

- слабка мотивація до постійного використання – через відсутність ігрових стимулів або системи досягнень користувачі можуть втратити інтерес після кількох днів активності.

Існуючі додатки для пар мають сильну емпатичну складову та забезпечують базову емоційну підтримку через комунікацію. Водночас їм бракує гнучкості й інноваційних гейміфікованих підходів, які могли б зробити взаємодію між партнерами цікавішою, глибшою та більш залученою. Це відкриває простір для створення нового застосунку, який би поєднав позитивні сторони попередників із динамічною, ігровою системою завдань, орієнтованих саме на підтримку й розвиток романтичних стосунків.

1.6 Постановка задачі

1.6.1 Основні функціональні вимоги до додатку

Реалізовані функції в MVP:

- реєстрація та вхід користувача через email і пароль та обліковий запис Google;
- синхронізація між двома акаунтами, що утворюють пару;
- відображення щоденного завдання, яке оновлюється раз на добу для обох користувачів;
- кнопка "Сказати привіт" – базова взаємодія між партнерами;
- проста внутрішня валюта ("сердечка"), яку користувач отримує за виконання завдань;
- стан виконання завдання, який синхронізується між партнерами.

Додаткові функції (плануються в повній версії):

- каталог завдань і порад з можливістю вибору та фільтрації;
- система нагород і рівнів, що підвищує мотивацію та залучення;
- push-сповіщення про нові завдання та активність партнера;
- можливість надсилати повідомлення, листівки, стікери тощо;
- аналітика прогресу пари: календар, статистика активності;
- розширені налаштування профілю та тем оформлення.

1.6.2 Нефункціональні вимоги

У рамках створення мінімально життєздатного продукту (MVP) додаток повинен відповідати таким базовим нефункціональним вимогам:

- зручність використання: інтерфейс має бути простим та зрозумілим для користувача, основні дії повинні виконуватись без складних переходів або зайвих кроків;
- мінімальна безпека: використовується проста автентифікація через email і пароль, дані зберігаються на сервері з базовим рівнем захисту, надання високого рівня безпеки не є пріоритетом на етапі MVP;
- продуктивність: додаток повинен працювати стабільно при середньому навантаженні, усі основні функції (вхід, синхронізація, перегляд завдань) мають виконуватись без тривалих затримок.

1.6.3 Визначення вхідних і вихідних даних

Вхідні дані:

- дані під час реєстрації: ім'я, email, пароль;
- код для з'єднання з партнером (або запит на створення пари);
- вибрані або створені користувачем завдання;
- щоденні дії користувача: виконання завдань, тощо.

Вихідні дані:

- щоденне завдання, яке призначається системою автоматично;
- стан виконання завдання (виконано/не виконано);
- поточний баланс внутрішньої валюти ("сердечка");
- стан пари: синхронізація, активність партнера.

1.6.4 Модель користувацької взаємодії (User Flow)

Модель користувацької взаємодії описує типовий шлях користувача від встановлення додатку до повсякденного користування ним.

Крок 1. Користувач відкриває додаток і проходить реєстрацію через email і пароль або через Google акаунт.

Крок 2. Після входу система пропонує створити або приєднатися до пари за допомогою унікального коду.

Крок 3. Після синхронізації пара отримує 5 сердечок для старту, щоденну пораду окремо згенеровану для кожного, можливість брати одне спільне завдання на день та можливість “привітатися” один з одним.

Крок 4. Користувач може ознайомитися із завданням, позначити його виконаним та отримати внутрішню винагороду, у випадку підтвердження зі сторони іншого партнера.

Крок 5. Протягом дня користувач також може натиснути кнопку "Сказати привіт" партнеру, при отриманні взаємної відповіді – користувачі отримають винагороду.

1.6.5 Сценарії використання

Сценарій 1. Створення акаунту та синхронізація:

- користувач завантажує додаток і проходить реєстрацію;
- створює пару або вводить код, отриманий від партнера.

Сценарій 2. Виконання щоденного завдання:

- на головному екрані відображається завдання дня;
- обоє партнерів читають його і позначають виконання, коли

завдання виконано.

Сценарій 3. Соціальна взаємодія: один з користувачів натискає кнопку "Сказати привіт", яка синхронізовано відображається партнеру.

1.7 Висновки за розділом

Проведений аналіз предметної області вказує на актуальність створення мобільного додатку для пар, який би поєднував гейміфікацію, поради, завдання, щоденні дії та синхронізацію між партнерами. Існуючі рішення не задовольняють повною мірою потреби сучасних користувачів у комплексному й захопливому інструменті підтримки стосунків. Запропонований підхід дозволить створити інноваційний застосунок, що стане практичним цифровим помічником у побудові гармонійних стосунків. Було сформульовано основні цілі та вимоги до додатку. Визначено, які функції реалізовані у межах MVP, а які плануються у майбутньому. Вхідні та вихідні дані, а також сценарії користування, дозволяють чітко уявити, як саме користувач взаємодіє з додатком. Отримані результати лягли в основу архітектурного проєктування системи.

2 АНАЛІЗ І ВИБІР МЕТОДІВ ТА ЗАСОБІВ РОЗРОБКИ

2.1 Вибір мови програмування та середовища розробки

При виборі мови програмування та технологій для розробки мобільного застосунку було проаналізовано кілька сучасних підходів до створення мобільних рішень. Нативна розробка (Kotlin для Android, Swift для iOS). Кросплатформні фреймворки: Flutter (Dart), React Native (JavaScript/TypeScript), Xamarin(.NET/C#), Kotlin Multiplatform.

Нативна розробка забезпечує найкращу продуктивність та гнучкість у використанні системних API. Однак вона вимагає окремої реалізації для кожної платформи (Android та iOS), що суттєво збільшує час і вартість розробки — особливо на стадії створення MVP.

React Native, хоча й дозволяє створювати кросплатформні застосунки за допомогою JavaScript, має певні обмеження у продуктивності (особливо для анімацій та складних інтерфейсів), а також часто потребує використання нативних модулів, що ускладнює підтримку проєкту [8].

Xamarin базується на технологіях Microsoft та дозволяє писати код мовою C#. Хоча він має повний доступ до API обох платформ, його екосистема поступово втрачає популярність, а сама технологія вже не розвивається так активно, як раніше [9].

Kotlin Multiplatform є перспективним, однак поки перебуває у відносно ранній стадії розвитку та не забезпечує повну кросплатформність для UI – розробка інтерфейсів все одно відбувається окремо для Android та iOS [10].

Таким чином, вибір Dart + Flutter є оптимальним для швидкої та ефективної реалізації MVP, забезпечуючи як якісний користувацький інтерфейс, так і можливість легкого розширення в майбутньому. Всі вище наведені порівняння за категоріями представлені в табличному вигляді у таблиці 2.1.

Таблиця 2.1 – Порівняльна характеристика інструментів для мобільної розробки

Критерії	Flutter (Dart)	React Native (JS/TS)	Нативна розробка (Kotlin/Swift)	Xamarin (.NET/C#)	Kotlin Multiplatform
Кросплатформність	Так, єдиний код для Android і iOS	Так, з можливістю нативних модулів	Ні, окремо Android та iOS	Так, але потребує обробки UI окремо	Часткова, UI — нативний для кожної платформи
Продуктивність	Висока, близька до нативної	Середня, залежить від мосту JS → натив	Максимальна (прямий доступ до API)	Висока, але складна підтримка	Висока, UI нативний
Швидкість розробки	Висока, Hot Reload, швидкий прототип	Висока, Hot Reload, багатий екосистем	Низька, два окремих проекти	Середня, складність в інтеграції	Середня, UI потребує окремої розробки
Гнучкість UI	Висока, кастомні анімації, власний рендеринг	Добра, обмежена нативними комп-нтами	Максимальна, нативні елементи UI	Добра, але залежить від платформи	Максимальна, UI нативний
Спільнота і підтримка	Активна, стрімко зростає	Дуже велика, багато пакетів	Велика, підтримка від Google/Apple	Зменшується, Microsoft фокусує увагу на MAUI	Розвивається, молодий проект
Навчання та складність	Легко вивчити, простий синтаксис Dart	Легко для JS-розробників	Вимагає знань платформ і мови	Середня, залежить від C# і .NET	Вимагає знань Kotlin та нативних UI
Підтримка MVP	Ідеальна, швидко та ефективно	Добра, швидко, але іноді треба більше правок	Повільна, потребує більше ресурсів	Помірна, залежить від досвіду	Помірна, UI окремий

У результаті порівняльного аналізу було обрано Flutter, як сучасне та стабільне рішення для кросплатформної мобільної розробки. Основні переваги Flutter [11]:

- повна уніфікація коду для Android та iOS, включно з інтерфейсом користувача;
- висока швидкість розробки завдяки механізму Hot Reload;
- гнучкий UI-фреймворк, який дозволяє легко реалізувати кастомізовані екрани та анімації;
- активна спільнота та широка екосистема пакетів;
- підтримка чистої архітектури, що дозволяє будувати масштабовані проєкти.

Мова Dart, яка використовується у Flutter, поєднує простоту синтаксису з підтримкою об'єктно-орієнтованого програмування. Вона розроблена з урахуванням потреб UI-розробки, забезпечує хорошу продуктивність та легко читається [11].

Як середовище розробки обрано Android Studio, оскільки вона має повноцінну підтримку Flutter, включає візуальний редактор UI, вбудований емулятор, інструменти налагодження та автоматизації збірки. Це дозволяє розробнику зосередитися на функціональності застосунку, не витрачаючи час на конфігурацію середовища [12].

2.2 Архітектурний підхід до проєкту

У процесі розробки мобільного застосунку ключовою вимогою стала масштабованість – здатність проєкту адаптуватися до розширення функціональності, зміни інтерфейсу, інтеграції з новими сервісами або навіть перенесення на інші платформи без необхідності повної переробки коду. Саме ця вимога зумовила вибір архітектурного підходу Clean Architecture [13] (рис. 2.1.).

