

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 РОЗРОБЛЕННЯ МОБІЛЬНОГО ЗАСТОСУНКУ
«ПЛАНУВАЛЬНИК СІМЕЙНОГО БЮДЖЕТУ»
DEVELOPMENT OF THE MOBILE APPLICATION
«FAMILY BUDGET PLANNER»

—

Виконав(ла): студент(ка) 4 курсу, групи КНТ-131

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

АНТРОПОВ Ю.О.

(ПРИЗВИЩЕ та ініціали)

Керівник ГОФМАН Є.О.

(ПРИЗВИЩЕ та ініціали)

Рецензент ЗЕЛЕНЬОВА І.Я.

(ПРИЗВИЩЕ 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

АНТРОПОВА Юрія Олексійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розроблення мобільного застосунку «Планувальник сімейного бюджету»

Development of the Mobile Application “Family Budget Planner”

керівник проєкту (роботи) к.т.н., ГОФМАН Євгеній Олександрович ,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

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

2. Строк подання студентом проєкту (роботи) 01 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. Розробка архітектури програми. 3. Розробка програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

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

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Прийняв виконане завдання
1-4 Основна частина	ГОФМАН Є. О., доцент		
Нормоконтроль	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання “ 14 ” квітня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів курсового проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2–3 тижні	Розділ 1
3	Розробка архітектури програми.	4 тиждень	Розділ 2
4	Розробка програми.	5 тиждень	Розділ 3
5	Тестування та експериментальне дослідження програми.	6 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування.	7 тиждень	Додатки
7	Захист роботи.	8 тиждень	

Студент(ка)

(підпис)

Юрій АНТРОПОВ
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис)

Євгеній ГОФМАН
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до випускної кваліфікаційної роботи бакалавра:
147с., 11 табл., 62 рис., 3 дод., 18 джерел.

БЮДЖЕТ, МОБІЛЬНИЙ ЗАСТОСУНОК, ОРГАНІЗАЦІЯ ФІНАНСІВ,
СІМЕЙНІ ФІНАНСИ, СИНХРОНІЗАЦІЯ, ШИФРУВАННЯ, ШТУЧНИЙ
ІНТЕЛЕКТ.

Об'єкт дослідження – процес автоматизації обліку та управління сімейними фінансами.

Предмет дослідження – методи та інструменти для розробки мобільного застосунку «Планувальник сімейного бюджету».

Мета роботи – підвищення ступіню автоматизації і ефективності процесу управління сімейними фінансами та розширення функціоналу мобільного застосунку за рахунок підключення засобів штучного інтелекту.

Матеріали, методи та технічні засоби: використано методи об'єктно–орієнтованого програмування, технології Flutter і Dart для створення кросплатформеного інтерфейсу, Firebase для аутентифікації, синхронізації та зберігання даних на сервері, SQLite + Drift для локального зберігання, бібліотеку encrypt для шифрування даних (AES + RSA). Для аналізу фінансових даних застосовано алгоритми обробки даних та штучний інтелект. Розроблення проводилося в середовищі Android Studio Meerkat на персональному комп'ютері з операційною системою Windows 11.

Результати. Розроблено кросплатформений мобільний застосунок «Планувальник сімейного бюджету», який забезпечує облік доходів і витрат, миттєву синхронізацію між членами сім'ї, шифрування даних та аналіз фінансової активності за допомогою штучного інтелекту. Реалізовано систему ролей із гнучким налаштуванням прав доступу, інтуїтивний інтерфейс із

підтримкою drag-and-drop, офлайн-режим із локальним зберіганням даних, а також чат для спілкування членів сім'ї.

Висновки. Досягнуто мети роботи: створено застосунок, який автоматизує управління сімейними фінансами, забезпечує безпечне зберігання та синхронізацію даних, а також надає рекомендації для оптимізації витрат. Застосунок підтримує українську та англійську мови, темну та світлу теми, біометричну аутентифікацію.

Галузь використання – управління сімейними фінансами для домогосподарств, а також як інструмент для фінансового планування в малих групах.

Економічна ефективність. Використання застосунку скорочує час на облік фінансів у 2 рази порівняно з ручними методами, зменшує ризик помилок у фінансових записах, сприяє економії завдяки рекомендаціям штучного інтелекту.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 147 pages, 11 tables, 62 figures, 3 appendixes, 18 sources.

BUDGET, MOBILE APPLICATION, FINANCIAL ORGANIZATION, FAMILY FINANCES, SYNCHRONIZATION, ENCRYPTION, ARTIFICIAL INTELLIGENCE.

Research object – the process of managing family finances using mobile applications.

Research subject – a mobile application for organizing and synchronizing family budgets using modern technologies.

The purpose of the work – to develop a mobile application for automating family finance management, ensuring instant data synchronization among users, and providing expenditure optimization recommendations using artificial intelligence.

Materials, methods, and technical means: object-oriented programming methods, Flutter and Dart technologies for creating a cross-platform interface, Firebase for authentication, synchronization, and data storage on the server, SQLite for local storage, and the encrypt library for data encryption (AES + RSA) were used. Data processing algorithms and artificial intelligence were applied for financial data analysis. Development was carried out in the Android Studio Meerkat (2024.3.1) environment on a personal computer running Windows 11.

Results. A cross-platform mobile application “Family Budget Planner” was developed, enabling income and expense tracking, instant synchronization among family members, data encryption, and financial activity analysis using artificial intelligence. A role-based system with flexible access rights configuration, an intuitive drag-and-drop interface, offline mode with local data storage, and a family chat were implemented.

Conclusions. The work's objective was achieved: an application was created that automates family finance management, ensures secure data storage and synchronization, and provides recommendations for expenditure optimization. The application supports Ukrainian and English languages, dark and light themes, and biometric authentication.

Field of application – family finance management for households and as a tool for financial planning in small groups.

Economic efficiency. The application reduces financial tracking time by 2 times compared to manual methods, minimizes errors in financial records, and promotes savings through artificial intelligence recommendations.

ЗМІСТ

	С.
Перелік скорочень та умовних познач	11
Вступ	12
1 Аналіз предметної області	15
1.1 Короткий опис досліджуваної предметної області.....	15
1.2 Огляд існуючих рішень проблеми	17
1.2.1 Застосунок Mobills.....	17
1.2.2 Застосунок Monefy	18
1.2.3 Застосунок Wallet.....	19
1.2.4 Порівняльна таблиця існуючих та розроблюваного програмних продуктів	21
1.3 Опис основних вимог до програми	22
1.3.1 Визначення функціональних вимог	22
1.3.2 Нефункціональні вимоги	24
1.4 Постановка завдання роботи	25
1.5 Висновки за розділом 1	26
2 Розробка архітектури програми	27
2.1 Структура системи	27
2.1.1 Основні компоненти системи.....	27
2.1.2 Взаємодія компонентів системи	28
2.2 Функціонування системи	28
2.2.1 Основні процеси системи	28
2.2.3 Логіка роботи системи	29
2.3 Вибір мови програмування та інструментарію для розробки програми	30
2.3.1 Мова програмування.....	30
2.3.2 Середовище розробки.....	32
2.3.3 База даних	34
2.4 Вимоги до технічних і програмних засобів	36
2.4.1 Необхідні технічні засобів для розробки та експлуатації.....	36

2.4.2	Вимоги до апаратного та програмного забезпечення	37
2.5	Висновки за розділом 2	37
3	Розробка програми	38
3.1	Архітектура системи.....	38
3.1.1	Структурна схема системи.....	39
3.2	Функціонування системи	40
3.2.1	Функціональна схема.....	40
3.2.2	Опис процесу обміну даними у системі.....	41
3.2.3	Сценарії використання	42
3.3	Опис модулів системи	44
3.3.1	Модуль реєстрації та авторизації.....	44
3.3.2	Модуль управління сімейними фінансами.....	46
3.3.3	Модуль синхронізації даних.....	49
3.3.4	Модуль статистики та аналізу.....	52
3.3.5	Модуль ІІІ–помічника.....	55
3.3.6	Модуль чату.....	58
3.3.7	Модуль налаштувань	60
3.4	Проектування бази даних	62
3.4.2	Взаємозв'язки між таблицями	68
3.4.3	Індексація та оптимізація	69
3.5	Висновки за розділом 3	70
4	Експлуатація, тестування та експериментальне дослідження програми	72
4.1	Опис застосування програми	72
4.1.1	Огляд планової цільової аудиторії.....	72
4.1.2	Візуалізація планованої цільової аудиторії.....	75
4.2	Інструкція по роботі з програмою	76
4.2.1	Встановлення програми	76
4.2.2	Запуск програми.....	77
4.2.3	Методика роботи з програмою	80
4.3	Приклад роботи користувача з програмою	81

	10
4.4 Тестування роботи програми	87
4.4.1 Функціональне тестування.....	87
4.4.2 Аналіз функціонального тестування	88
4.4.3 Тестування продуктивності	89
4.4.4 Аналіз продуктивності	90
4.4.5 Тестування безпеки	91
4.4.6 Аналіз безпеки	92
4.4.7 Аналіз користувацького досвіду	93
4.5 Рекомендації щодо використання	94
4.6 Плани з розвитку.....	94
Висновки.....	96
Перелік джерел посилання	98
Додаток А Технічне завдання.....	100
Додаток Б Текст програми.....	106
Додаток В Слайди презентації	143

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

AES	– Advanced Encryption Standard;
API	– Application Programming Interface;
FCM	– Firebase Cloud Messaging;
IV	– Initialization Vector;
NFC	– Near Field Communication;
ORM	– Object–Relational Mapping;
RSA	– Rivest–Shamir–Adleman;
UI	– User Interface;
UX	– User Experience;
ЗК(FK)	– зовнішній ключ;
ПК(PK)	– первинний ключ.

ВСТУП

Сучасний стан розвитку інформаційних технологій характеризується швидким зростанням популярності мобільних застосунків, які полегшують повсякденне життя. Однією з ключових сфер, де такі технології знаходять застосування, є управління особистими та сімейними фінансами. Проблема ефективного обліку доходів і витрат, синхронізації фінансових даних між членами сім'ї та забезпечення безпеки цих даних залишається актуальною. Незважаючи на існування застосунків для фінансового планування, більшість із них не враховують специфіку сімейного використання, не забезпечують миттєвої синхронізації або мають обмежені можливості аналізу та оптимізації витрат.

Проблема ефективного управління сімейними фінансами є актуальною в усьому світі. За даними дослідження Schwab Modern Wealth Survey 2024, значна частина домогосподарств не веде систематичного обліку та планування доходів і витрат, що призводить до фінансових труднощів [1]. У той же час сучасні технології, такі як мобільні застосунки, дозволяють автоматизувати цей процес, забезпечуючи зручність і доступність. Проте більшість існуючих рішень або не враховують потреби сімейного використання, або мають обмежений функціонал, наприклад, відсутність синхронізації чи аналізу витрат.

Проблема дослідження полягає в необхідності створення інструменту, який забезпечує зручний облік сімейних фінансів, миттєву синхронізацію даних між користувачами, захист інформації та інтелектуальний аналіз фінансових операцій. Огляд літературних та аналітичних джерел показує, що хоча подібні задачі частково вирішуються в таких застосунках, як Mobills, Monefy чи Wallet, вони часто не підтримують гнучкі ролі користувачів, офлайн-режим або інтеграцію штучного інтелекту для рекомендацій [2]. Провідні дослідження в галузі мобільної розробки (зокрема, праці з конференцій Google I/O та публікації щодо Firebase) вказують на

перспективність використання кроссплатформених фреймворків і хмарних технологій, однак бракує комплексних рішень для сімейного використання.

Об'єктом дослідження є процес управління сімейними фінансами за допомогою мобільних застосунків.

Предметом дослідження є мобільний застосунок для організації, синхронізації та аналізу сімейного бюджету з використанням сучасних технологій.

Мета роботи – розроблення мобільного застосунку для автоматизації управління сімейними фінансами, забезпечення миттєвої синхронізації даних між користувачами та надання рекомендацій щодо оптимізації витрат за допомогою штучного інтелекту.

Для досягнення поставленої мети в роботі вирішувалися такі завдання:

- проаналізувати сучасні методи та інструменти управління сімейними фінансами, а також технології для створення мобільних застосунків;
- розробити архітектуру кроссплатформеного мобільного застосунку з підтримкою офлайн-режиму, синхронізації та шифрування даних;
- програмно реалізувати функціонал обліку доходів і витрат, системи ролей, чату, статистики та інтелектуального аналізу;
- експериментально перевірити працездатність застосунку, його швидкодію, безпеку та зручність використання.

Методика дослідження включає аналіз літератури та аналогів, об'єктно-орієнтоване програмування, використання фреймворку Flutter і платформи Firebase для розроблення, тестування в реальних умовах із залученням тестових користувачів. Для шифрування застосовується гібридна схема (AES + RSA), а для аналізу даних – алгоритми обробки фінансової інформації.

Застосунок може бути використаний домогосподарствами для управління фінансами, а також як інструмент для навчання фінансовій грамотності. Результати роботи мають практичну цінність, оскільки

зменшують час на облік фінансів і сприяють економії завдяки рекомендаціям штучного інтелекту.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Короткий опис досліджуваної предметної області

Управління сімейними фінансами – це процес, який охоплює планування, облік і контроль доходів і витрат у межах домогосподарства. Для багатьох сімей він зводиться до рутинних записів у зошитах чи таблицях, що часто спричиняє помилки, втрату даних або труднощі в обміні інформацією. Це підкреслює потребу в сучасних інструментах, здатних автоматизувати фінансовий облік.