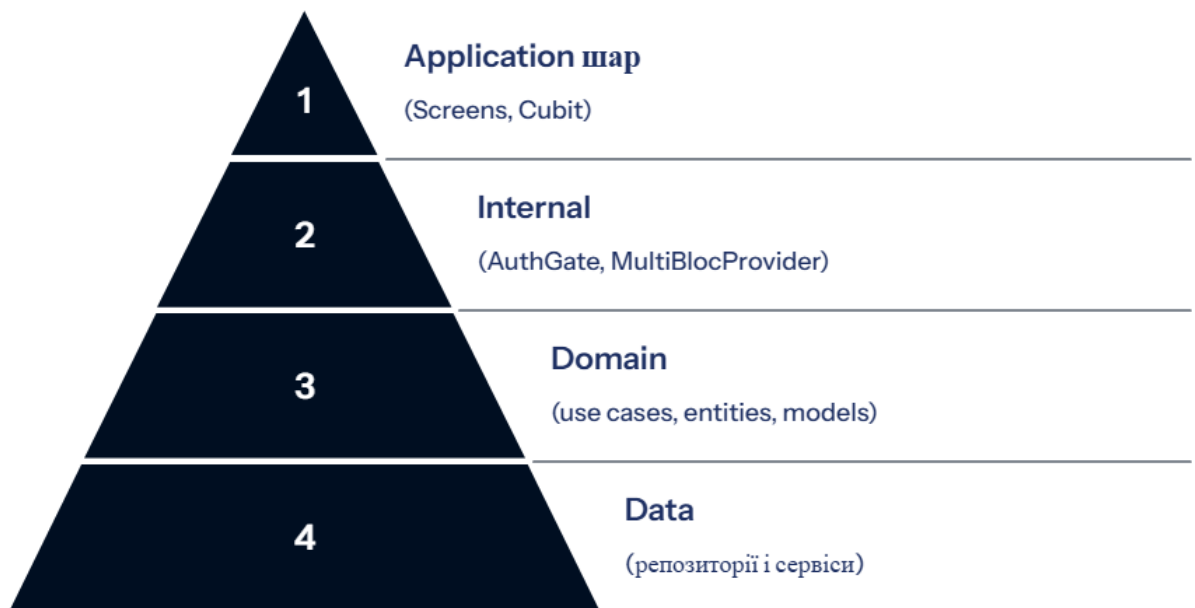


Рисунок 2.1 – Схематичне представлення архітектури

Оскільки застосунок планується не лише як дипломна робота, а як повноцінний продукт із потенціалом для подальшого розвитку, критично важливо:

- швидко додавати нові функції (ігрові механіки, типи завдань, внутрішню економіку тощо);
- змінювати інтерфейс без зміни логіки;
- легко адаптувати логіку синхронізації пари;
- додати нові джерела даних або перенести додаток на веб-платформу;
- clean architecture забезпечує таку гнучкість завдяки розділенню відповідальностей між шарами та суворому контролю залежностей.

Основні принципи Clean Architecture [13]:

- шарова структура: Domain Layer – бізнес-логіка, сутності, use cases; Data Layer – джерела даних, API, моделі; Presentation Layer – UI, керування станом (Cubit);
- залежності завжди спрямовані всередину – інтерфейси визначаються у внутрішніх шарах, а реалізації – у зовнішніх;

– інверсія залежностей – зовнішні ресурси (наприклад, бази даних або API) не впливають на бізнес-логіку.

Переваги Clean Architecture для проєкту зазначено в таблиці 2.2.

Таблиця 2.2 – Переваги Clean Architecture в контексті проєкту

Перевага	Значення для проєкту
Масштабованість	Дозволяє легко додавати нові функції без ризику для існуючих
Гнучкість	Можливість замінити інтерфейс або бекенд без змін у логіці
Підтримуваність	Чітке розділення обов’язків спрощує підтримку та налагодження
Тестованість	Легко покривати бізнес-логіку unit-тестами
Повторне використання коду	Деякі частини логіки можна використовувати в інших проєктах

Додаткові підходи:

- Cubit (Bloc) – стейт-менеджер, для ефективного керування станом у Presentation-шарі;
- Repository Pattern – забезпечує незалежність бізнес-логіки від джерел даних.

Таким чином, масштабованість, як стратегічна мета проєкту, напряду вплинула на вибір архітектури. Clean Architecture не лише забезпечує стабільну основу для MVP, а й відкриває широкі можливості для розширення функціоналу, переходу на інші платформи та підтримки високої якості коду впродовж усього життєвого циклу продукту.

2.3 Сторонні бібліотеки та сервіси

У процесі розробки мобільного застосунку було використано низку зовнішніх сервісів та бібліотек, які значно спростили реалізацію базових функцій, таких як автентифікація, зберігання даних та керування станом.

2.3.1 Firebase Authentication

Для забезпечення швидкої та безпечної реєстрації й входу користувачів було обрано Firebase Authentication. Цей сервіс є зручним і простим в інтеграції, а також надає гнучкі можливості для масштабування. [14]

Переваги:

- простота налаштування та документація;
- підтримка реєстрації за email та паролем, а також OAuth;
- високий рівень безпеки та надійність від Google.

2.3.2 Supabase – база даних та бекенд-сервіс

Supabase використовується як основна система для зберігання даних користувача – інформації про пари, завдання, поради, внутрішню валюту тощо [15].

Переваги:

- відкрита система на основі PostgreSQL;
- зручне RESTful API та бібліотека `supabase_flutter`;
- можливість швидко будувати прототип без створення власного бекенду.

Недоліки:

- потреба у ручному налаштуванні правил доступу (RLS) для безпечної взаємодії між користувачами;
- обмеження продуктивності у безкоштовному тарифі.

2.3.3 Flutter Bloc (Cubit) – керування станом

Для ефективного розділення логіки застосунку та інтерфейсу використовується бібліотека `flutter_bloc`, зокрема її полегшений варіант Cubit [16].

Застосування у проєкті:

- керування станом автентифікації;
- синхронізація між партнерами;
- обробка завдань, порад, внутрішньої валюти та кнопки “Say Hello”.

Переваги:

- простота структурування коду;
- підвищена тестованість логіки;
- чітке відокремлення станів від UI.

У процесі розробки було розглянуто декілька варіантів для реалізації ключових підсистем: автентифікації користувачів та зберігання даних (табл. 2.3 – 2.4). Остаточний вибір зупинився на Firebase Authentication для авторизації та Supabase для роботи з базою даних. Це рішення було прийнято після аналізу функціональних можливостей, простоти інтеграції, а також гнучкості й потенціалу масштабування.

Таблиця 2.3 – Порівняння Firebase та Supabase (автентифікація)

Критерій	Firebase Authentication	Supabase Auth
Простота інтеграції	Висока (готові SDK для Flutter)	Помірна (потребує налаштування RLS, логіки доступу)
Надійність	Висока, розробка Google	Висока, але новіша технологія
Документація та підтримка	Дуже широка	Обмеженіша
Підтримка OAuth	Є	Є
Анонімна автентифікація	Є	Немає
Мультиплатформеність	Так	Так
Гнучкість налаштувань доступу	Обмежена через Firebase rules	Дуже гнучка через RLS

Таблиця 2.4 – Порівняння Firebase та Supabase (зберігання даних)

Критерій	Supabase (PostgreSQL)	Firebase Firestore
Тип БД	Реляційна (SQL)	Документна (NoSQL)
Гнучкість запитів	Висока (SQL)	Середня (обмежені запити NoSQL)
Міграції та моделі	Повноцінна підтримка	Відсутні міграції
Робота зі зв'язками	Підтримка JOIN, зовнішніх ключів	Не підтримується
REST API	Генерується автоматично	Відсутнє, потрібна окрема логіка
Додаткові можливості	Вбудований SQL-редактор, Auth, Storage	Realtime, Hosting
Крива навчання	Помірна	Низька

2.4 Інструменти для управління проєктом

Для ефективної організації процесу розробки було використано набір інструментів, що забезпечували контроль над завданнями, відстеження прогресу, зберігання документації та ведення щоденника розробника.

Notion використовувався як головний інструмент для планування, організації та ведення документації [17]:

- створення баз знань, сторінок для фіксації ідей, нотаток та технічних рішень;
- гнучка система таблиць, баз даних і статусів дозволила реалізувати Kanban-дошку для управління завданнями (To Do, In Progress, Done);
- можливість швидко перемикається між форматами перегляду – списком, дошкою, календарем – полегшувала планування роботи;
- використовувався також як особистий щоденник розробника, куди заносилися думки, проблеми й рішення протягом усього процесу.

У майбутньому планується використання системи контролю версій Git разом із платформою GitHub для:

- збереження історії змін у коді;
- зручного ведення розробки з розділенням на гілки;

- відкритої публікації проєкту як прикладу реалізації повноцінного застосунку з використанням Flutter та сучасних підходів.

Репозиторій буде створено з відкритим доступом (public), що дозволить іншим розробникам ознайомитися з реалізацією, надати відгуки або використати його як базу для власних ідей.

Перші UI/UX рішення створювалися у вигляді ескізів на папері, але частину макетів пізніше було реалізовано у Figma:

- простота створення інтерактивних прототипів;
- візуалізація майбутніх екранів застосунку перед етапом розробки;
- можливість точно передати розміщення елементів, кольорову палітру та стилі.

2.5 Висновки за розділом

У ході аналізу було обрано мову програмування Dart та фреймворк Flutter як основні технології для розробки мобільного застосунку завдяки їх кросплатформеності, ефективності та активній спільноті. Архітектурний підхід із використанням Clean Architecture забезпечує модульність, масштабованість і полегшує подальший розвиток проєкту.

Для зберігання даних було обрано Supabase як надійне і гнучке рішення, що підтримує сучасні стандарти та дозволяє легко масштабувати проєкт. Для аутентифікації користувачів використовується сервіс Firebase, який забезпечує безпечну, швидку і зручну інтеграцію.

Управління проєктом здійснюється за допомогою сервісу Notion, що дозволяє організувати робочий процес, планувати задачі та відслідковувати прогрес. У майбутньому планується впровадження системи контролю версій Git та публікація проєкту на платформі GitHub для покращення співпраці та доступності коду.

Таким чином, вибір інструментів і методів розробки відповідає вимогам до функціональності, зручності підтримки та масштабованості застосунку.

3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Загальна структура застосунку

Застосунок побудований відповідно до принципів Clean Architecture(рис. 2.1), яка забезпечує чітке розділення відповідальностей між різними частинами системи. Загалом структура проєкту включає понад 60 окремих Dart-класів (табл. 3.1), кожен з яких виконує чітко визначену функцію у відповідному шарі архітектури.

Таблиця 3.1 – Статистика структури проєкту (на момент створення MVP)

Категорія класів	Кількість	Коментар
Класи-екрани (screens)	9	Реалізують основні UI-сторінки застосунку
Класи-стану (Cubit, State)	11	Управління логікою авторизації, задач, порад тощо
Сервісні класи	7	Взаємодія з Supabase, Firebase, зберігання локальних даних
Інтерфейси репозиторіїв	7	Оголошення контрактів між domain та data
DTO-класи (представлення даних)	4	Конвертація між даними з БД та доменними сутностями
Use-case класи	2	Інкапсуляція бізнес-логіки
Сутності (Entities)	4	Основні бізнес-об'єкти: Task, Advice, Partner тощо
Виділені UI-компоненти (Widgets)	12	Повторно використовувані компоненти інтерфейсу

Структура має такі основні шари перелічені далі.

Presentation (Інтерфейс користувача) – містить усі екрани, віджети та логіку відображення. Саме тут реалізована взаємодія користувача з додатком.

Internal (Внутрішній шар) – містить інфраструктурні компоненти, такі як MultiBlocProvider (або MultiProvider), які забезпечують доступ до Cubit/Bloc

для всіх підлеглих екранів. Також включає компонент AuthGate, що відповідає за перевірку авторизації користувача і перенаправлення на відповідні екрани.

Domain (Бізнес-логіка) – реалізує основні правила і сценарії роботи застосунку: синхронізацію пари, керування завданнями, порадами, монетами тощо. Містить сутності, інтерфейси репозиторіїв та use case.

Data (Робота з даними) – реалізує доступ до зовнішніх джерел даних, таких як база даних Supabase та авторизаційний сервіс Firebase. Включає реалізації репозиторіїв, DTO-моделі, джерела даних (remote data sources).

Така структура забезпечує модульність, тестованість і зручне масштабування проєкту, що особливо важливо для потенційного розширення функціональності в майбутньому.

На рисунку 3.1 зображена діаграма, яка описує кількість спроектованих класів за категорією.

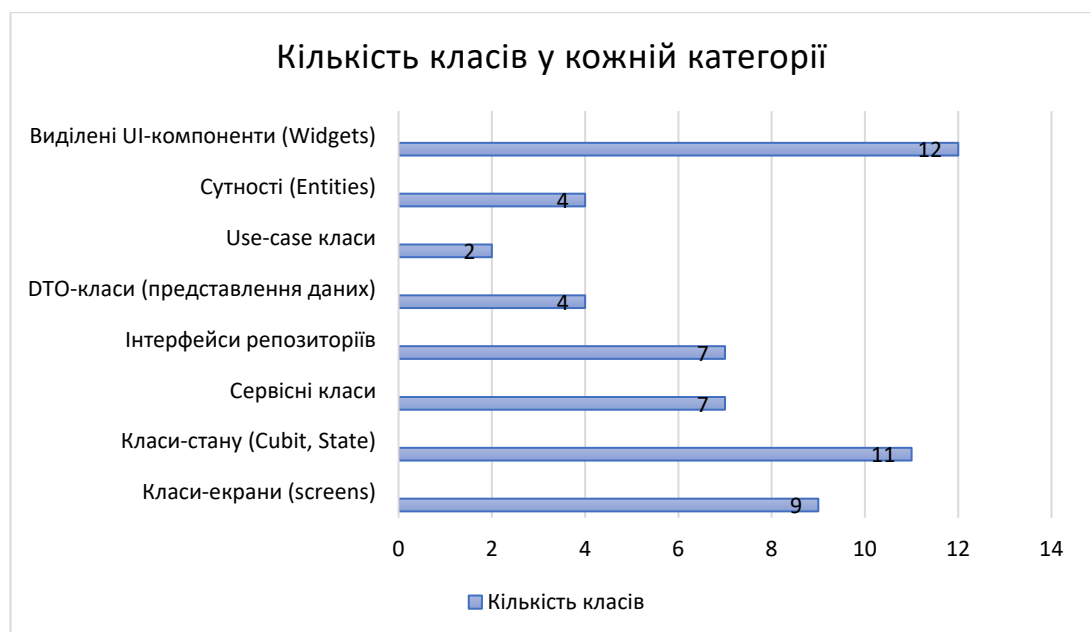


Рисунок 3.1 – Кількість спроектованих класів у системі за категоріями

3.2 UI/UX-дизайн та ескізи екранів

Проектування інтерфейсу користувача (UI) та досвіду користувача (UX) є критично важливою частиною розробки застосунку, особливо у

випадку соціально-орієнтованих сервісів, що спрямовані на щоденне використання. Основна мета дизайну в цьому проєкті – забезпечити інтуїтивну навігацію, приємний візуальний стиль і емоційно позитивний досвід взаємодії з додатком для обох партнерів.

Основні принципи UX-дизайну:

- простота – користувачі мають швидко розуміти, як працює застосунок без додаткових інструкцій;
- емоційність – дизайн має викликати приємні асоціації, сприяти комунікації та створювати відчуття «домашності» та близькості;
- синхронізація – передбачено обов’язкове з’єднання з обліковим записом партнера, що дозволяє взаємодіяти з функціоналом програми разом;
- гейміфікація – реалізовано внутрішню валюту, яка нагороджує за виконання завдань, натхненні поради та «сказати Привіт» для збереження щоденного контакту.

У межах проєкту було створено низку ескізів інтерфейсу, які демонструють логіку переходів та розташування ключових елементів. Дизайн екранів виконано з урахуванням мобільного використання, тому застосовано вертикальне компонування, великі інтерактивні зони та м’яку кольорову палітру.

Екрани авторизації та синхронізації - дозволяє користувачеві увійти в акаунт, а також створити або приєднатись до пари. Стан пари зберігається у віддаленій базі даних, що забезпечує синхронізацію між двома пристроями.

Домашній екран - містить щоденне завдання, поради, кнопку «Сказати привіт» та баланс валюти. Дизайн побудований навколо карток, які користувач може гортати або натискати для деталей.

Екран завдань – відображає доступні завдання за категоріями. Завдання можна позначати як виконані, після чого пара отримує бонус у внутрішній валюті.

Екран з порадами - показує короткі психологічні або емоційні підказки, які сприяють покращенню комунікації. Поради змінюються щодня.

Візуальний стиль:

- обрана палітра складається з теплих пастельних тонів, що не перевантажують зір і викликають емоційну теплоту;
- використано сучасні шрифти для легкості читання;
- мінімалістичні, стилізовані під мобільні гайдлайни Android та iOS;
- застосовано легкі анімаційні переходи між екранами для плавності UX.

Усі екрани (рис. 3.3-3.7) були попередньо спроектовані у вигляді ескізів у графічному редакторі Figma. Це дозволило оперативно протестувати логіку переходів між екранами, розташування елементів інтерфейсу та швидко внести зміни на етапі прототипування перед реалізацією.

3.3 Проектування бази даних

Для забезпечення надійного зберігання та ефективного доступу до даних у розробленому застосунку використовується хмарна реляційна база даних, розгорнута на платформі Supabase. Цей сервіс надає готову інфраструктуру на основі PostgreSQL та дозволяє швидко інтегрувати бекенд із Flutter-застосунком через REST або RPC-запити.

У проєкті було створено кілька ключових таблиць, які забезпечують зберігання та взаємозв'язок усіх сутностей системи. Нижче в таблицях 3.2 -3.8 подано короткий опис найважливіших з них:

Таблиця 3.2 – Структура таблиці users (Користувачі)

Поле	Тип	Опис
email	text	Адреса електронної пошти користувача
username	text	Ім'я користувача у системі
created_at	timestamp	Дата створення облікового запису
firebase_uid	text	Унікальний ідентифікатор з Firebase

Таблиця 3.3 – Структура таблиці partners (Партнерські зв'язки)

Поле	Тип	Опис
id	int8	Первинний ключ
user_uid	text	UID поточного користувача
partner_uid	text	UID партнера
sync_code	text	Код для синхронізації
is_synced	bool	Стан синхронізації (true — синхронізовано)
created_at	timestamp	Дата створення запису
updated_at	timestamp	Дата останнього оновлення
hearts_count	int4	Кількість накоплених «сердеч» у пари

Таблиця 3.4 – Структура таблиці tasks (Завдання)

Поле	Тип	Опис
id	int8	Первинний ключ
name	text	Назва завдання
description	text	Опис завдання
base_rate	numeric	Базова нагорода (в балах)
created_at	timestamp	Дата створення завдання

Таблиця 3.5 – Структура таблиці tasks_log (Журнал завдань)

Поле	Тип	Опис
id	int8	Первинний ключ
task_id	int8	Ідентифікатор завдання (tasks.id)
sender_uid	text	UID користувача, який надіслав завдання
receiver_uid	text	UID користувача, який отримав завдання
task_status	text	Статус завдання (напр. 'pending', 'done')
sender_confirmed	bool	Підтвердження відправника
receiver_confirmed	bool	Підтвердження отримувача

Таблиця 3.6 – Структура таблиці daily_tips (Щоденні поради)

Поле	Тип	Опис
id	int8	Первинний ключ
tip	text	Текст поради

Таблиця 3.7 – Структура таблиці daily_tips_log (Журнал порад)

Поле	Тип	Опис
id	int8	Первинний ключ
user_uid	text	UID користувача, який отримав пораду
tip_id	int8	Ідентифікатор поради (daily_tips.id)
created_at	timestamp	Дата отримання поради

Таблиця 3.8 – Структура таблиці likes Привітання(Say Hello)

Поле	Тип	Опис
id	int8	Первинний ключ
sender_uid	text	UID користувача, який надіслав “лайк”
receiver_uid	text	UID користувача, якому надіслано “лайк”
created_at	timestamp	Дата взаємодії
mutual	bool	Ознака взаємної симпатії (true — якщо обидва лайкнули)

На рисунку 3.2 зображено схематичне представлення спроектованої бази даних.