Основні поняття, пов'язані з управлінням фінансами, включають:

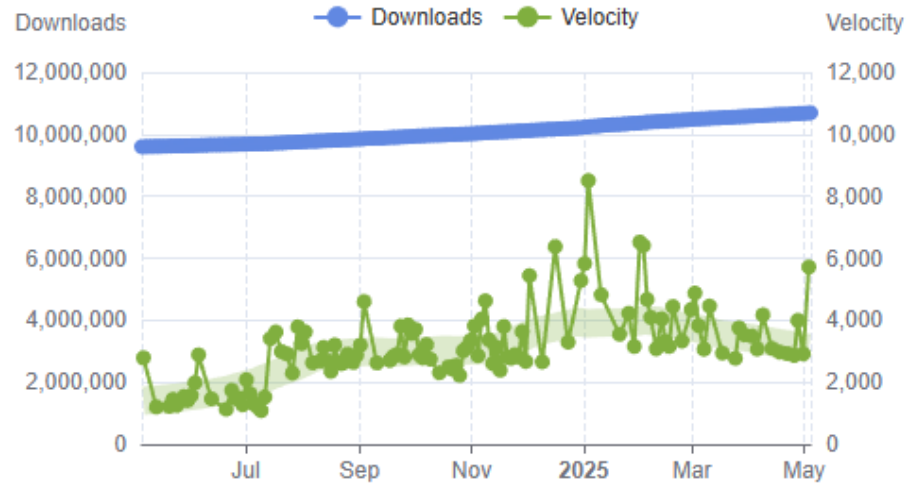
- бюджет – план доходів і витрат на певний період, що допомагає контролювати фінансові потоки;
- доходи – кошти, отримані від заробітної плати, інвестицій, подарунків тощо;
- витрати – кошти, витрачені на товари, послуги чи інші потреби;
- фінансовий аналіз – оцінка фінансових даних для виявлення тенденцій, оптимізації витрат і прийняття обґрунтованих рішень.

Сьогодні управління сімейними фінансами стикається з низкою викликів. По–перше, це брак часу на ведення детального обліку, адже сучасний ритм життя залишає мало часу на такі задачі. По–друге, складність синхронізації даних між членами сім'ї, особливо коли кожен використовує власний пристрій. По–третє, питання безпеки: фінансові дані є чутливими, і їх втрата чи витік можуть мати серйозні наслідки. Нарешті, більшість сімей не мають доступу до інструментів, які б не лише фіксували витрати, а й пропонували розумні поради щодо їх оптимізації.

Серед тенденцій у цій галузі можна виділити зростання популярності мобільних застосунків для фінансового планування, інтеграцію штучного інтелекту для аналізу витрат і рекомендацій, а також акцент на кросплатформених рішеннях, які працюють як на Android, так і на iOS. Графік

зростання кількості користувачів мобільного застосунка Wallet та Monefy за останній рік зображено на рисунку 1.1 та 1.2.

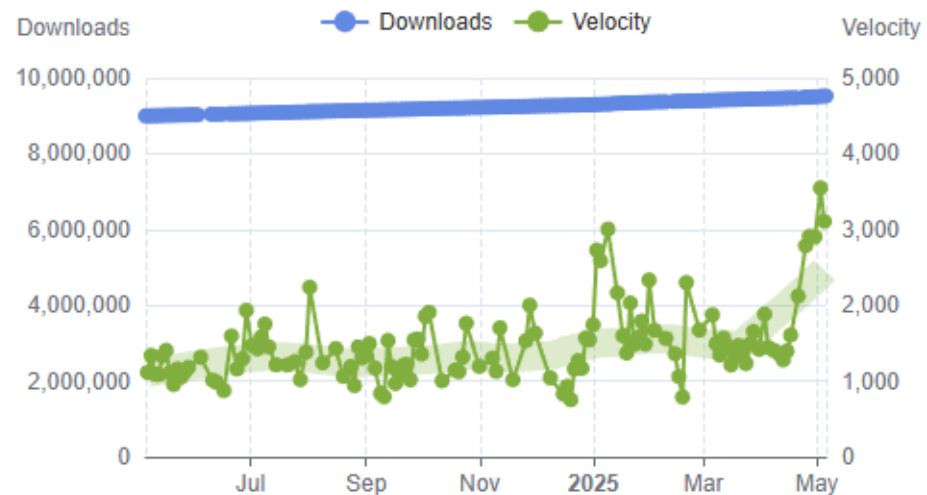
Downloads over time



Wallet: Budget Expense Tracker has been downloaded *11 million* times. Over the past 30 days, it averaged *3.4 thousand* downloads per day.

Рисунок 1.1 – Графік зростання кількості користувачів мобільного застосунка Wallet за останній рік [3]

Downloads over time



Monefy - Budget & Expenses app has been downloaded *9.5 million* times. Over the past 30 days, it averaged *2.2 thousand* downloads per day.

Рисунок 1.2 – Графік зростання кількості користувачів мобільного застосунка Monefy за останній рік [3]

1.2 Огляд існуючих рішень проблеми

Розглянемо три популярні мобільні застосунки для управління фінансами: Mobills, Monefy та Wallet. Кожен із них має свої особливості, але жоден повною мірою не відповідає потребам сімейного використання.

1.2.1 Застосунок Mobills

Mobills – це бразильський застосунок для управління особистими фінансами, доступний на Android та iOS. Він дозволяє вести облік доходів і витрат, створювати категорії, планувати бюджет і переглядати статистику у вигляді графіків. Застосунок підтримує синхронізацію через хмарне сховище, а також інтеграцію з банківськими рахунками в деяких країнах. Є функція встановлення фінансових цілей, наприклад, накопичення на відпустку [4].

Переваги застосунку:

- зручний інтерфейс із підтримкою графіків і діаграм;
- можливість синхронізації між пристроями (у преміум-версії);
- інтеграція з банківськими рахунками (обмежено регіонами).

Недоліки:

- обмежена підтримка сімейного використання: немає ролей чи розмежування прав доступу;
- більшість функцій (зокрема синхронізація) доступні лише в платній версії;
- відсутність офлайн-режиму для повноцінної роботи;
- немає інтеграції штучного інтелекту для аналізу чи рекомендацій.

Вигляд домашній сторінки застосунку Mobills продемонстровано на рисунку 1.3.

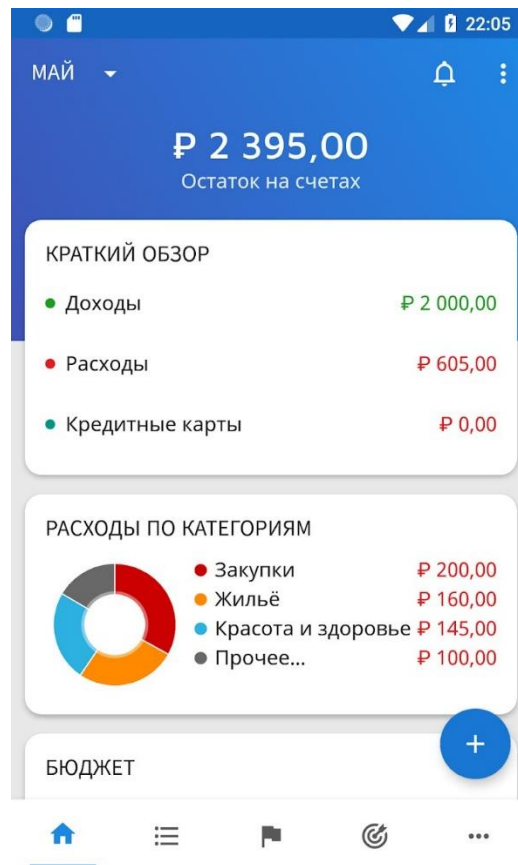


Рисунок 1.3 – Головний екран Mobills [4]

1.2.2 Застосунок Monefy

Monefy – це простий і популярний застосунок для ведення бюджету, доступний на Android та iOS. Його головна особливість – мінімалістичний інтерфейс, який дозволяє швидко додавати витрати чи доходи одним натисканням. Застосунок підтримує категорії витрат, статистику у вигляді кругових діаграм і синхронізацію через Google Drive або Dropbox у преміум-версії [5].

Основні переваги застосунку:

- простий і швидкий у використанні, ідеальний для новачків;
- підтримка офлайн-режиму для базових функцій;
- низькі вимоги до ресурсів пристрою.

Недоліки:

- відсутність підтримки сімейного використання чи ролей користувачів;

- синхронізація доступна лише в платній версії;
- обмежені можливості аналізу: немає інтелектуальних рекомендацій;
- відсутність шифрування даних, що знижує безпеку.

Вигляд домашній сторінки застосунку Monefy продемонстровано на рисунку 1.4.

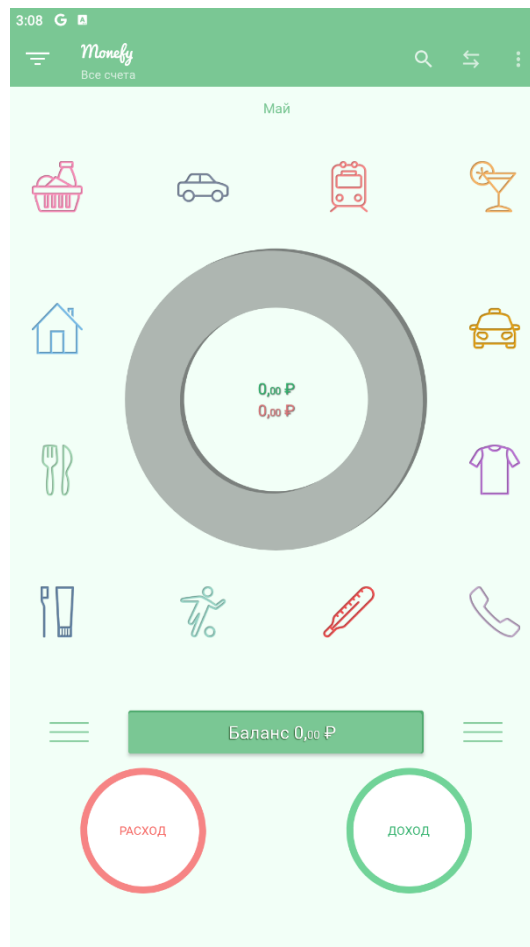


Рисунок 1.4 – Головний екран Monefy [5]

1.2.3 Застосунок Wallet

Wallet, розроблений BudgetBakers, є одним із найфункціональніших застосунків для управління фінансами. Він підтримує автоматичний імпорт транзакцій із банківських рахунків, створення детальних звітів, планування бюджету та синхронізацію між пристроями через хмарне сховище. Застосунок має гнучкі налаштування категорій і тегів для витрат [6].

Основні переваги:

- автоматичний імпорт транзакцій із банків (залежить від регіону);
- детальна статистика та звіти;
- синхронізація між пристроями в базовій версії.

Недоліки:

- складний інтерфейс для новачків через велику кількість функцій;
- відсутність підтримки сімейного використання з розподілом ролей;
- обмежена безпека: немає інформації про шифрування даних;
- багато функцій доступні лише в преміум-версії.

Вигляд домашній сторінки застосунку Wallet продемонстровано на рисунку 1.5.

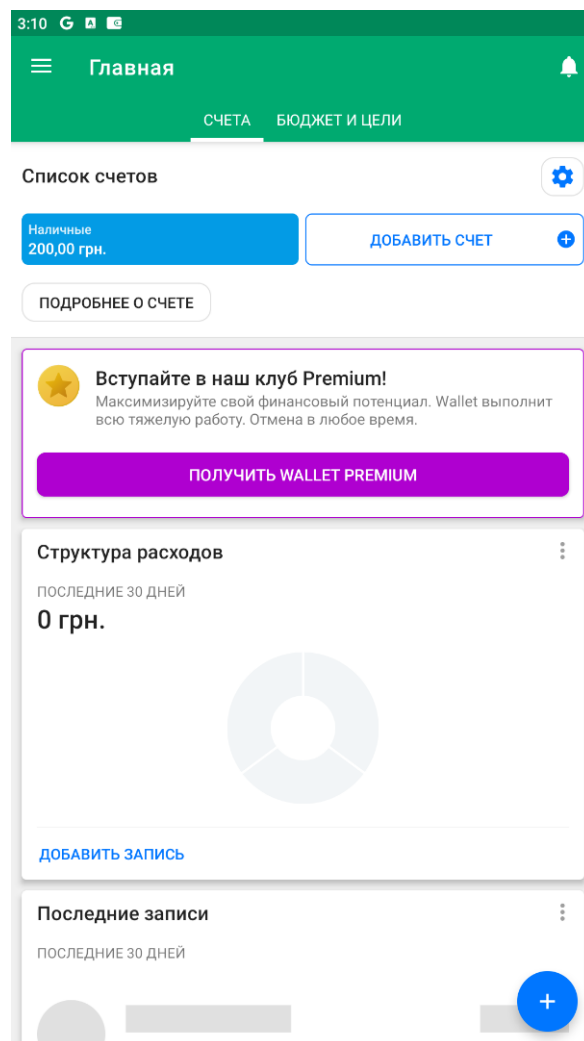


Рисунок 1.5 – Головний екран Wallet [6]

1.2.4 Порівняльна таблиця існуючих та розроблюваного програмних продуктів

Для оцінки сильних і слабких сторін розглянутих застосунків порівняємо їх із розроблюваним застосунком «Планувальник сімейного бюджету» за ключовими критеріями: функціональність, зручність використання, безпека та підтримка сімейного використання.

В таблиці 1.1 проведено порівняння вже існуючих застосунків та програми, яка буде розроблена в ході виконання роботи.