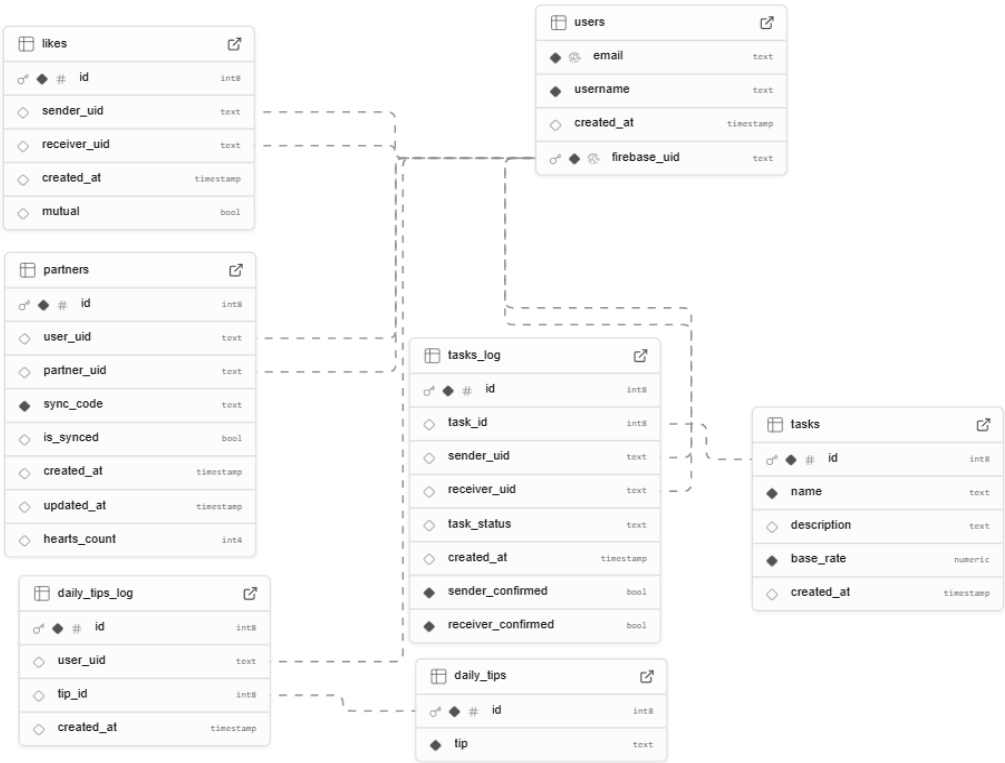


Рисунок 3.2 – Схематичне представлення бази даних

Зв'язки реалізовані через текстові поля *_uid, які відповідають унікальним ідентифікаторам у таблиці users. Основні залежності:

- partners.user_uid → users.firebase_uid;
- partners.partner_uid → users.firebase_uid;
- tasks_log.sender_uid / receiver_uid → users.firebase_uid;
- tasks_log.task_id → tasks.id;
- daily_tips_log.tip_id → daily_tips.id;
- likes.sender_uid / receiver_uid → users.firebase_uid.

Це дозволяє масштабувати систему без жорстких зовнішніх ключів, водночас зберігаючи логічну цілісність на рівні додатку.

Особливості:

- гнучка синхронізація: реалізована через таблицю partners, що дозволяє користувачам надсилати запити на зв'язок за допомогою коду sync_code;
- відстеження дій: таблиці tasks_log, daily_tips_log, likes дають змогу фіксувати взаємодії користувачів та адаптувати контент;
- простота моделі: структура таблиць збережена максимально простою для швидкої розробки MVP.

Проектована база даних повністю відповідає потребам MVP, дозволяє реалізувати всі ключові функції застосунку та залишається гнучкою для подальшого розширення системи в наступних версіях продукту.

3.4 Організація авторизації користувачів

Процес авторизації є критично важливою частиною функціональності застосунку, оскільки він забезпечує індивідуальний доступ до даних, синхронізацію з партнером та захист персональної інформації. Для реалізації цієї частини системи було обрано Firebase Authentication, оскільки ця технологія надає готові та безпечні механізми авторизації. На рисунках 3.3-3.4 наведено побудовані екрани для авторизації і реєстрації користувачів.

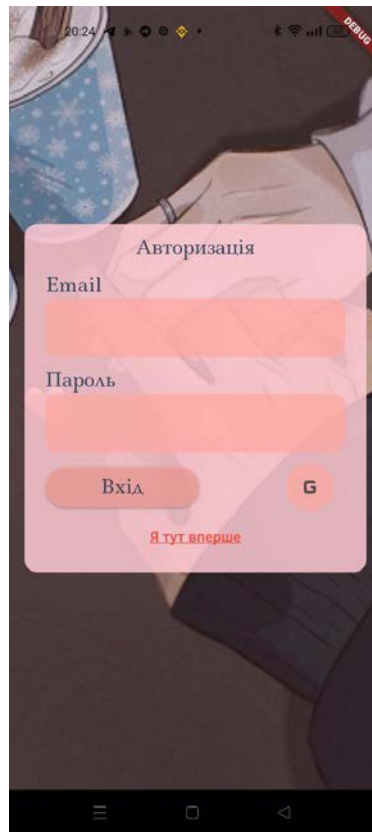


Рисунок 3.3 – Екран авторизації

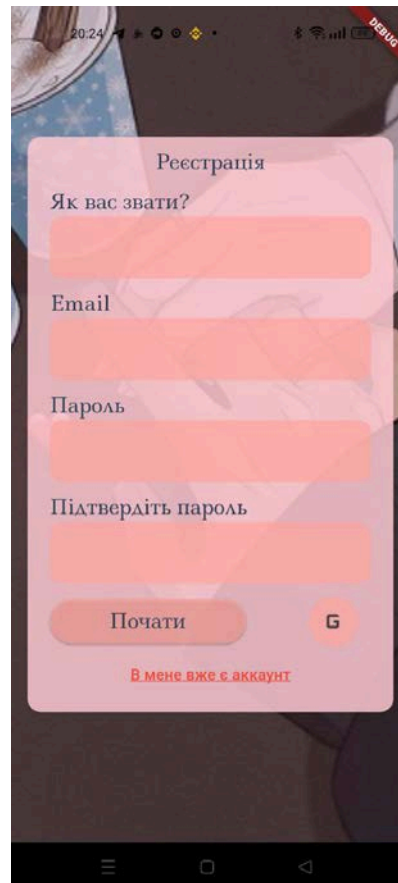


Рисунок 3.4 – Екран Реєстрації

3.4.1 Підтримувані способи входу

У застосунку реалізовано два способи авторизації:

- вхід за email та паролем;
- вхід через обліковий запис Google (OAuth 2.0).

Це дозволяє надати користувачу гнучкість у виборі способу реєстрації, а також пришвидшити перший вхід за допомогою Google-акаунта.

3.4.2 Архітектурна реалізація

Функціональність авторизації побудована з урахуванням принципів Clean Architecture:

- Domain-шар містить абстрактний інтерфейс `AuthService`, який визначає методи реєстрації, входу, виходу та перевірки стану;
- Data-шар реалізує `AuthServiceImpl`, який працює з Firebase Auth SDK;
- Presentation-шар реалізований через `AuthCubit`, який управляє станами `AuthInitial`, `AuthLoading`, `AuthAuthenticated`, `AuthError` тощо.

3.4.3 Основні функції авторизації

Авторизація є критично важливим компонентом будь-якого застосунку, який передбачає персоналізовану взаємодію з користувачем. Вона забезпечує контроль доступу до персональних даних, дозволяє реалізувати індивідуальні сценарії взаємодії та гарантує безпеку користувача у цифровому середовищі. У контексті розроблюваного застосунку, авторизація є не лише засобом ідентифікації користувача, але й основою для реалізації багатьох інших функціональних можливостей, таких як синхронізація даних, збереження прогресу, доступ до спільного контенту тощо.

Для забезпечення зручності та гнучкості входу до системи, у застосунку реалізовано декілька основних сценаріїв авторизації, зокрема реєстрацію через email і пароль, вхід через Google-акаунт, автоматичний вхід та вихід із системи. Ці механізми дозволяють користувачеві швидко відновити сесію, забезпечують збереження ідентифікаторів, а також інтеграцію з Firebase для централізованого управління обліковими записами.

Основні функції авторизації представлені нижче (табл. 3.9).