Таблиця 1.1 — Порівняння функціональності застосунків для управління фінансами

Критерій	Mobills	Monefy	Wallet	Планувальник сімейного бюджету
1	2	3	4	5
Функціональність	Стандартний облік доходів і витрат, графіки, цілі	Базовий облік, діаграми	Імпорт транзакцій, фінансові звіти	Облік, аналіз із використанням ШІ, чат, ролі
Сімейне використання	Не підтримується	Не підтримується	Не підтримується	Підтримується (ролі, синхронізація)
Синхронізація	Платна, хмарна	Платна, Google Drive / Dropbox	Безкоштовна, хмарна	Безкоштовна, через Firebase
Офлайн-режим	Обмежений	Повний	Обмежений	Повний (на базі SQLite + Drift)

Продовження таблиці 1.1

1	2	3	4	5
Безпека	Базова	Відсутнє шифрування	Базовий рівень	Шифрування AES + RSA
Зручність (UI/UX)	Зручний, але складний інтерфейс	Дуже простий	Складний для новачків	Інтуїтивний інтерфейс, drag-and-drop
Рекомендації ШІ	Відсутні	Відсутні	Відсутні	Підтримуються

1.3 Опис основних вимог до програми

1.3.1 Визначення функціональних вимог

Функціональні вимоги визначають, що саме повинна робити система. Для мого мобільного застосунку «Планувальник сімейного бюджету» визначено такі основні функціональні вимоги:

- а) реєстрація та автентифікація користувачів:
 - 1) реєстрація через email/пароль і Google;
 - 2) вхід за допомогою біометричних даних (Face ID / Touch ID) або PIN-коду;
 - 3) скидання пароля через email;
- б) управління сім'єю:
 - 1) створення сім'ї адміністратором із генерацією унікального коду запрошення;
 - 2) запрошення членів сім'ї через email або месенджери;
 - 3) призначення ролей (адміністратор, дитина, користувачькі ролі);
 - 4) налаштування індивідуальних прав доступу;
 - 5) перегляд і видалення членів сім'ї;
- в) облік фінансів:

- 1) створення та управління рахунками (банківські картки, готівка тощо);
 - 2) введення доходів і витрат через інтуїтивний drag-and-drop інтерфейс;
 - 3) категоризація транзакцій з можливістю додавання власних категорій;
 - 4) перекази між рахунками;
- г) синхронізація даних:
- 1) миттєва синхронізація через Firebase Firestore між усіма пристроями сім'ї;
 - 2) офлайн-режим із локальним зберіганням даних;
 - 3) обробка конфліктів синхронізації;
- д) статистика та аналітика:
- 1) детальна статистика доходів і витрат із фільтрацією;
 - 2) візуалізація даних: графіки, діаграми, таблиці;
 - 3) інтеграція ІШ-помічника для аналізу витрат і рекомендацій;
- е) чат:
- 1) внутрішньосімейний чат із шифруванням повідомлень;
 - 2) збереження повідомлень у зашифрованому вигляді;
- ж) безпека:
- 1) шифрування даних на пристрої та під час передачі;
 - 2) біометрична автентифікація;
 - 3) система ролей та прав доступу;
- з) додаткові функції:
- 1) геометки для транзакцій;
 - 2) експорт/імпорт даних;
 - 3) вибір мови та теми;
 - 4) налаштування сповіщень.

1.3.2 Нефункціональні вимоги

Нефункціональні вимоги визначають характеристики та обмеження системи:

а) вимоги до безпеки:

- 1) шифрування повідомлень чату за алгоритмом AES–256;
- 2) локальне шифрування фінансових даних;
- 3) безпечне зберігання ключів шифрування з апаратним захистом;
- 4) захист від несанкціонованого доступу через біометричну автентифікацію;

б) вимоги до продуктивності:

- 1) час запуску застосунку не більше 2 секунд;
- 2) час відгуку інтерфейсу не більше 0,5 секунди;
- 3) підтримка офлайн–режиму з повною функціональністю;
- 4) оптимізація використання пам'яті (не більше 100 Мб у робочому режимі);
- 5) розмір застосунку не більше 20 Мб;

в) вимоги до користувацького інтерфейсу:

- 1) інтуїтивно зрозумілий інтерфейс з мінімальною кількістю кроків для виконання операцій;
- 2) підтримка жестів drag–and–drop для додавання транзакцій;
- 3) адаптивний дизайн для різних розмірів екранів;
- 4) підтримка світлої та темної теми;

г) вимоги до надійності:

- 1) автоматичне збереження даних кожні 30 секунд;
- 2) відновлення після збоїв без втрати даних;
- 3) резервне копіювання даних;

д) вимоги до масштабованості:

- 1) підтримка до 10 користувачів у одній сім'ї;

- 2) зберігання історії транзакцій до 10 років;
 - 3) можливість розширення функціональності без зміни архітектури;
- е) вимоги до сумісності:
- 1) підтримка Android 6.0 (API 23) і вище;
 - 2) підтримка iOS 12 і вище;
 - 3) коректна робота на різних пристроях (смартфони, планшети);
- ж) вимоги до локалізації:
- 1) підтримка української та англійської мов;
 - 2) можливість додавання інших мов в майбутньому;
- з) вимоги до підтримки:
- 1) автоматичне оновлення через магазини застосунків;
 - 2) зворотний зв'язок через вбудовану форму.

1.4 Постановка завдання роботи

Розроблення мобільного застосунку «Планувальник сімейного бюджету» спрямоване на вирішення проблеми неефективного управління сімейними фінансами.

Основні завдання роботи:

- аналіз вимог. Вивчити потреби користувачів щодо управління сімейними фінансами, визначити функціональні та нефункціональні вимоги до застосунку;
- розроблення архітектури. Створити кросплатформений застосунок на основі Flutter і Dart із підтримкою локального зберігання (SQLite + Drift), хмарної синхронізації (Firebase) і шифрування даних (AES + RSA);
- реалізація функціоналу. Програмно реалізувати облік доходів і витрат, систему ролей із гнучким налаштуванням прав, інтуїтивний інтерфейс із drag-and-drop, модуль статистики, чат і штучний інтелект для рекомендацій;

- забезпечення безпеки. Впровадити гібридне шифрування, біометричну аутентифікацію та захист даних у Firebase;
- тестування та оптимізація. Провести тестування функціональності, безпеки, швидкодії та зручності, оптимізувати розмір застосунку до 10–20 Мб.

Цілі розробки:

- створити зручний і безпечний інструмент для сімейного бюджетування;
- забезпечити миттєву синхронізацію даних між членами сім'ї;
- надати користувачам інтелектуальні рекомендації для оптимізації витрат;
- забезпечити підтримку офлайн-режиму та кросплатформність.

1.5 Висновки за розділом 1

Аналіз предметної області показав, що управління сімейними фінансами є актуальною проблемою через обмежену функціональність наявних рішень. Існуючі застосунки, такі як Mobills, Monefy і Wallet, частково вирішують задачу індивідуального бюджетування, але не відповідають потребам сімейного використання. Основними викликами є забезпечення миттєвої синхронізації, захисту даних і простоти інтерфейсу. Розроблюваний застосунок «Планувальник сімейного бюджету» спрямований на подолання цих обмежень шляхом інтеграції ролей користувачів, шифрування, офлайн-режиму та штучного інтелекту.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Структура системи

2.1.1 Основні компоненти системи

Архітектура застосунку, базується на шаблоні проєктування MVVM (Model–View–ViewModel) з використанням Provider для управління станом [7]. Система складається з таких основних компонентів.

Шар представлення (View):

- екрани (Screens) — відповідають за відображення інтерфейсу користувача;
- віджети (Widgets) — багаторазові компоненти інтерфейсу.

Шар моделі представлення (ViewModel):

- провайдери (Providers) — керують станом застосунку та бізнес–логікою;
- сервіси (Services) — забезпечують функціональність для різних частин застосунку.

Шар моделі (Model):

- моделі даних (Models) — представляють структуру даних;
- база даних (Database) — забезпечує зберігання та доступ до даних.

Шар сервісів (Services):

- сервіс автентифікації (AuthService);
- сервіс бази даних (DbService);
- сервіс шифрування (EncryptionService);
- сервіс ІІІ (AIService);
- сервіс валют (CurrencyService);
- сервіс резервного копіювання (BackupService);
- сервіс NFC (NFCService).

Шар зовнішніх інтеграцій:

- Firebase (автентифікація, база даних, сховище);
- API для ІІІ–рекомендацій.

2.1.2 Взаємодія компонентів системи

Взаємодія між компонентами системи відбувається за такою схемою.

Користувацький інтерфейс – ViewModel:

- екрани та віджети відображають дані з провайдерів;
- користувацькі дії передаються до провайдерів для обробки.

ViewModel – Сервіси:

- провайдери використовують сервіси для виконання операцій;
- сервіси повертають результати до провайдерів.

Сервіси – Модель даних:

- сервіси працюють з моделями даних;
- моделі даних зберігаються в базі даних.

Сервіси – Зовнішні інтеграції:

- сервіси взаємодіють з Firebase та іншими зовнішніми API;
- результати інтегруються назад у систему.

2.2 Функціонування системи

2.2.1 Основні процеси системи

Функціонування системи можна описати через основні процеси, які відбуваються під час використання програми.

Процес реєстрації та входу:

- користувач реєструється через email/пароль або Google;
- система створює обліковий запис у Firebase Authentication;
- при наступних входах користувач може використовувати біометричну автентифікацію.

Процес створення сім'ї:

- користувач створює сім'ю, стаючи її адміністратором;
- система генерує унікальний код запрошення;

- користувач запрошує інших членів сім'ї через код, посилання, або користувач самостійно знаходить сім'ю через функцію пошуку сімей через локальну мережу.

Процес додавання транзакцій:

- користувач перетягує рахунок на категорію витрати, чи категорію доходу на рахунок, для створення транзакції;
- система зберігає транзакцію локально та в Firebase;
- система оновлює баланс рахунку.

Процес синхронізації даних:

- система автоматично синхронізує дані між пристроями через Firebase;
- у разі відсутності інтернету дані зберігаються локально;
- при відновленні з'єднання відбувається синхронізація з вирішенням конфліктів.

Процес аналізу та рекомендацій:

- система аналізує фінансові дані користувача;
- ШІ–помічник генерує рекомендації щодо оптимізації витрат;
- користувач отримує персоналізовані поради.

2.2.3 Логіка роботи системи

Логіка роботи системи базується на таких принципах.

Принцип локального–першого зберігання:

- усі дані спочатку зберігаються локально для забезпечення швидкодії;

- потім відбувається асинхронна синхронізація з хмарию;

Це забезпечує роботу в офлайн–режимі та швидкий відгук інтерфейсу.

Принцип шифрування даних:

- чутливі дані шифруються перед збереженням;
- для чату використовується end–to–end шифрування;
- ключі шифрування зберігаються в захищеному сховищі.

Принцип ролей та прав:

- кожен користувач має роль з певними правами;
- система перевіряє права перед виконанням операцій;
- адміністратор може гнучко налаштовувати права.

Принцип реактивного оновлення:

- інтерфейс автоматично оновлюється при зміні даних;
- використовується патерн Provider для управління станом;
- Firebase забезпечує реактивні оновлення в реальному часі.

2.3 Вибір мови програмування та інструментарію для розробки програми

2.3.1 Мова програмування

Для розробки мобільного застосунку «Планувальник сімейного бюджету» обрано мову програмування Dart, яка використовується разом із фреймворком Flutter [8]. Вибір зумовлений кількома ключовими факторами, що відповідають вимогам проєкту:

Кросплатформність: Dart у поєднанні з Flutter дозволяє створювати нативні застосунки для платформ Android та iOS із єдиної кодової бази. Це значно скорочує час і ресурси на розробку та подальшу підтримку застосунку, що є критичним для забезпечення сумісності з різними пристроями, як зазначено в нефункціональних вимогах (підтримка Android 6.0+ та iOS 12+).

Продуктивність: Dart компілюється у нативний машинний код за допомогою AOT (Ahead-of-Time) компіляції, що забезпечує високу швидкість виконання застосунку. Це особливо важливо для фінансових застосунків, де користувачі очікують миттєвого відгуку інтерфейсу (вимога: час відгуку не більше 0,5 секунди).

Зрозумілий синтаксис: Dart має простий і чистий синтаксис, схожий на JavaScript і C#, що полегшує розробку, зрозуміння коду та його підтримку командою розробників. Це сприяє швидкому написанню коду та зменшенню кількості помилок.

Підтримка асинхронного програмування: вбудована підтримка асинхронних операцій (`async/await`) дозволяє ефективно працювати з мережевими запитами, локальними базами даних і синхронізацією через Firebase, що є ключовим для реалізації офлайн-режиму та миттєвої синхронізації.

Активна підтримка та екосистема: Dart підтримується компанією Google, що гарантує регулярні оновлення, стабільність і доступ до широкої екосистеми бібліотек та інструментів. Flutter SDK включає великий набір готових віджетів і плагінів, які спрощують реалізацію складних функцій, таких як `drag-and-drop` інтерфейс чи інтеграція з ІІІ.

2.3.1.1 Опис переваг та обмежень вибраної мови

Переваги.

Швидкість розробки забезпечується кросплатформністю та великою кількістю готових компонентів у Flutter, що робить розробку на Dart швидшою порівняно з нативними мовами, такими як Java/Kotlin для Android чи Swift для iOS.

Висока продуктивність досягається завдяки АОТ-компіляції, яка забезпечує швидкодію, близьку до нативних застосунків, що є критичним для обробки фінансових даних у реальному часі.

Функція `Hot Reload` у Flutter прискорює процес тестування та налагодження коду.

Розвинена екосистема включає бібліотеки для роботи з Firebase, шифрування (`encrypt`), локальних баз даних (`Drift`) і ІІІ (`http`), що полегшує інтеграцію необхідних функцій.

UI-гнучкість завдяки Flutter дозволяє створювати кастомізовані інтерфейси з підтримкою анімацій, що ідеально підходить для реалізації інтуїтивного `drag-and-drop` дизайну.

Обмеження.

Популярність Dart є нижчою порівняно з JavaScript чи Java, що може ускладнити пошук розробників із досвідом роботи з цією мовою для розширення проекту.

Розмір застосунку на Flutter більший порівняно з нативними (приблизно у два рази), що може створювати проблеми для користувачів із обмеженим простором на пристрої, хоча це відповідає вимозі до розміру (не більше 20 Мб).

Обмежена підтримка платформ проявляється у тому, що хоча Flutter активно розвивається, підтримка десктопних і веб версій ще не настільки зріла, як для мобільних платформ, що може стримувати майбутнє розширення застосунку.

Порівняння мов програмування наведено в таблиці 2.1.

Таблиця 2.1 – Порівняння мов програмування

Критерій	Dart	Java	Swift
Кросплатформність	Висока (Flutter)	Середня (Android)	Низька (iOS)
Продуктивність	Висока (AOT)	Висока (JVM)	Висока (нативна)
Синтаксис	Чистий, простий	Складний	Чистий
Підтримка	Google	Oracle	Apple
Екосистема	Розвинена	Дуже розвинена	Розвинена
Розмір застосунку	10–20 Мб	5–15 Мб	5–10 Мб
Швидкість розробки	Висока (Hot Reload)	Середня	Середня
Критерій	Dart	Java	Swift

Dart обрано через його кросплатформність, продуктивність і зручність розробки, що ідеально відповідає потребам проекту.

2.3.2 Середовище розробки

Для розробки застосунку обрано Android Studio – інтегроване середовище розробки (IDE), яке є стандартом для створення Android–

застосунків і офіційно підтримує Flutter. Це рішення обґрунтоване такими причинами:

Офіційна підтримка Flutter.

Android Studio є рекомендованим середовищем для розробки на Flutter, що забезпечує повну інтеграцію з Dart і Flutter SDK. Це включає автодоповнення коду, інструменти налагодження та профілювання.

Потужні інструменти.

Середовище надає широкий набір функцій: редактор коду з інтелектуальним автодоповненням, вбудований емулятор, підтримка Git, профілювання продуктивності та пам'яті.

Підтримка плагінів.

Android Studio підтримує плагіни для Dart і Flutter, що спрощують розробку, тестування та розгортання застосунку.

Емулятор.

Вбудований емулятор дозволяє тестувати застосунок на різних версіях Android без фізичних пристроїв, що прискорює процес відлагодження.

Інтеграція з Git.

Вбудована підтримка систем контролю версій полегшує командну роботу та відстеження змін у коді.

2.3.2.1 Обґрунтування вибору та опис його можливостей

Android Studio обрано через його комплексність і адаптованість до Flutter, що відповідає вимогам до швидкості розробки та тестування. Основні можливості.

Редактор коду: інтелектуальне автодоповнення, рефакторинг, швидкий доступ до документації.

Налагодження: підтримка точок зупину, перегляду змінних, стеку викликів.

Тестування: інтеграція з інструментами для юніт-тестування та UI-тестування.

Профілювання: аналіз використання CPU, пам'яті та мережі для оптимізації продуктивності.

Візуальний редактор: Flutter Inspector для налаштування інтерфейсу в реальному часі.

Порівняння середовищ розробки наведено в таблиці 2.2.

Таблиця 2.2 – Порівняння середовищ розробки

Критерій	Android Studio	Visual Studio Code	Xcode
Підтримка Flutter	Повна	Часткова	Відсутня
Інструменти	Потужні	Середні	Потужні (iOS)
Емулятор	Android	Зовнішній	iOS
Інтеграція з Git	Вбудована	Вбудована	Вбудована
Плагіни	Багато	Багато	Обмежено
Розмір	~1 Гб	~200 Мб	~10 Гб

Android Studio обрано через його спеціалізацію на Flutter і наявність усього необхідного для розробки, тестування та оптимізації.

2.3.3 База даних

Для забезпечення роботи застосунку використано дві бази даних: локальну (SQLite з бібліотекою Drift) [9]-[10] для офлайн-режиму та хмарну (Firebase Firestore) [11] для синхронізації даних у реальному часі.

2.3.3.1 SQLite з Drift

SQLite – це легка реляційна база даних, яка працює на пристрої. Drift – це бібліотека для Dart, яка спрощує роботу з SQLite, надаючи типобезпечний доступ і автоматичну генерацію коду.

Переваги:

- офлайн–доступ. Дані зберігаються локально, що дозволяє працювати без підключення до інтернету;
- швидкість. Локальна база забезпечує швидкий доступ до даних без затримок мережі;
- безпека. Дані шифруються на пристрої за допомогою бібліотеки encrypt.

Обмеження. Необхідність синхронізації з хмарою для обміну даними між пристроями.

2.3.3.2 Firebase Firestore

Хмарна NoSQL–база даних від Google, яка підтримує реальний час і автоматичну синхронізацію.

Переваги:

- синхронізація в реальному часі: дані автоматично оновлюються на всіх пристроях сім'ї;
- масштабованість: хмарна інфраструктура дозволяє легко масштабувати застосунок;
- безпека: вбудовані механізми автентифікації та шифрування передачі даних.

Обмеження: Залежність від інтернету для синхронізації, що компенсується локальною базою.

Для порівняння баз даних розглянуто також PostgreSQL – реляційну базу даних, яка могла б бути альтернативою для серверної реалізації. PostgreSQL підтримує складні запити та великі обсяги даних, але потребує серверної інфраструктури, що ускладнює її використання в мобільному застосунку з офлайн–режимом. Комбінація SQLite із Drift і Firebase Firestore обрана як оптимальне рішення, оскільки вона забезпечує баланс між швидкодією, офлайн–доступом і синхронізацією, що ідеально відповідає вимогам проєкту.

Порівняння баз даних наведено в таблиці 2.3.

Таблиця 2.3 – Порівняння баз даних

Критерій	SQLite з Drift	Firebase Firestore	PostgreSQL
Тип бази даних	Реляційна (локальна)	NoSQL (хмарна)	Реляційна (серверна)
Офлайн–доступ	Високий	Низький	Відсутній
Синхронізація	Відсутня	В реальному часі	Через сервер
Швидкість запитів	Висока (локально)	Середня (залежить від мережі)	Висока (на сервері)
Масштабованість	Низька	Висока	Висока
Безпека	Шифрування (encrypt)	Вбудоване шифрування	Шифрування (на сервері)
Розмір бази	Менше 10 Мб	Залежить від даних	Залежить від сервера
Складність інтеграції	Проста (Drift)	Середня (Firebase SDK)	Складна (сервер)
Підтримка	Google, спільнота	Google	Спільнота

Комбінація SQLite з Drift і Firebase Firestore забезпечує баланс між швидкістю, офлайн–доступом і синхронізацією, що ідеально відповідає вимогам застосунку.

2.4 Вимоги до технічних і програмних засобів

2.4.1 Необхідні технічні засоби для розробки та експлуатації

Для розробки та експлуатації застосунку «Планувальник сімейного бюджету» необхідні такі технічні засоби:

- комп’ютер для розробки: для написання коду, компіляції та тестування;
- смартфон або планшет: для тестування застосунку на реальних пристроях (Android 6.0+ або iOS 12+);
- інтернет–з’єднання: для синхронізації даних із Firebase та завантаження оновлень.

2.4.2 Вимоги до апаратного та програмного забезпечення

Апаратне забезпечення:

- процесор: Intel Core i5 або еквівалент (для швидкої компіляції Flutter);
- оперативна пам'ять: 8 Гб (рекомендується 16 Гб для комфортної роботи з емуляторами);
- вільне місце на диску: 10 Гб (включаючи Android Studio, Flutter SDK і залежності);
- смартфон/планшет: сумісний з Android 6.0+ (API 23) або iOS 12+ для тестування.

Програмне забезпечення:

- операційна система: Windows 10/11, macOS 10.14+, або Linux (Ubuntu 18.04+);
- Android Studio: версія 4.0 або новіша для інтеграції з Flutter;
- Flutter SDK: версія 2.0+ для кросплатформної розробки;
- Dart SDK: версія 2.12+ як основна мова програмування;
- Firebase CLI: для розгортання та налаштування хмарних сервісів.

2.5 Висновки за розділом 2

У другому розділі було описано архітектуру мобільного застосунку «Планувальник сімейного бюджету», визначено основні компоненти системи та їхню взаємодію. Вибір мови програмування Dart і фреймворку Flutter обґрунтовано їхньою кросплатформністю, високою продуктивністю та зручністю розробки, що дозволяє створювати застосунок для Android та iOS із єдиною кодовою базою. Android Studio обрано як середовище розробки завдяки його інтеграції з Flutter і потужним інструментам для написання, тестування та налагодження коду. Використання локальної бази даних SQLite (з бібліотекою Drift) у поєднанні з хмарною Firebase Firestore забезпечує баланс між офлайн-доступом і синхронізацією даних у реальному часі.

З РОЗРОБКА ПРОГРАМИ

3.1 Архітектура системи

Архітектура застосунку «Планувальник сімейного бюджету» базується на шаблоні проєктування MVVM (Model–View–ViewModel) з використанням Provider для управління станом. Така архітектура забезпечує чіткий розподіл відповідальності між компонентами, що полегшує тестування, підтримку та розширення функціональності.

Основні компоненти архітектури.

Model (Модель) — представляє дані та бізнес–логіку застосунку:

- класи моделей даних (Account, Transaction, User, Family);
- база даних (AppDatabase з використанням Drift).

View (Представлення) — відповідає за відображення інтерфейсу користувача:

- екрани (HomeScreen, AccountsScreen, StatsScreen тощо);
- віджети (BudgetCircle, UserAvatar, DragDropWidget).

ViewModel (Модель представлення) — з'єднує Model і View, обробляє дані для відображення:

- провайдери (UserProvider);
- сервіси (AuthService, DbService, EncryptionService тощо).

Services (Сервіси) — забезпечують функціональність для різних частин застосунку:

- AuthService — автентифікація користувачів;
- DbService — робота з базою даних;
- EncryptionService — шифрування даних;
- AIService — інтеграція з ІІІ для рекомендацій;
- CurrencyService — конвертація валют;
- BackupService — резервне копіювання даних;
- NFCService — обмін даними через NFC.

Схема взаємодії компонентів системи зображена на рисунку 3.1.

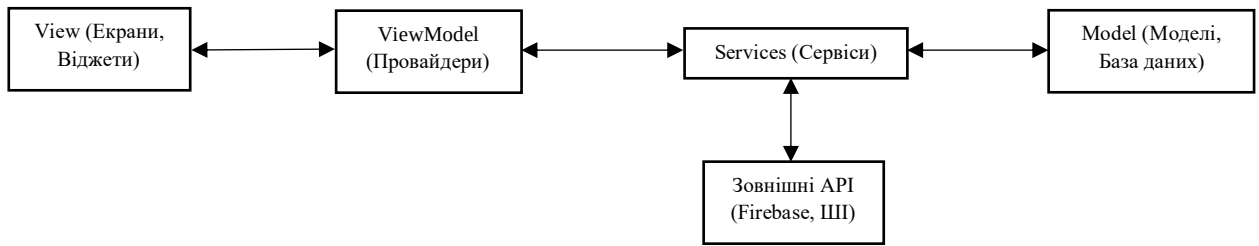


Рисунок 3.1 – Схема взаємодії компонентів системи

3.1.1 Структурна схема системи

Проект організовано за принципом функціональної декомпозиції, де код розділено на логічні модулі відповідно до їх призначення.

Структурну схему системи зображено на рисунку 3.2.