Таблиця 3.9 – Основні функції авторизації

Функція	Опис
Реєстрація користувача	Створення нового акаунта за email та паролем, збереження <code>firebase_uid</code> у таблиці <code>users</code> .
Вхід до системи	Вхід за email та паролем, отримання сесії, збереження UID локально.
Вхід через Google	Вхід за допомогою Google OAuth, автоматична реєстрація нового користувача при першому вході.
Автоматичний вхід	Перевірка наявності активної сесії при запуску застосунку.
Вихід із системи	Виклик <code>signOut()</code> у Firebase та очищення локального стану.

3.4.4 Складності реалізації

Найбільші труднощі виникли під час обробки винятків Firebase, зокрема для входу через Google, де додатково потрібно було правильно налаштувати OAuth-клієнт у Firebase Console. Також управління станами авторизації, що вимагало чіткої синхронізації між UI та логікою. Та забезпечення коректного збереження та передачі UID, необхідного для запису даних у базу.

Також окрема увага приділялася UX: користувачу надаються повідомлення про помилки (наприклад, "невірний пароль", "акаунт не існує" або "не вдалося авторизуватись через Google").

Приклад потоку:

- користувач обирає спосіб входу (email або Google);
- AuthCubit викликає відповідний метод у AuthRepository;
- у разі успішного входу отримується `firebase_uid`, який використовується для збереження або оновлення даних користувача у таблиці `users`;
- стан застосунку змінюється на `AuthAuthenticated`, і користувач потрапляє на головний екран.

3.5 Модель синхронізації партнерів

Синхронізація акаунтів двох користувачів – це основна функція застосунку, що дозволяє створити зв'язок між партнерами для спільного використання функціоналу: обміну завданнями, повідомленнями, використання внутрішньої валюти тощо. На рисунках 3.5 – 3.6 зображено екрани синхронізації користувачів.



Рисунок 3.5 – Екран синхронізації

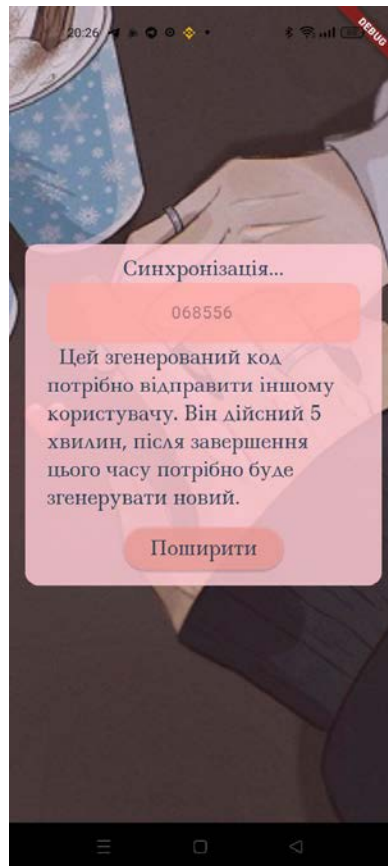


Рисунок 3.6 – Екран отримання коду синхронізації

3.5.1 Принцип роботи

Механізм синхронізації базується на одноразовому шестизначному коді, який генерується одним з користувачів та передається іншому.

Генерація коду:

- користувач натискає кнопку "Створити код пари";
- у базі даних створюється запис з шестизначним кодом, UID ініціатора, часовою міткою створення;
- код дійсний 5 хвилин.

Підключення партнера:

- інший користувач вводить отриманий код у відповідному полі;
- система перевіряє наявність запису з таким кодом і перевіряє його строк дії;

- у разі успішного підтвердження, обидва користувачі отримують `partner_id`, який вказує на UID одне одного.

Очистка коду:

- після встановлення зв'язку непотрібні записи видаляються;
- якщо термін дії минув – код автоматично вважається недійсним.

У процесі синхронізації партнерів були реалізовані механізми перевірки даних для запобігання некоректній або дубльованій інформації в базі даних.

Користувач не може синхронізуватися сам із собою. Це перевіряється на етапі надсилання коду синхронізації – якщо `user_uid == partner_uid`, синхронізація блокується.

Код синхронізації дійсний лише 5 хвилин з моменту створення. Якщо спроба приєднання відбувається після закінчення терміну дії, вона відхиляється з відповідним повідомленням про помилку.

Після створення нового запису перевіряється, чи дублюється активна пара. У випадку дублювання, а це можливо якщо два користувача згенерували код для синхронізації: після об'єднання двох користувачів у пару – один запис видаляється.

Після успішної синхронізації запис оновлюється відповідно до стану `is_synced = true`, щоб уникнути повторного використання коду.

Переваги моделі:

- простота використання: користувачам достатньо передати короткий код;
- безпека: код не зберігає жодної персональної інформації, короткочасний;
- швидка перевірка: система легко обробляє запити завдяки простій структурі таблиці.

Обрана модель синхронізації з короткоживучим кодом є зручною та безпечною для користувачів MVP-версії застосунку. У майбутньому можна

розширити логіку – додати повідомлення про запити, історію пар або можливість розірвання зв'язку.

3.6 Реалізація завдань та порад

Важливою частиною застосунку є функціональність, яка стимулює активну взаємодію між партнерами – через виконання завдань і отримання щоденних порад. Ці механізми служать як інструменти для покращення стосунків, підвищення рівня довіри, залученості та емоційної близькості між користувачами. Як завдання, так і поради відображаються на головному екрані користувачів (рис. 3.7).



Рисунок 3.7 – Головний екран застосунку.

3.6.1 Щоденні поради (Daily Tips)

Система щоденних порад створена для того, щоб користувачі регулярно отримували невеликі рекомендації або нагадування, які допомагають будувати тепліші та довірчіші стосунки. Кожна порада є коротким текстовим повідомленням і подається випадковим чином щодня.

Логіка:

- при відкритті застосунку система перевіряє, чи вже надавалась порада за поточну дату;
- якщо поради ще не було – вибирається випадкова з бази;
- запис створюється у `daily_tips_log`.

3.6.2 Завдання для партнерів (Tasks)

Завдання – це механізм активної взаємодії між партнерами. Вони мають на меті заохочувати спільну діяльність, романтичні жести або прості вчинки, що сприяють емоційному зв'язку.

У поточній версії застосунку реалізовано просту логіку: щодня кожному користувачу випадковим чином призначається одне завдання зі заздалегідь сформованого списку

Завдання зберігаються у відповідній колекції бази даних (`daily_tasks`). Під час відкриття застосунку користувач отримує випадково обране завдання на день. У майбутньому передбачено вдосконалення цієї моделі шляхом:

- персоналізації (на основі історії взаємодій);
- синхронізації завдань між партнерами (одне спільне завдання на двох);
- можливості оцінювати або змінювати завдання;
- формування статистики виконання завдань та отримання заохочень (наприклад, збільшення нагороди, подарунки, тощо.).

3.7 Функція “Say Hello” та внутрішня валюта

У сучасних цифрових застосунках, особливо тих, що орієнтовані на покращення міжособистісних відносин, важливу роль відіграють прості засоби для підтримки комунікації та залучення користувачів. Функція “Say Hello” є таким елементом, який дозволяє партнерам легко і швидко взаємодіяти, висловлюючи увагу і турботу один про одного. Щоб підсилити мотивацію користувачів до регулярного використання цієї функції, у застосунку впроваджено внутрішню валюту – ігровий механізм, що стимулює активність і позитивну взаємодію.

3.7.1 Функція “Say Hello”: концепція і соціальний вплив

“Say Hello” – це простий інструмент, який дозволяє користувачам надіслати партнеру коротке привітання або сигнал уваги у вигляді текстового повідомлення, анімації або іконки. Вона знаходиться на головному екрані користувачів (рис. 3.7), і зручна для використання.

Підтримка регулярної комунікації. Навіть просте “привіт” може стати приводом для розмови, нагадати про себе, зменшити відстань між партнерами.

Зміцнення емоційного зв’язку. Регулярні взаємодії підтримують відчуття близькості, що особливо важливо у відносинах на відстані.

Простота та швидкість. Не потребує багато часу, доступна у будь-який момент.

Ця функція знижує бар’єр для спілкування, що є однією з причин конфліктів або відчуження у стосунках.

3.7.2 Внутрішня валюта: мотивація через гейміфікацію

У проєкті передбачена внутрішня валюта – «Сердечка», яка має слугувати мотиваційним механізмом для заохочення користувачів до активної

взаємодії із застосунком. Вона нараховується користувачам за виконання певних дій, зокрема:

- виконання завдань;
- за використання функції "Say Hello".

Наразі бекенд підтримує лише механізми нарахування валюти, а функції її витрачання і відповідні транзакції ще перебувають у стадії розробки. Це означає, що користувачі можуть накопичувати "Сердечка", але поки не можуть їх витратити на внутрішні покупки чи бонуси.

3.7.3 Потенційне використання валюти

Хоча витрачання валюти ще не реалізовано, у майбутніх версіях планується впровадження таких можливостей:

- обмін "Сердечок" на унікальні візуальні подарунки, анімації, персоналізовані повідомлення;
- розблокування нових функціональних модулів та контенту;
- підвищення рівня взаємодії та гейміфікація, що стимулюватиме користувачів частіше повертатися до застосунку.

Реалізація цих функцій потребує додаткової роботи над бекендом – розробки сервісів для списання валюти, а також оновлення UI/UX.

3.7.4 Технічна реалізація (поточний стан)

Нарахування валюти. Реалізовано через бекенд API, яке оновлює баланс користувача при успішному виконанні завдань.

Зберігання балансу. Баланс "Сердечок" зберігається в базі даних і синхронізується із клієнтом.

Відображення валюти. Інтерфейс користувача показує поточний баланс валюти.

Відсутність списання. Функціонал списання та витрачання валюти наразі не реалізований, тому всі операції є беззбитковими для користувача.

3.7.5 Примітки щодо подальшого розвитку

Для повноцінної гейміфікації необхідно розробити механізми витрачання валюти, включно з відповідною логікою на бекенді, обробкою транзакцій та захистом від можливих шахрайств.