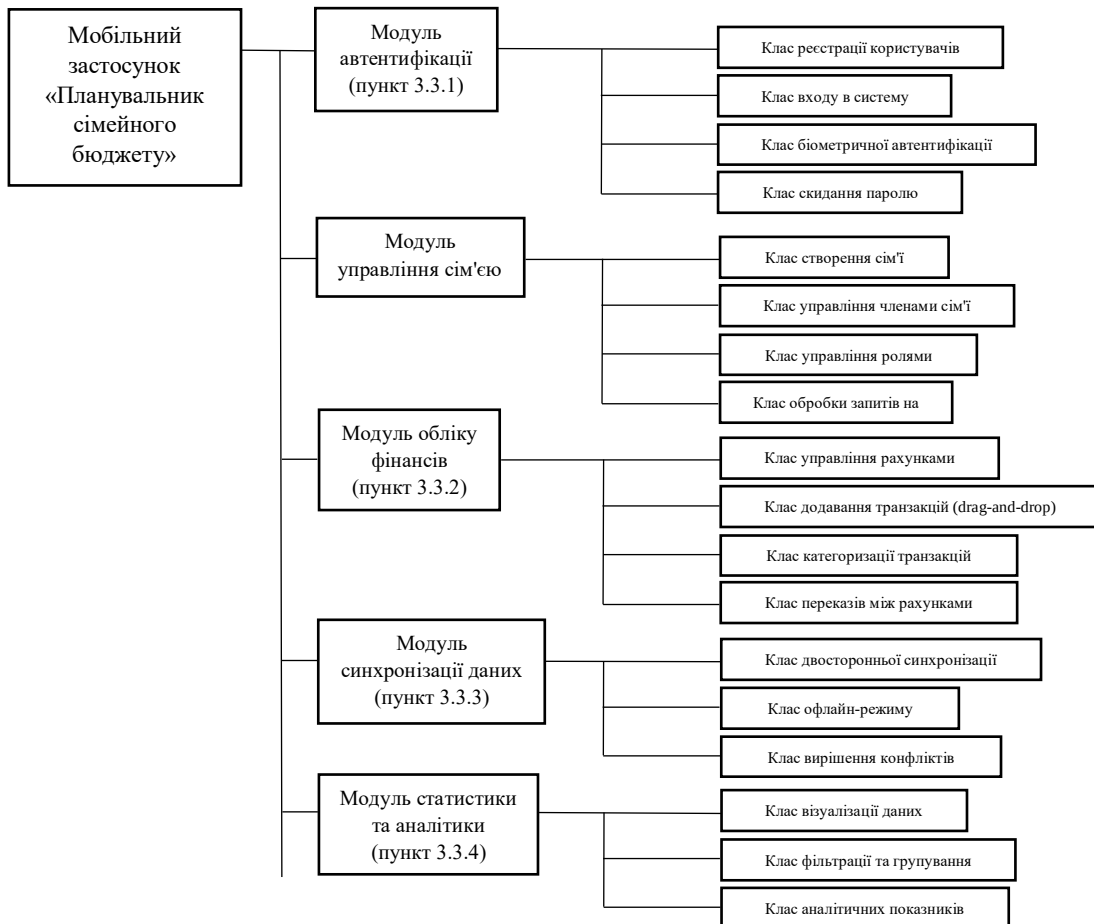


Рисунок 3.2 – Структурна схема системи

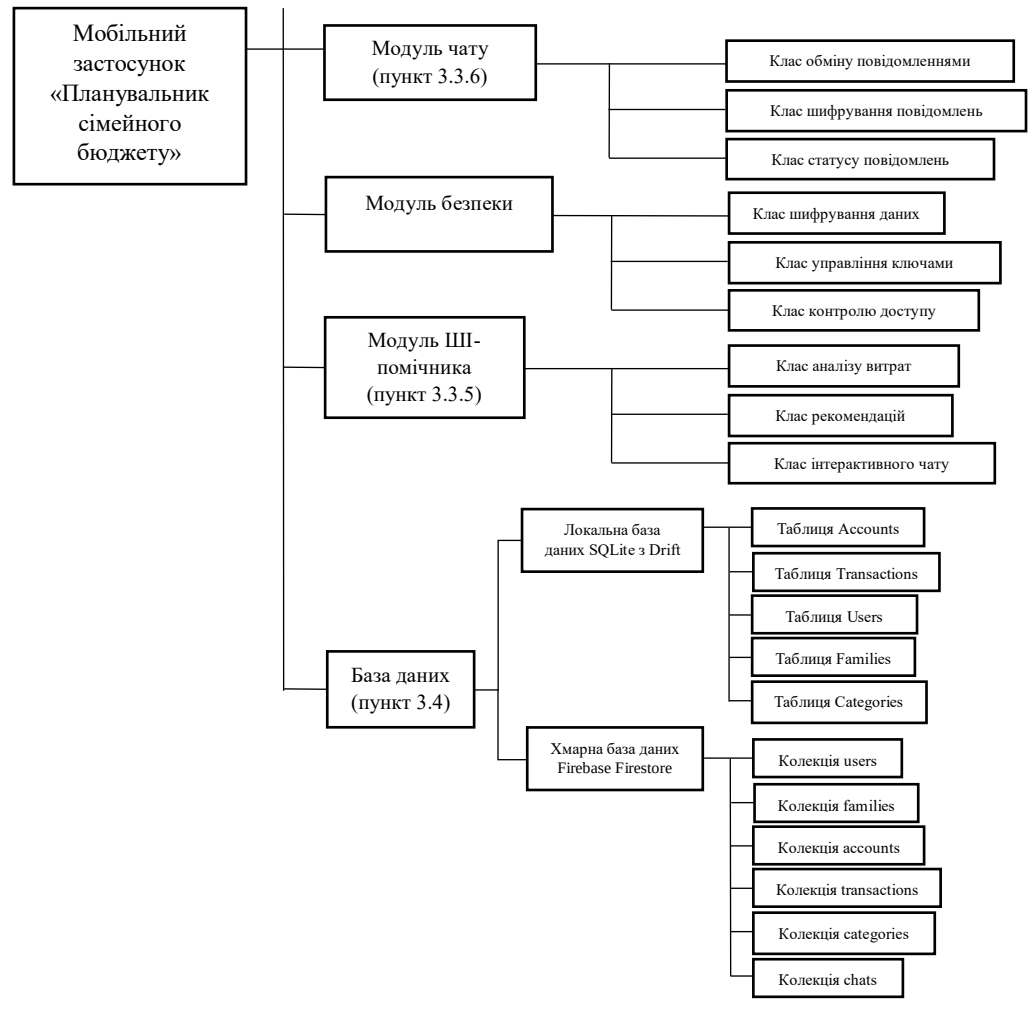


Рисунок 3.2, лист 2

3.2 Функціонування системи

3.2.1 Функціональна схема

Мобільний застосунок «Планувальник сімейного бюджету» функціонує за складною системою алгоритмів, які забезпечують його роботу як у режимі онлайн, так і офлайн. Розглянемо основні алгоритми, що лежать в основі програми (рис. 3.3).

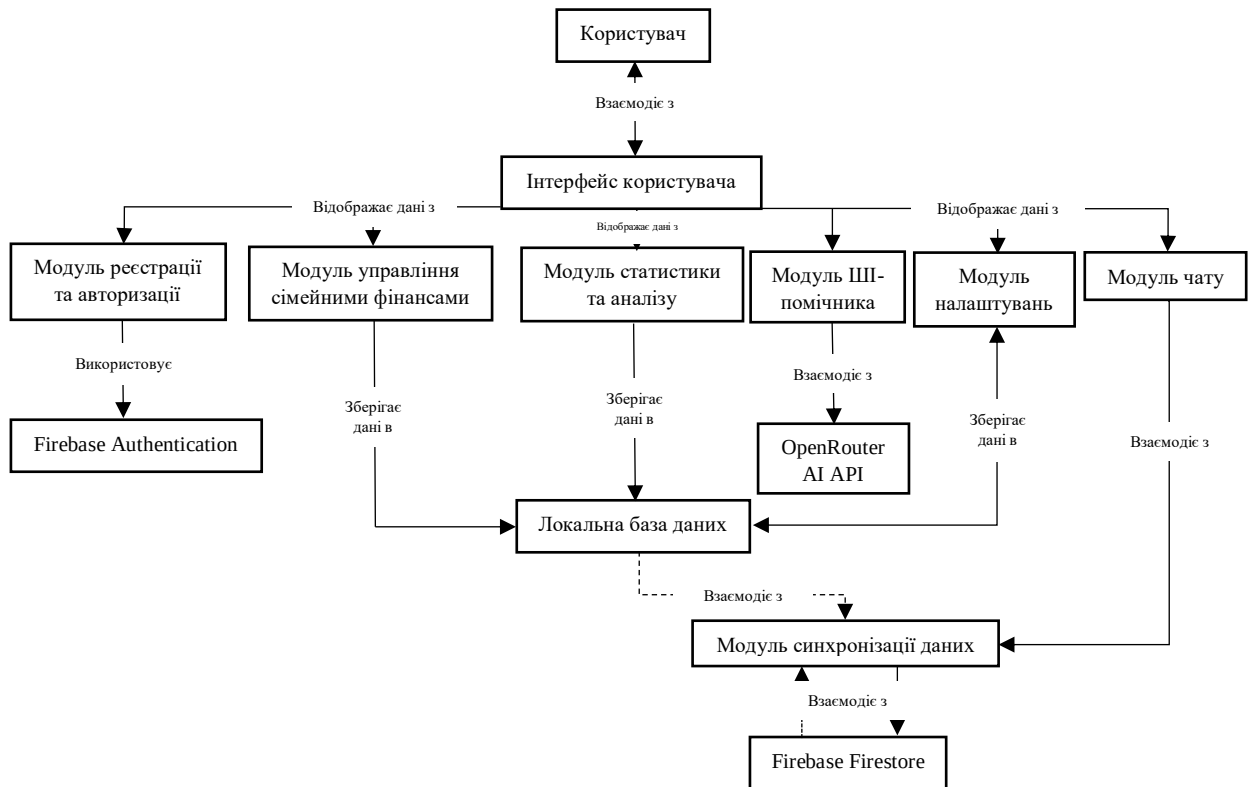


Рисунок 3.3 – Функціональна схема розробленої програми

3.2.2 Опис процесу обміну даними у системі

Процес обміну даними у системі організовано за принципом реактивного програмування, де зміни в даних автоматично відображаються в інтерфейсі користувача. Основні етапи потоку даних (рис.3.4):

- введення даних користувачем. Через інтерфейс користувач вводить дані (наприклад, додає транзакцію);
- обробка даних у ViewModel. Провайдер обробляє дані та викликає відповідні сервіси;
- збереження даних у локальній базі. Дані зберігаються в локальній базі SQLite через Drift;
- синхронізація з хмарою. Дані синхронізуються з Firebase Firestore (якщо є підключення);
- оновлення інтерфейсу. Зміни в даних автоматично відображаються в інтерфейсі через механізм Provider.

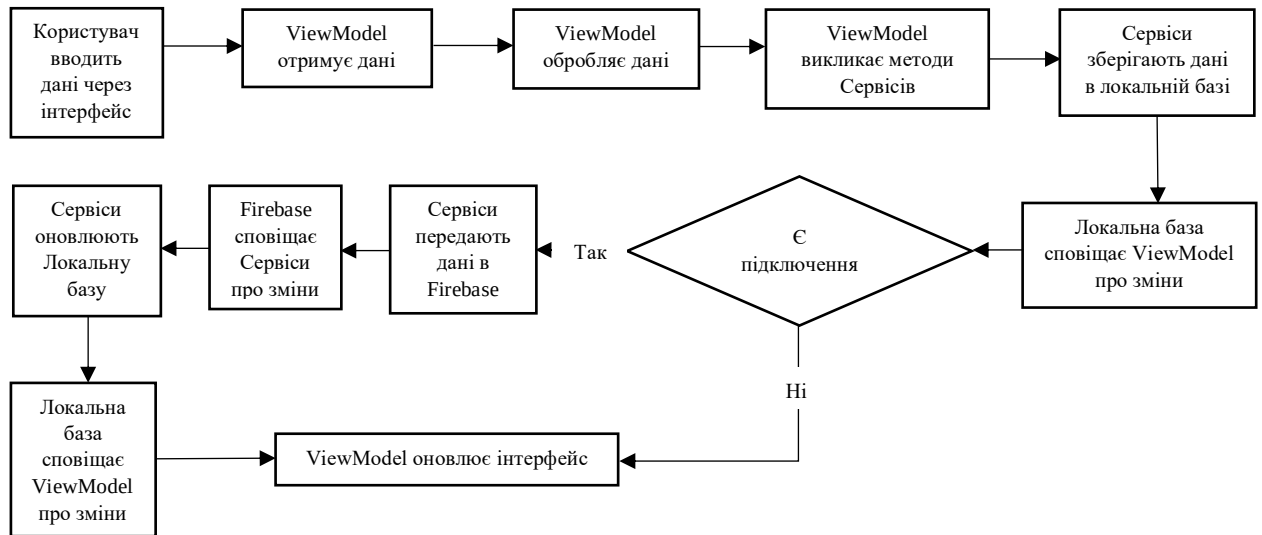


Рисунок 3.4 – Схема процесу обміну даними у системі

3.2.3 Сценарії використання

Основні сценарії використання системи.

Сценарій "Створення сім'ї та запрошення членів":

- користувач реєструється в системі;
- користувач створює сім'ю та отримує код або посилання на запрошення;
- користувач надсилає код іншим членам сім'ї;
- інші користувачі приєднуються до сім'ї за кодом, посиланням, або через пошук сімей через локальну мережу;
- адміністратор призначає ролі та права доступу.

Сценарій "Додавання витрати":

- користувач відкриває головний екран;
- користувач перетягує категорію витрат на рахунок;
- користувач вводить суму, дату, час та коментар;
- система зберігає витрату та оновлює баланс;
- дані синхронізуються з іншими пристроями.

Сценарій "Перегляд статистики":

- користувач відкриває екран статистики;
- користувач вибирає період та фільтри;

- система відображає графіки та діаграми;
- користувач отримує рекомендації від ШІ-помічника.

Сценарій "Спілкування в чаті":

- користувач відкриває екран чату;
- користувач пише повідомлення;
- система шифрує повідомлення та надсилає його;
- інші члени сім'ї отримують повідомлення в реальному часі.

Сценарій "Робота в офлайн-режимі":

- користувач втрачає з'єднання з інтернетом;
- користувач продовжує працювати з застосунком;
- система зберігає всі зміни локально;
- при відновленні з'єднання дані синхронізуються.

Схема розподілу прав доступу зображена на рисунку 3.5.