Планується інтеграція функції “Say Hello” із внутрішньою валютою – наприклад, "Say Hello" може стати платною або безкоштовною з бонусним нарахуванням "Сердечок".

Необхідно передбачити UX/UI для інформування користувача про баланс та можливості валюти, а також про відсутність функції витрачання у поточній версії.

3.8 Тестування MVP

Метою тестування MVP було підтвердити стабільність, коректність роботи ключових функцій та відповідність системи вимогам предметної області. Тестування проводилося автором проєкту із застосуванням активного логування, що дало змогу детально відстежувати процеси всередині застосунку. Результати всіх проведених тестів наведено в таблиці 3.10.

Застосунок працює стабільно і без збоїв у всіх реалізованих модулях, що підтверджує якість розробленого коду і правильність вибраної архітектури.

Функція “Say Hello” безперебійно виконує свої завдання.

Автентифікація та реєстрація користувачів працюють коректно та швидко.

Синхронізація даних між акаунтами здійснюється без помилок, надійно і стабільно.

Таблиця 3.10 – Результати тестування застосунку

Функція	Очікуваний результат	Результат	Примітка
Реєстрація та вхід	Користувач може зареєструватися та увійти в акаунт	Успішно	-
Синхронізація з партнером	Користувач бачить ім'я партнера після введення коду	Успішно	Синхронізація працює
Кнопка «Привітатись»	Після взаємного натискання додається 1 сердечко	Успішно	Обмеження 1 раз/день
Відображення завдання	Користувач бачить завдання на сьогодні	Успішно	-
Початок завдання	Завдання активується і зберігається статус	Успішно	-
Порада дня	Щоденно відображається нова порада	Успішно	-
Вихід із системи	Користувач може вийти з акаунта	Успішно	-

Механізм нарахування внутрішньої валюти повністю відповідає поставленим вимогам.

Інтерфейс користувача є інтуїтивним і зручним для цільової аудиторії.

Активне логування допомогло підтвердити відсутність технічних проблем у роботі системи.

Хоча на поточному етапі всі технічні модулі працюють стабільно, рекомендовано розширити тестування залученням реальних користувачів для оцінки юзабіліті та перевірки застосунку в різних реальних умовах

експлуатації. Це допоможе виявити можливі шляхи для подальшого вдосконалення інтерфейсу та функціоналу.

Також планується розробити та протестувати механізми витрачання внутрішньої валюти, що стане наступним кроком у розвитку застосунку.

3.9 Інструкція користувача

Мобільний застосунок, розроблений у межах даного проєкту, орієнтований на пари, які бажають покращити емоційну близькість, комунікацію та спільно проводити час у цікавій гейміфікованій формі. Інтерфейс інтуїтивно зрозумілий і не потребує спеціальних технічних знань.

3.9.1 Встановлення застосунку

На момент написання роботи застосунок доступний у вигляді APK-файлу. Його можна встановити наступним чином:

- завантажити APK-файл із посилання (буде додано після розміщення);
- надати дозволи на встановлення з невідомих джерел (у разі потреби);
- встановити застосунок на смартфон з операційною системою Android (версія 8.0 і вище);
- відкрити застосунок та пройти первинну авторизацію.

3.9.2 Реєстрація та вхід

Після запуску користувач потрапляє на екран авторизації.

Необхідно зареєструватися за допомогою електронної пошти та пароля, або виконати вхід через гугл акаунт.

Після реєстрації система запропонує створити акаунт пари – шляхом обміну унікальним кодом із партнером.

3.9.3 Основні функції застосунку

Завдання:

- кожного дня користувачі отримують нове завдання для пари;
- завдання розраховані на зміцнення зв'язку, розвиток довіри або спільне проведення часу;
- після виконання завдання обидва партнери можуть позначити його як виконане.

Поради: розділ містить корисні психологічні поради для покращення комунікації та підтримки у парі.

Функція “Say Hello”:

- безоплатна функція, доступна один раз на день для кожного користувача;
- якщо обидва партнери протягом дня надіслали одне одному привітання, вони отримують +1 «серце» – умовну нагороду за взаємність та активність.

Внутрішня валюта:

- за активність у застосунку (виконання завдань, взаємодію з партнером) користувачі отримують “серця”;
- “Серця” використовуються для розблокування нових функцій і візуальних елементів.

Для того, щоб вручну оновити інформацію (наприклад, завдання, поради чи поточну активність партнера), користувач може свайпнути вниз по екрану. Цей жест ініціює повторне завантаження даних із сервера.

ВИСНОВКИ

У рамках даної роботи було розроблено MVP мобільного застосунку, який покликаний підтримувати романтичні стосунки за допомогою функцій синхронізації акаунтів партнерів, внутрішньої валюти та інтерактивної функції “Say Hello”. Основні результати та досягнення роботи полягають у наступному:

- реалізовано стабільну систему автентифікації та реєстрації користувачів, що забезпечує безпеку та зручність доступу до застосунку;
- розроблено механізм нарахування внутрішньої валюти, який є фундаментом для мотивації користувачів виконувати завдання у застосунку;
- створено функцію “Say Hello”, що слугує інструментом для підтримки живої комунікації між партнерами та підвищення емоційної взаємодії;
- запроваджено надійну синхронізацію даних між акаунтами партнерів, що забезпечує єдність інформації та спільний досвід користування;
- проведене тестування MVP підтвердило високу стабільність і коректність роботи основного функціоналу, що є важливою передумовою для подальшого масштабування та розвитку застосунку;
- визначено напрями подальшого розвитку проєкту, зокрема: впровадження механізмів витрачання внутрішньої валюти, розширене юзабіліті-тестування, автоматизоване тестування та удосконалення інтерфейсу.

Отже, поставлені завдання було успішно виконано, MVP продемонстрував працездатність і відповідає визначеним технічним вимогам. Розроблений застосунок має потенціал для подальшого розвитку та комерційного впровадження з урахуванням рекомендацій щодо вдосконалення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Типи міжособистісних відносин [Електрон. ресурс]. – Режим доступу: <https://vseosvita.ua/library/embed/0100aglm-e80a.doc.html>.
2. Between – Private Couples App [Electronic resource]. – Access mode: <https://play.google.com/store/apps/details?id=kr.co.vcnc.android.couple>.
3. Couple – Private Couples App [Electronic resource]. – Access mode: <https://play.google.com/store/apps/details?id=com.ibillstudio.thedaycouple>.
4. Love Nudge [Electronic resource]. – Access mode: <https://play.google.com/store/apps/details?id=com.grootersproductions.challenge>.
5. Agapé: Feel Close When Apart [Electronic resource]. – Access mode: <https://play.google.com/store/apps/details?id=com.getdailyagape>.
6. Coral: Couples & Relationship [Electronic resource]. – Access mode: <https://play.google.com/store/apps/details?id=app.getcoral.m>.
7. Paired: Couples & Relationship [Electronic resource]. – Access mode: <https://play.google.com/store/apps/details?id=com.getpaired.app>.
8. React Native: Getting Started [Electronic resource]. – Access mode: <https://reactnative.dev/docs/getting-started>
9. Xamarin documentation [Electronic resource]. – Access mode: <https://learn.microsoft.com/ru-ru/previous-versions/xamarin>.
10. Kotlin Multiplatform quickstart [Electronic resource]. – Access mode: <https://kotlinlang.org/docs/multiplatform.html>.
11. Офіційна документація Flutter [Електрон. ресурс]. – Режим доступу: <https://docs.flutter.dev>.
12. Android Studio [Electronic resource]. – Access mode: <https://developer.android.com/studio>.
13. Martin R. The Clean Architecture [Electronic resource] / R. Martin. – Access mode: <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>.

14. Офіційна документація Firebase [Електрон. ресурс]. – Режим доступу: <https://firebase.google.com/docs>.
15. Офіційна документація Supabase [Електрон. ресурс]. – Режим доступу: <https://supabase.com/docs>.
16. Офіційна документація Flutter bloc [Електрон. ресурс]. – Режим доступу: https://pub.dev/packages/flutter_bloc.
17. Notion для управління проєктами: офіційний сайт [Електрон. ресурс]. – Режим доступу: <https://www.notion.so>.

ДОДАТОК А
Текст програми

A.1 Файл main_screen.dart

```

class MainScreen extends StatefulWidget {
  final DomainUser user;
  const MainScreen({super.key, required this.user});

  @override
  State<MainScreen> createState() => _MainScreenState();
}

class _MainScreenState extends State<MainScreen> {

  @override
  void initState() {
    log("MainScreen: initState called");
    _initUserData();
    super.initState();
  }

  Future<void> _refreshData() async {

    _initUserData();
    setState(() {});
    return Future<void>.delayed(const Duration(seconds: 3));
  }

  Future<void> _initUserData() async {
    final uid = widget.user.firebase_uid;

```

```

await Future.wait([
  context.read<DailyTipCubit>().loadTodaysTip(uid),
  context.read<PartnerCubit>().loadPartnerData(uid),
  context.read<DailyTaskCubit>().loadTask(uid),
]);
if (!mounted) return;
}

```

```
@override
```

```

Widget build(BuildContext context) {
  return Scaffold(
    extendBodyBehindAppBar: true,
    appBar: _buildAppBar(widget.user),
    body: Stack(
      children: [
        // Основной контент
        _buildBody(widget.user),

        // Свайп-зона сверху (не перекрывает весь экран)
        Align(
          alignment: Alignment.topCenter,
          child: SizedBox(
            height: 200, // тільки верхня частина
            child: IgnorePointer(
              ignoring: false, // дозволяє ловити свайп
              child: SwipeToRefreshArea(onRefresh: _refreshData),
            ),
          ),
        ),
      ],
    ),
  );
}