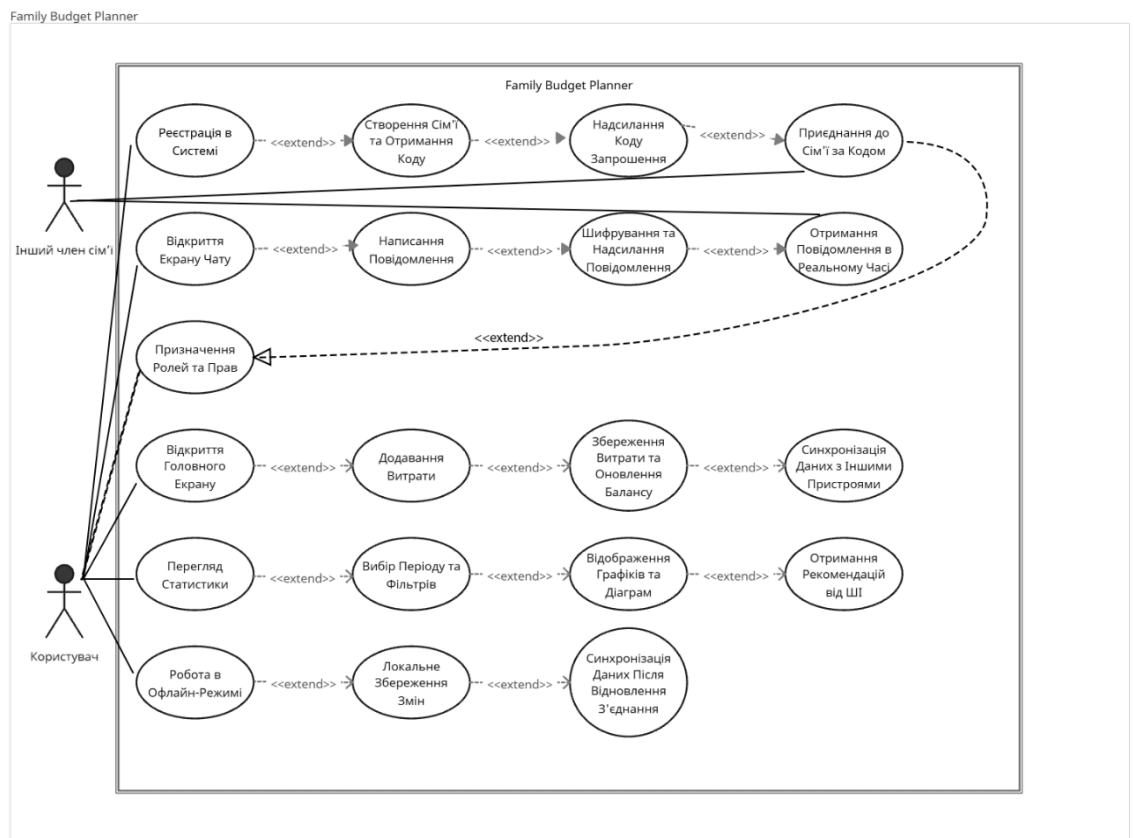


Рисунок 3.5 – Схема розподілу прав доступу

3.3 Опис модулів системи

3.3.1 Модуль реєстрації та авторизації

3.3.1.1 Алгоритм автентифікації та авторизації

Алгоритм автентифікації користувача включає кілька шляхів входу: через email/пароль, через Google-акаунт або за допомогою біометричних даних. Процес автентифікації відбувається за такою схемою (рис. 3.6):

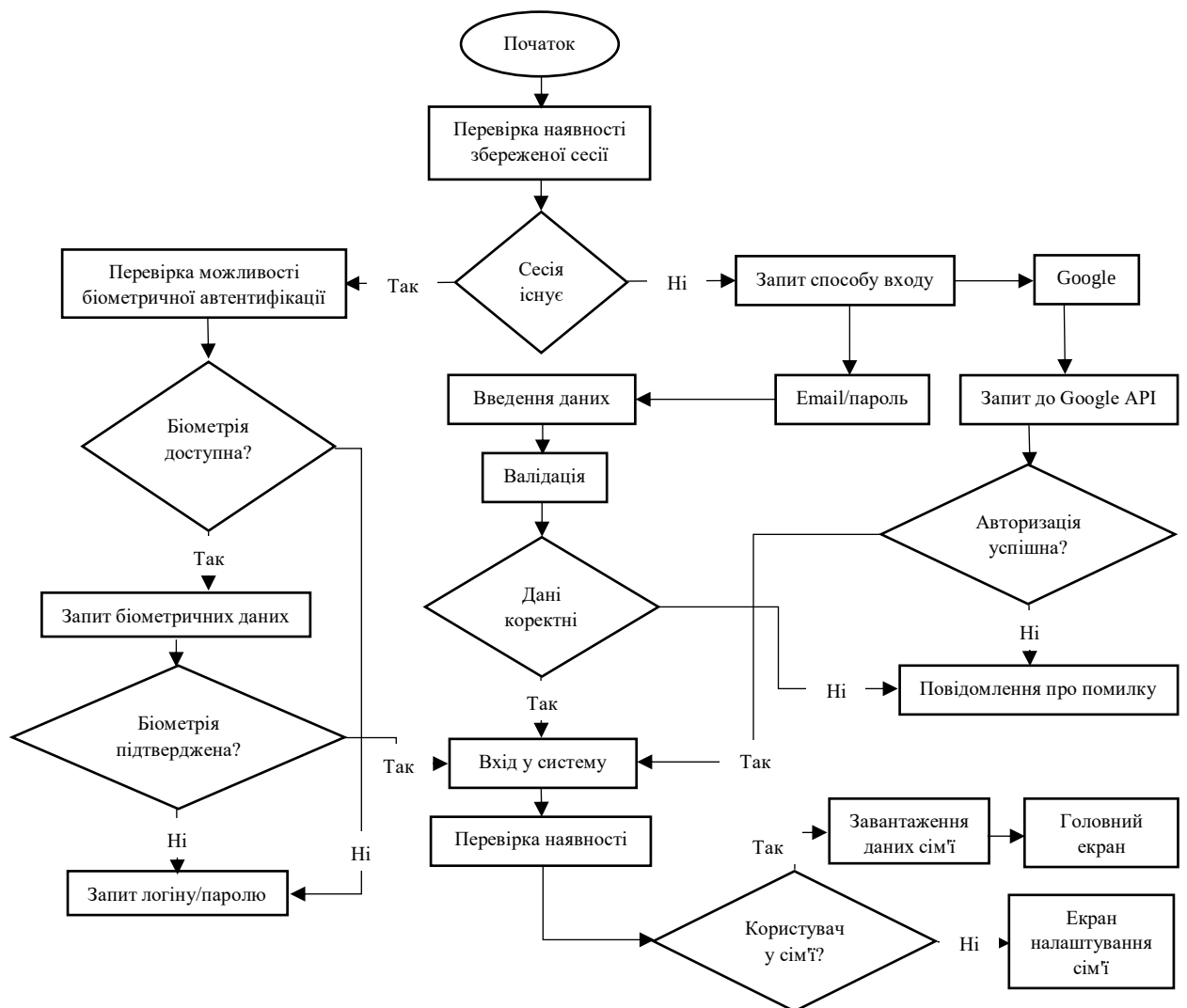


Рисунок 3.6 — Алгоритм автентифікації та авторизації користувача

3.3.1.2 Функціональність модуля

Модуль реєстрації та авторизації забезпечує такі функції.

Реєстрація нових користувачів:

- реєстрація через email та пароль;
- реєстрація через Google-акаунт;
- валідація введених даних;
- створення профілю користувача в Firebase.

Автентифікація існуючих користувачів:

- вхід через email та пароль;
- вхід через Google-акаунт;
- біометрична автентифікація (Face ID/Touch ID);
- збереження сесії користувача.

Управління профілем:

- оновлення даних профілю;
- зміна паролю;
- скидання паролю через email.

3.3.1.3 Реалізація та використані технології

Модуль реалізовано за допомогою таких технологій:

- Firebase Authentication — для управління обліковими записами користувачів;

- Google Sign-In — для інтеграції з Google-акаунтами;
- Local Authentication — для біометричної автентифікації;
- Flutter Secure Storage — для безпечного зберігання токенів автентифікації.

Основні класи та файли модуля:

- auth_service.dart — сервіс для роботи з автентифікацією;
- login_screen.dart — екран входу в систему;
- register_screen.dart — екран реєстрації;
- profile_settings_screen.dart — екран налаштування профілю.

Ключові фрагменти коду, такі як реєстрація через email та пароль, вхід через Google та біометрична автентифікація, наведено в Додатку Б.8.

Екран входу в систему реалізовано як StatefulWidget, який взаємодіє з AuthService для автентифікації користувача (рис.3.7). Реалізацію кодом наведено в Додатку Б.5.

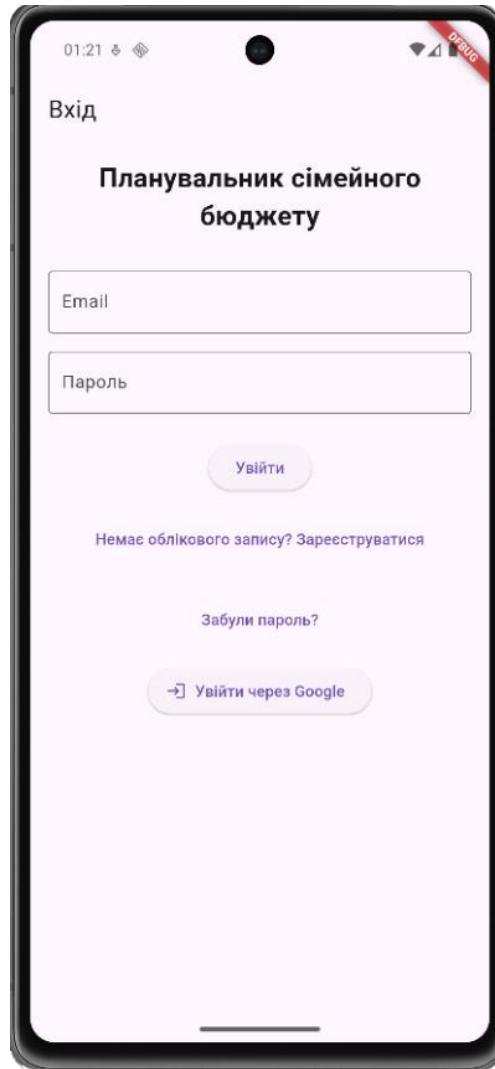


Рисунок 3.7 — Екран входу в систему

3.3.2 Модуль управління сімейними фінансами

3.3.2.1 Алгоритм додавання транзакції

Інтуїтивний інтерфейс drag-and-drop для додавання транзакцій реалізований за допомогою спеціального алгоритму, який обробляє перетягування категорій на рахунки або навпаки (рис. 3.8).

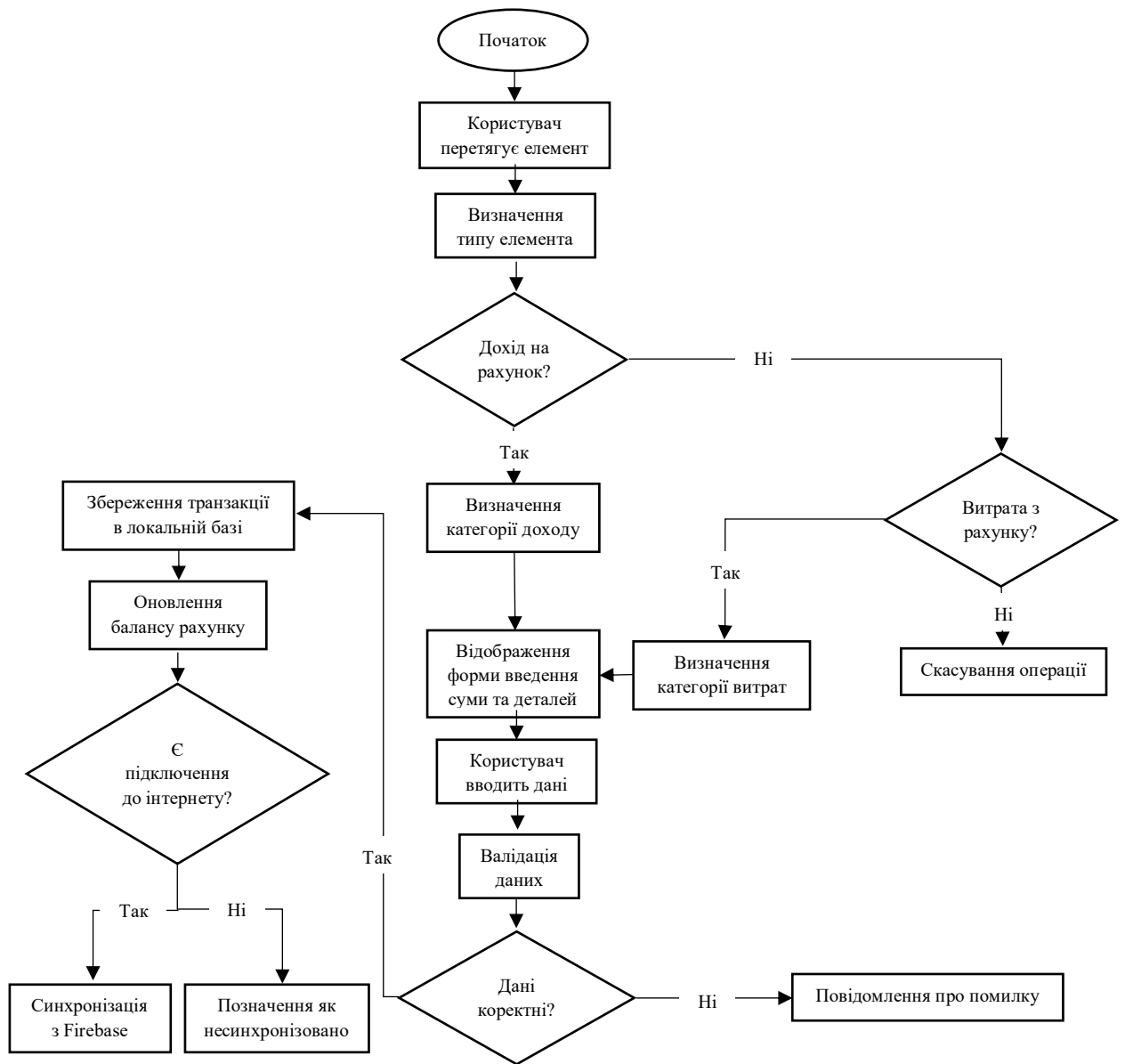


Рисунок 3.8 — Алгоритм додавання транзакції через drag-and-drop

3.3.2.2 Функціональність модуля

Модуль управління сімейними фінансами є центральним компонентом застосунку і забезпечує такі функції.

Управління рахунками:

- створення та редагування рахунків;
- перегляд балансу рахунків;
- переказ коштів між рахунками;
- налаштування доступу до рахунків для членів сім'ї.

Облік доходів і витрат:

- додавання транзакцій через інтуїтивний drag-and-drop інтерфейс;
- категоризація транзакцій;
- додавання коментарів та геометок до транзакцій;
- перегляд історії транзакцій.

Управління категоріями:

- створення власних категорій доходів і витрат;
- редагування та видалення категорій;
- сортування категорій за пріоритетом.

3.3.2.3 Реалізація та використані технології

Модуль реалізовано за допомогою таких технологій:

- Flutter Widgets — для створення інтерактивного інтерфейсу;
- Provider — для управління станом та передачі даних між компонентами;

- Drift (SQLite) — для локального зберігання даних;
- Firebase Firestore — для синхронізації даних між пристроями;
- Geolocator — для отримання геолокації транзакцій.

Основні класи та файли модуля:

- db_service.dart — сервіс для роботи з базою даних;
- home_screen.dart — головний екран з відображенням рахунків і категорій;

- accounts_screen.dart — екран управління рахунками;
- income_screen.dart — екран додавання доходів;
- expenses_screen.dart — екран управління витратами;
- drag_drop_widget.dart — віджет для drag-and-drop інтерфейсу.

Ключові фрагменти коду, такі як додавання транзакцій, показано в Додатку Б.4.

Інтерфейс drag-and-drop для додавання транзакцій реалізовано за допомогою віджетів Draggable та DragTarget (рис. 3.9). Реалізація кодом показана в Додатку Б.13.



Рисунок 3.9 — Інтерфейс drag-and-drop для додавання транзакцій

3.3.3 Модуль синхронізації даних

3.3.3.1 Алгоритм синхронізації даних

Одним із ключових алгоритмів застосунку є синхронізація даних між локальною базою даних (SQLite з Drift) та хмарною базою (Firebase Firestore) (рис. 3.10). Цей алгоритм забезпечує роботу застосунку в офлайн-режимі з подальшою синхронізацією при відновленні з'єднання.

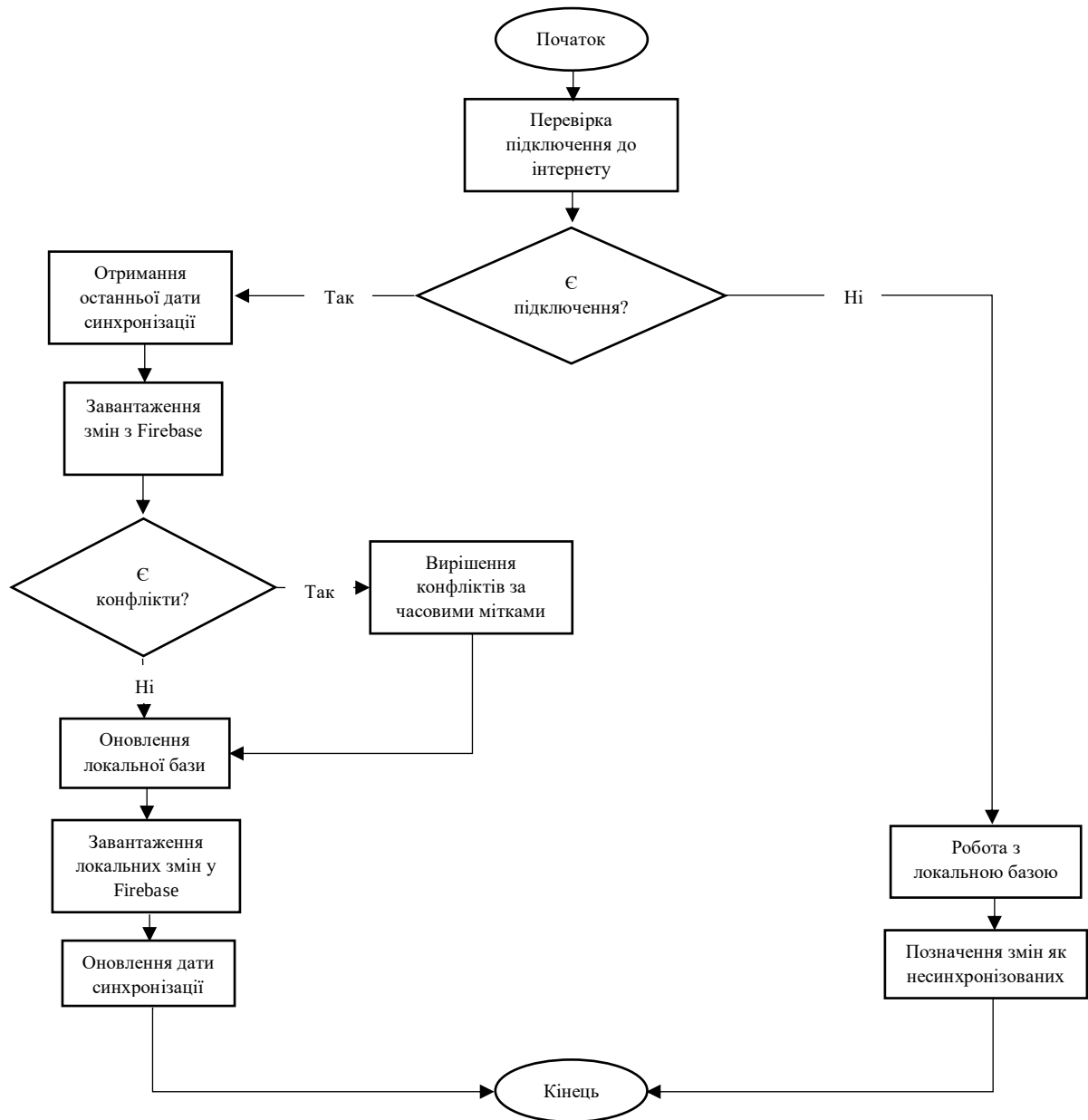


Рисунок 3.10 — Алгоритм синхронізації даних

Вирішення конфліктів відбувається за принципом "останні зміни перемагають", тобто запис з пізнішою часовою міткою має пріоритет. Це дозволяє уникнути втрати даних при одночасному редагуванні з різних пристроїв.

3.3.3.2 Функціональність модуля

Модуль синхронізації даних забезпечує безперебійну роботу застосунку як в онлайн, так і в офлайн режимі, а також синхронізацію даних між усіма пристроями членів сім'ї та примусову синхронізацію після натискання відповідної кнопки в налаштуваннях (рис. 3.11).

Основні функції модуля.

Двостороння синхронізація:

- завантаження змін з Firebase до локальної бази даних;
- вивантаження локальних змін до Firebase;
- відстеження змін у реальному часі.

Робота в офлайн-режимі:

- збереження всіх операцій у локальній базі даних;
- позначення несинхронізованих записів;
- автоматична синхронізація при відновленні з'єднання.

Вирішення конфліктів:

- виявлення конфліктуючих змін;
- вирішення конфліктів за часовими мітками;
- збереження цілісності даних.

3.3.3.3 Реалізація та використані технології

Модуль реалізовано за допомогою таких технологій:

- Drift (SQLite) — для локального зберігання даних;
- Firebase Firestore — для хмарного зберігання та синхронізації;
- Connectivity — для відстеження стану підключення до інтернету.

Основні класи та файли модуля:

- db_service.dart — сервіс для роботи з базою даних та синхронізації;
- database.dart — визначення структури локальної бази даних;
- user_provider.dart — провайдер для управління станом користувача та синхронізації.

Ключові фрагменти коду, такі як синхронізація всіх даних, синхронізація рахунків з Firebase, синхронізація транзакцій з Firebase та примусова синхронізація всіх даних, наведено в Додатку Б.10.

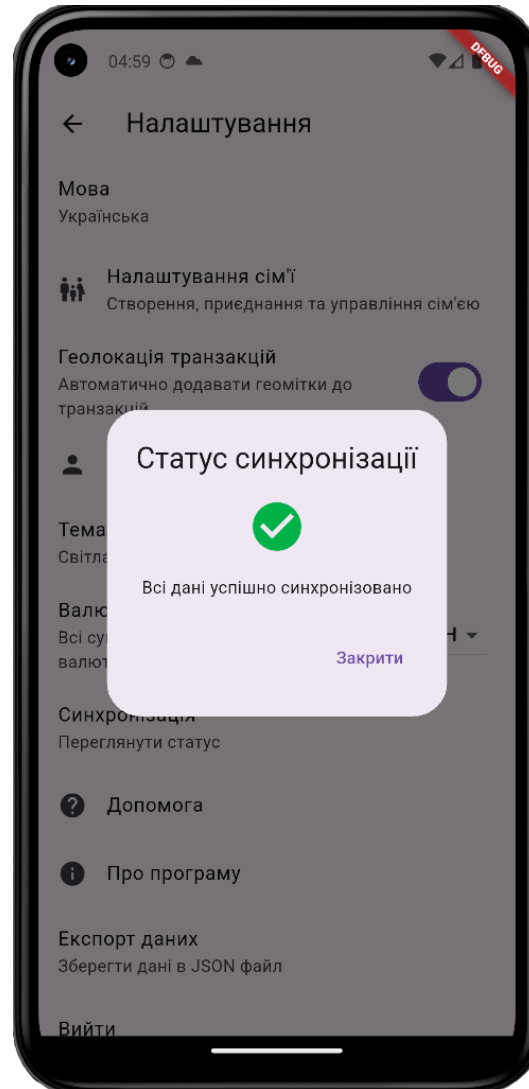


Рисунок 3.11 — Примусова синхронізація даних відповідним пунктом в налаштуваннях

3.3.4 Модуль статистики та аналізу

3.3.4.1 Функціональність модуля

Модуль статистики та аналізу надає користувачам детальну інформацію про їхні фінанси та допомагає аналізувати витрати й доходи.

Основні функції модуля.

Візуалізація даних:

- кругові діаграми розподілу витрат за категоріями;
- графіки динаміки доходів і витрат;
- порівняльні діаграми за різні періоди.

Фільтрація та групування:

- фільтрація за періодом (день, тиждень, місяць, рік);
- групування за категоріями, рахунками, користувачами;
- сортування за сумою або датою.

Аналітичні показники:

- середні витрати за категоріями;
- тренди зростання/зменшення витрат;
- порівняння з попередніми періодами.

3.3.4.2 Реалізація та використані технології

Модуль реалізовано за допомогою таких технологій:

- FL Chart — для створення інтерактивних графіків і діаграм;
- Intl — для форматування дат і чисел;
- Provider — для управління станом та передачі даних.

Основні класи та файли модуля:

- stats_screen.dart — екран статистики та аналізу;
- budget_circle.dart — віджет для відображення кругової діаграми;
- currency_service.dart — сервіс для конвертації валют.

Ключові фрагменти коду наведено в Додатку Б.6.

На рисунку 3.12 та 3.13 продемонстровано вигляд екрану статистики.

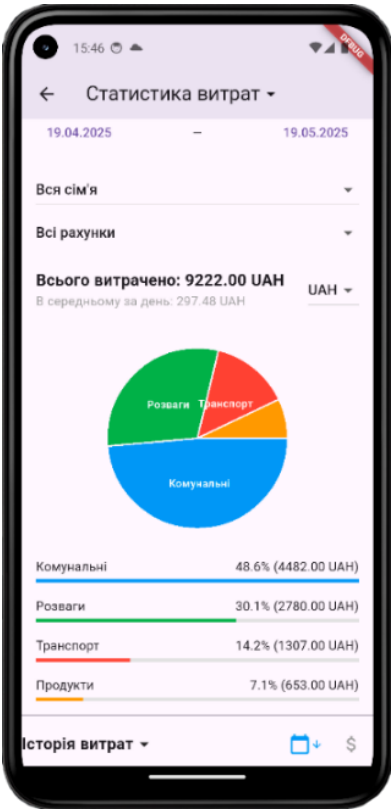


Рисунок 3.12 — Екран статистики з круговою діаграмою

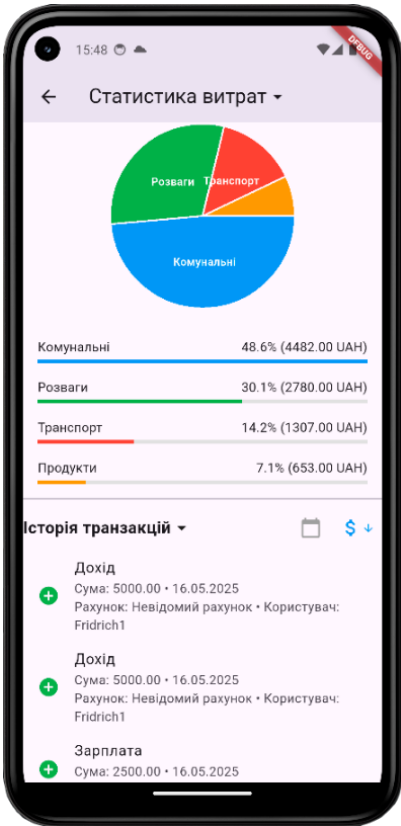


Рисунок 3.13 — Екран статистики з круговою діаграмою

3.3.5 Модуль ІІІ–помічника

3.3.5.1 Алгоритм аналізу витрат та рекомендацій за допомогою ІІІ

Модуль ІІІ–помічника використовує алгоритм аналізу фінансових даних для надання персоналізованих рекомендацій щодо оптимізації витрат (рис. 3.14).