```

```

    ),
  );
}

```

```

AppBar _buildAppBar(DomainUser user) {
  return AppBar(
    centerTitle: true,
    backgroundColor: AppStyles.mainColor,
    leading: PopupMenuButton<int>(
      icon: const Icon(Icons.account_circle, size: 40),
      onSelect: (value) {
        if (value == 2) {
          context.read<AuthCubit>().signOut();
          Navigator.pushAndRemoveUntil(
            context,
            MaterialPageRoute(builder: (_) => LoginScreen()),
            (_) => false,
          );
        }
      },
      itemBuilder: (context) => [
        const PopupMenuItem(value: 1, child: Text("...")),
        PopupMenuItem(
          value: 2,
          child: Row(
            children: const [
              Icon(Icons.exit_to_app, size: 20),
              SizedBox(width: 10),
              Text("Вийти")
            ],
          ),
        ),
      ],
    ),
  );
}

```

```

    ),
  ),
],
),
title: Text(user.username, style: AppStyles.titleText),
actions: [
  Padding(
    padding: const EdgeInsets.only(right: 5),
    child: BlocBuilder<PartnerCubit, PartnerState>(
      builder: (context, state) {
        if (state is PartnerLoaded) {
          return Text("${state.hearts} ♥ ", style: AppStyles.titleText);
        } else {
          return const SizedBox(
            width: 20,
            height: 20,
            child: CircularProgressIndicator(strokeWidth: 2),
          );
        }
      },
    ),
  ),
],
);
}

```

```

Widget _buildBody(DomainUser user) {
  return Container(
    decoration: BoxDecoration(
      image: DecorationImage(

```

```

image: const AssetImage("assets/images/bg.png"),
fit: BoxFit.cover,
colorFilter: ColorFilter.mode(
  const Color.fromRGBO(255, 202, 212, 0.15),
  BlendMode.color,
),
),
),
child: Column(
  children: [
    const SizedBox(height: 120),
    _buildPartnerCard(user),
    const SizedBox(height: 15),
    DailyTaskCard(userId: user.firebase_uid),
    const SizedBox(height: 15),
    _buildDailyTipCard(user),
  ],
),
);
}

```

```

Widget _buildPartnerCard(DomainUser user) {
  return Container(
    margin: const EdgeInsets.symmetric(horizontal: 15),
    padding: const EdgeInsets.symmetric(vertical: 10, horizontal: 20),
    decoration: BoxDecoration(
      color: AppStyles.mainColor,
      borderRadius: const BorderRadius.all(Radius.circular(12)),
    ),
    child: BlocBuilder<PartnerCubit, PartnerState>(

```

```

builder: (context, state) {
  if (state is PartnerLoaded) {
    return Column(
      mainAxisAlignment: MainAxisAlignment.min,
      children: [
        const Center(child: Text("Ваш партнер:", style: AppStyles.titleText)),
        const SizedBox(height: 10),
        Row(
          children: [
            const Icon(Icons.account_circle, size: 40),
            const SizedBox(width: 5),
            Text(state.partner.username, style: AppStyles.titleText),
          ],
        ),
        const SizedBox(height: 10),
        _buildSayHelloButton(user, state),
      ],
    );
  } else if (state is PartnerLoading) {
    return const SizedBox(
      width: 20,
      height: 20,
      child: CircularProgressIndicator(strokeWidth: 2),
    );
  } else if (state is PartnerError) {
    return Text("Помилка: ${state.message}", style: AppStyles.titleText);
  } else {
    return Text("Немає даних", style: AppStyles.titleText);
  }
},

```

```

    ),
  );
}

```

```

Widget _buildSayHelloButton(DomainUser user, PartnerLoaded state) {
  String partnerUid = state.partner.firebase_uid;

  if (state.likeStatus == "can_like" || state.likeStatus == "incoming") {
    String text = state.likeStatus == "incoming"
      ? "Привітатися у відповідь! +1  "
      : "Привітатися! +1  ";

    return ElevatedButton(
      onPressed: () async {
        await context.read<PartnerCubit>().sayHello(user.firebase_uid, partnerUid);
      },
      style: AppStyles.buttonStyle,
      child: Text(text, style: AppStyles.titleText, textAlign: TextAlign.center),
    );
  }

  if (state.likeStatus == "mutual") {
    return ElevatedButton(
      onPressed: null,
      style: ElevatedButton.styleFrom(backgroundColor: Colors.grey),
      child: LikeCountdownButton(
        onComplete: () async {
          await context.read<PartnerCubit>().loadPartnerData(user.firebase_uid);
        },
      ),
    );
  }
}

```

```

);
}

return const SizedBox(
  width: 20,
  height: 20,
  child: CircularProgressIndicator(strokeWidth: 2),
);
}

Widget _buildDailyTipCard(DomainUser user) {
  return Container(
    margin: const EdgeInsets.symmetric(horizontal: 15),
    padding: const EdgeInsets.symmetric(vertical: 10, horizontal: 20),
    decoration: BoxDecoration(
      color: AppStyles.mainColor,
      borderRadius: const BorderRadius.all(Radius.circular(12)),
    ),
    child: Column(
      mainAxisAlignment: MainAxisAlignment.min,
      children: [
        const Center(child: Text("Порада дня:", style: AppStyles.titleText)),
        const SizedBox(height: 10),
        Row(
          children: [
            Expanded(child: DailyTipText(userUid: user.firebase_uid)),
          ],
        ),
      ],
    ),
  ),

```

```
);
}
}
```

A.2 Файл `auth_gate.dart`

```
class AuthGate extends StatelessWidget {
  AuthGate({super.key});
  bool _navigated = false;

  @override
  Widget build(BuildContext context) {
    return BlocListener<AuthCubit, AuthState>(
      listener: (context, state) {
        // Если пользователь аутентифицирован, переходим в MainScreen
        if (state.status == AuthStatus.authenticated && state.user != null) {
          Navigator.pushAndRemoveUntil(
            context,
            MaterialPageRoute(
              builder: (_) => MainScreen(user: state.user!),
            ),
            (_) => false,
          );
        }
        // Если статус unsynced, переходим в SyncScreen
        else if (state.status == AuthStatus.unsynced && state.user != null) {
          Navigator.pushAndRemoveUntil(
            context,
            MaterialPageRoute(
              builder: (_) => SyncScreen(state.user!),
            ),
          );
        }
      },
    );
  }
}
```

```

    ),
    (_) => false,
  );
}
// Если email не подтверждён, переходим в EmailVarifiedScreen
else if (state.status == AuthStatus.emailUnverified) {
  Navigator.pushAndRemoveUntil(
    context,
    MaterialPageRoute(
      builder: (_) => const EmailVarifiedScreen(),
    ),
    (_) => false,
  );
}
// Если нет username, переходим в NoUsernameScreen
else if (state.status == AuthStatus.noUsername && state.user != null) {
  Navigator.pushAndRemoveUntil(
    context,
    MaterialPageRoute(
      builder: (_) => NoUsernameScreen(user: state.user!),
    ),
    (_) => false,
  );
}
// Если произошла ошибка или не прошли авторизацию — переходим на
LoginScreen
else {
  Navigator.pushAndRemoveUntil(
    context,
    MaterialPageRoute(

```

```

        builder: (_) => const LoginScreen(),
      ),
      (_) => false,
    );
  }
},
// Пока происходит слушание (и навигация) отображаем индикатор
загрузки
child: const Scaffold(
  body: Center(child: CircularProgressIndicator()),
),
);
}
}

```

A.3 Файл `partner_cubit.dart`

```

class PartnerCubit extends Cubit<PartnerState> {
  PartnerCubit() : super(PartnerInitial());

  final SupabaseService _supa = SupabaseServiceImpl();
  final PartnerService _partnerS = PartnerServiceImpl();

  Future<void> loadPartnerData(String userId) async {
    emit(PartnerLoading());
    try {
      final partner = await _supa.getPartner(userId);
      final hearts = await _supa.getHeartsCount(userId);
      final alreadyLiked = await _partnerS.checkIfAlreadyLikedToday(userId,
partner.firebase_uid);

```

```

if (alreadyLiked){
    emit(PartnerLoaded(partner: partner, hearts: hearts, likeStatus: "mutual", ));
    return;
}
final likeStatus = await _partnerS.checkLikeStatus(currentUserId: userId,
partnerUid: partner.firebase_uid);
    emit(PartnerLoaded(partner: partner, hearts: hearts, likeStatus: likeStatus, ));
} catch (e) {
    emit(PartnerError("Не вдалося завантажити партнера($e)"));
}
}

```

```

Future<void> sayHello(String userId, String partnerUid) async {
    try {
        await _partnerS.createOrRespondToLike( currentUserId: userId,
partnerUid: partnerUid,);
        await loadPartnerData(userId);
        log('👏 Привіт сказано або лайк взаємний');
    } catch (e, stack) {
        log('❌ Помилка в sayHello: $e\n$stack');
    }
}
}

```

ДОДАТОК Б
Слайди презентації

Національний університет “Запорізька політехніка”

Дипломна кваліфікаційна робота бакалавра
на тему: **РОЗРОБКА МОБІЛЬНОГО
ЗАСТОСУНКУ ДЛЯ ПОКРАЩЕННЯ
ВІДНОСИН ІЗ СИСТЕМОЮ
ГЕЙМІФІКАЦІЇ**

Виконав: студент групи КНТ-151

Максим СЕВЕРЕНКО

Керівник: к.т.н, доцент

Ольга ГЛАДКОВА

2025

1

Рисунок Б.1 – Слайд 1

Мета та завдання проекту

Мета роботи – розробити мобільний застосунок для пар, який сприяє зміцненню стосунків через інтерактивні завдання, поради, систему винагород і спільну взаємодію.

Завдання для виконання:

- розробка структури застосунку;
- розробка інтерфейсу та системи завдань;
- забезпечити синхронізацію між партнерами;
- реалізувати гейміфіковані функції;
- розробити програмний застосунок.

2

Рисунок Б.2 – Слайд 2

Вибір засобів розробки

Критерії	Flutter (Dart)	React Native (JS/TS)	Нативна розробка (Kotlin/Swift)	Xamarin (.NET/C#)	Kotlin Multiplatform
Кросплатформність	Так, єдиний код для Android і iOS	Так, з можливістю нативних модулів	Ні, окремо Android та iOS	Так, але потребує обробки UI окремо	Часткова, UI — нативний для кожної платформи
Продуктивність	Висока, близька до нативної	Середня, залежить від мосту JS → натив	Максимальна (прямий доступ до API)	Висока, але складна підтримка	Висока, UI нативний
Швидкість розробки	Висока, Hot Reload, швидкий прототип	Висока, Hot Reload, багатий екосистем	Низька, два окремих проекти	Середня, складність в інтеграції	Середня, UI потребує окремої розробки
Гнучкість UI	Дуже висока, кастомні анімації, власний рендеринг	Добра, обмежена нативними компонентами	Максимальна, нативні елементи UI	Добра, але залежить від платформи	Максимальна, UI нативний
Спільнота і підтримка	Активна, стрімко зростає	Дуже велика, багато пакетів	Дуже велика, підтримка 3 від Google/Apple	Зменшується, від Microsoft фокусує увагу на MAUI	Розвивається, молодий проект
Навчання та складність	Легко вивчити, простий синтаксис Dart	Легко для розробників	JS. Вимагає знань платформи і мови	Середня, залежить від C# і .NET	Вимагає знань Kotlin та нативних UI
Підтримка MVP	Ідеальна, швидко та ефективно	Добра, швидко, але іноді треба більше правок	Повільна, потребує більше ресурсів	Помірна, залежить від досвіду	Помірна, UI окремий

Рисунок Б.3 – Слайд 3

Загальна структура застосунку

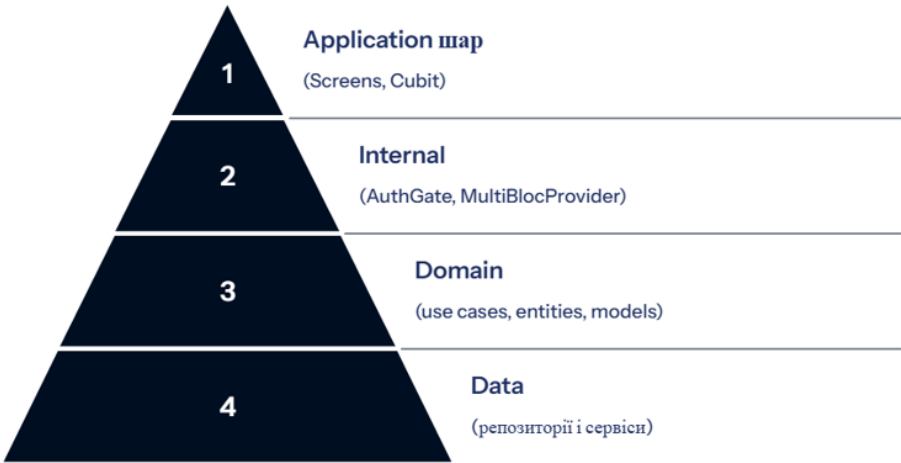


Рисунок Б.4 – Слайд 4

Синхронізація пари

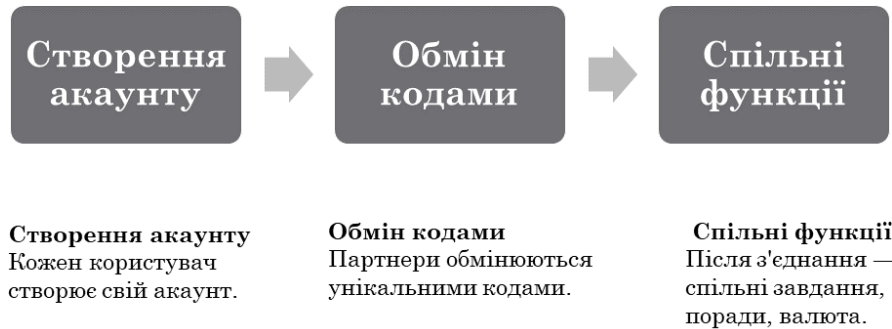


Рисунок Б.5 – Слайд 5

Завдання, поради, "Say Hello"

Основні елементи екрану:

Say Hello:

- Кнопка «Привітатися! +1 ♥» дозволяє надіслати коротке привітання партнеру.
- Кожне натискання — +1 до «валюти».
- Дизайн кнопки емоційний, із серцем, що підкреслює ніжність дії.

Завдання:

- Щоденне завдання для спільного розвитку стосунків.
- Якщо ще не обране — кнопка «Почати!» пропонує нове.
- В майбутньому — можливість додавати/обирати завдання вручну.

Порада дня:

- Коротке натхнення або підказка для роздумів.

Рисунок Б.6 – Слайд 6

Схема бази даних

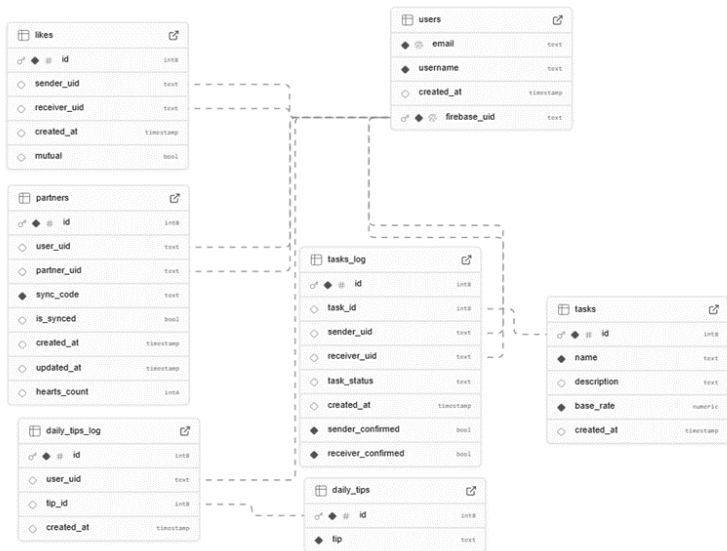


Рисунок Б.7 – Слайд 7

UI/UX і екрани

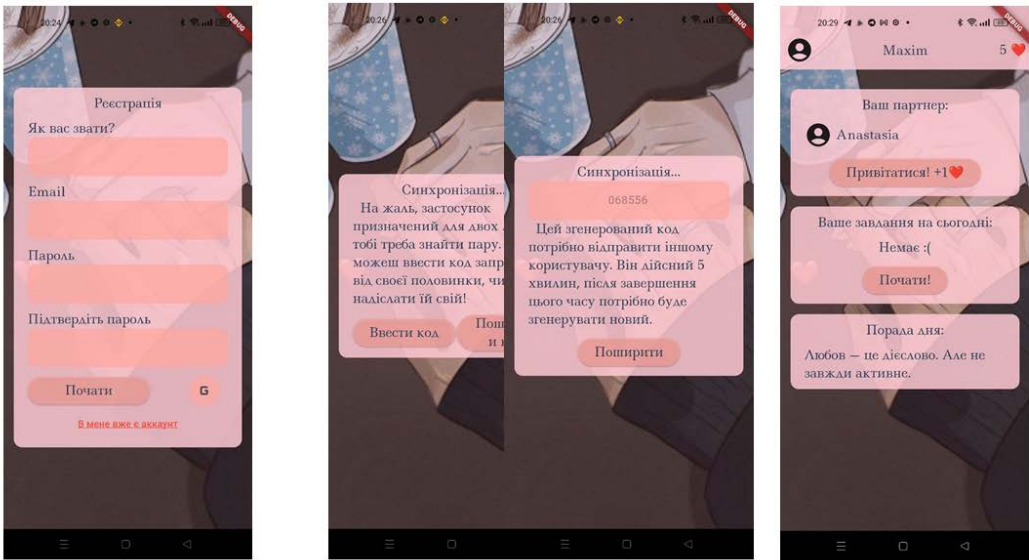


Рисунок Б.8 – Слайд 8

Тестування та результати

Тестування MVP
Проведено на Android, з використанням активного логування і 2ух пристроїв.

Стабільність функцій
Основні функції працюють стабільно.

№	Функція	Очікуваний результат	Результат	Примітка
1	Реєстрація та вхід	Користувач може зареєструватися та увійти в акаунт	✓ Успішно	-
2	Синхронізація з партнером	Користувач бачить ім'я партнера після введення коду	✓ Успішно	Синхронізація працює
3	Кнопка «Привітатись»	Після взаємного натискання додається 1 ❤️	✓ Успішно	Обмеження 1 раз/день
4	Відображення завдання	Користувач бачить завдання на сьогодні	✓ Успішно	-
5	Початок завдання	Завдання активується і зберігається статус	✓ Успішно	-
6	Порада дня	Щоденно відображається нова порада	✓ Успішно	-
7	Вихід із системи	Користувач може вийти з акаунта	✓ Успішно	-

Рисунок Б.9 – Слайд 9

Висновки

Реалізовано MVP застосунку для пар, що підтримує:

- авторизацію користувачів та їх синхронізацію;
- щоденні завдання;
- поради для покращення стосунків;
- функцію «Say Hello» з накопиченням.

Вибір технологій (Flutter + Supabase) дозволив швидко реалізувати кросплатформене рішення з сучасною архітектурою.

Можливі напрямки розвитку — можна додати чат, нагадування, спільні цілі, розширену аналітику, тощо.

Рисунок Б.10 – Слайд 10