Рисунок 3.14 — Алгоритм аналізу витрат та надання рекомендацій за допомогою ІІІ

3.3.5.2 Функціональність модуля

Модуль ІІІ–помічника надає користувачам інтелектуальні рекомендації щодо оптимізації витрат та управління фінансами.

Основні функції модуля.

Аналіз витрат:

- виявлення аномальних витрат;
- аналіз тенденцій витрат за категоріями;
- порівняння з середніми показниками.

Рекомендації щодо оптимізації:

- персоналізовані поради щодо економії;
- пропозиції щодо перерозподілу бюджету;
- рекомендації щодо фінансових цілей.

Інтерактивний чат:

- відповіді на запитання про фінанси;
- пояснення фінансових термінів;
- допомога з плануванням бюджету.

3.3.5.3 Реалізація та використані технології

Модуль реалізовано за допомогою таких технологій:

- HTTP — для взаємодії з API штучного інтелекту;
- OpenRouter API — для доступу до моделей ШІ [12];
- JSON — для обробки відповідей API.

Основні класи та файли модуля:

- ai_service.dart — сервіс для взаємодії з API штучного інтелекту;
- ai_screen.dart — екран ШІ-помічника.

Екран ШІ-помічника реалізовано як StatefulWidget з двома вкладками: для рекомендацій та чату (рис. 3.15 та 3.16). Ключові фрагменти коду наведено в Додатку Б.7.

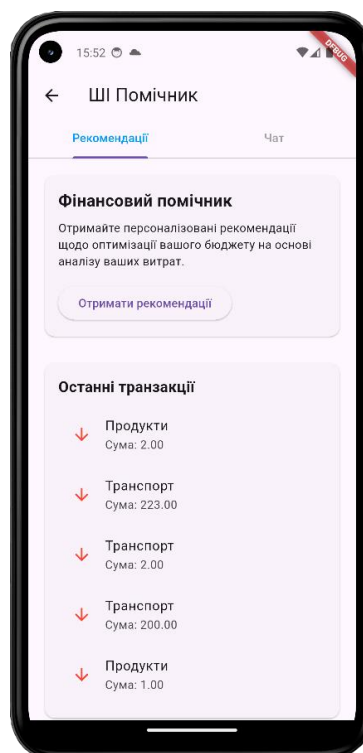


Рисунок 3.15 — Екран ШІ-помічника з рекомендаціями та чатом

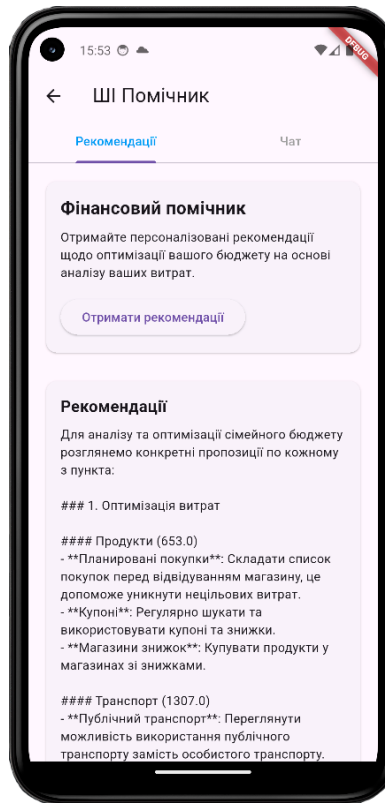


Рисунок 3.16 — Екран ШІ-помічника з рекомендаціями та чатом

Чат з ШІ помічником зображено на риунку 3.17.

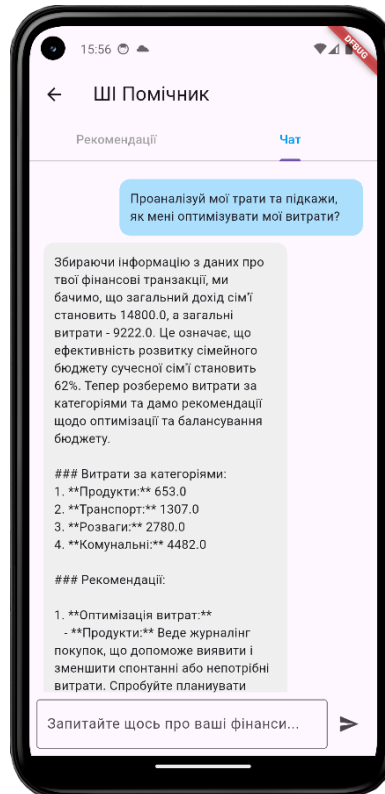


Рисунок 3.17 — Екран ШІ-помічника з рекомендаціями та чатом

3.3.6 Модуль чату

3.3.6.1 Алгоритм шифрування даних

Безпека даних забезпечується алгоритмом шифрування, який використовує гібридний підхід: AES–256 для шифрування даних [13] та безпечне зберігання ключів (рис. 3.18).

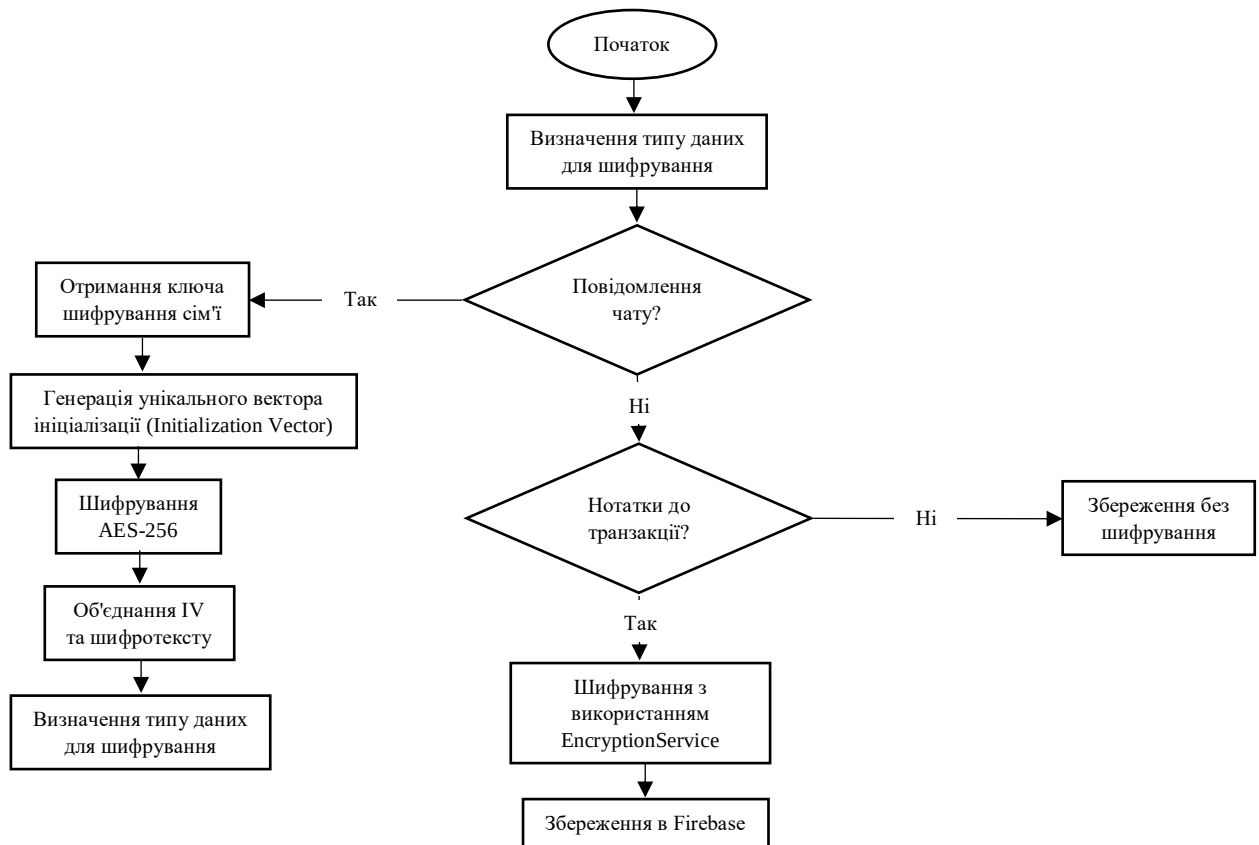


Рисунок 3.18 — Алгоритм шифрування даних

3.3.6.2 Функціональність модуля

Модуль чату забезпечує безпечне спілкування між членами сім'ї безпосередньо в застосунку.

Основні функції модуля.

Обмін повідомленнями:

- надсилання та отримання текстових повідомлень;
- відображення історії повідомлень;

– індикація статусу повідомлень (надіслано, доставлено, прочитано).

Безпека повідомлень:

- End-to-end шифрування повідомлень;
- унікальний ключ шифрування для кожної сім'ї;
- зберігання повідомлень у зашифрованому вигляді.

Інтерфейс чату:

- зручний інтерфейс у стилі сучасних месенджерів;
- відображення аватарів користувачів;
- перегляд профілів користувачів.

3.3.6.3 Реалізація та використані технології

Модуль реалізовано за допомогою таких технологій:

- Firebase Firestore — для зберігання та синхронізації повідомлень;
- Encrypt — для шифрування повідомлень (AES-256);
- StreamBuilder — для відображення повідомлень у реальному часі.

Основні класи та файли модуля:

- chat_screen.dart — екран чату;
- encryption_service.dart — сервіс для шифрування повідомлень;
- user_avatar.dart — віджет для відображення аватарів користувачів.

Сервіс шифрування реалізовано з використанням бібліотеки encrypt.

Ключові фрагменти коду наведені в Додатку Б.2 та Б.11.

Вигляд екрану чату продемонстровано на рисунку 3.19.

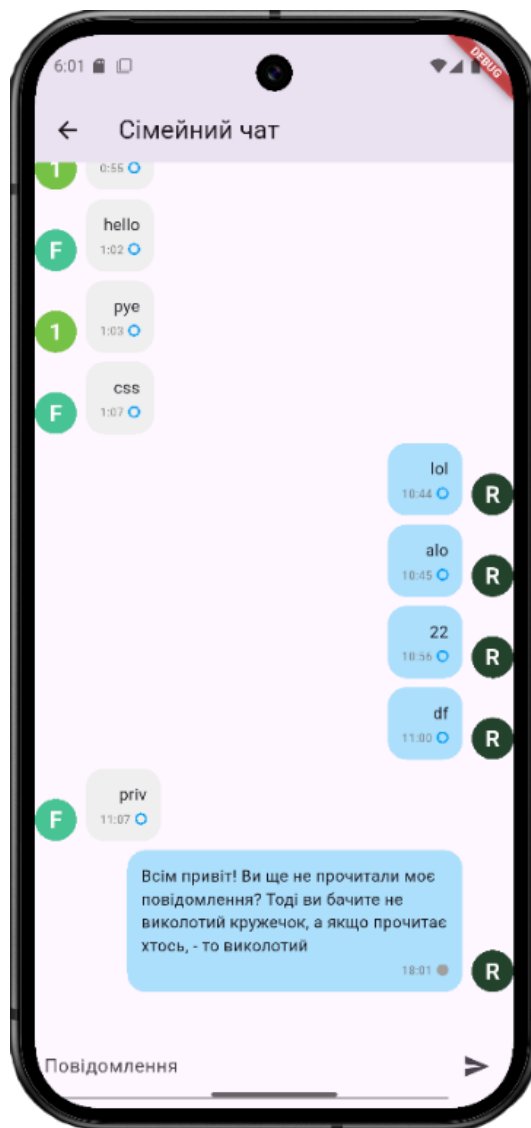


Рисунок 3.19 — Екран чату з шифрованими повідомленнями

3.3.7 Модуль налаштувань

3.3.7.1 Функціональність модуля

Модуль налаштувань дозволяє користувачам персоналізувати застосунок відповідно до їхніх потреб.

Основні функції модуля.

Налаштування профілю:

- редагування особистої інформації;
- зміна фото профілю;
- налаштування безпеки.

Налаштування сім'ї:

- управління членами сім'ї;
- налаштування ролей та прав доступу;
- генерація та поширення кодів запрошення.

Загальні налаштування:

- вибір мови інтерфейсу;
- вибір валюти за замовчуванням;
- налаштування сповіщень;
- резервне копіювання та відновлення даних.

3.3.7.2 Реалізація та використані технології

Модуль реалізовано за допомогою таких технологій:

- Flutter Widgets — для створення інтерфейсу налаштувань;
- Firebase Storage — для зберігання фото профілю;
- Image Picker — для вибору зображень з галереї;
- Share Plus — для поширення кодів запрошення.

Основні класи та файли модуля:

- settings_screen.dart — головний екран налаштувань;
- profile_settings_screen.dart — екран налаштування профілю;
- family_settings_screen.dart — екран налаштування сім'ї;
- backup_service.dart — сервіс для резервного копіювання.

Екран налаштування сім'ї дозволяє створювати сім'ю, запрошувати нових членів та управляти ролями.

Ключові фрагменти коду наведено в Додатку Б.3.

Екран налаштувань застосунку продемонстровано на рисунку 3.20.

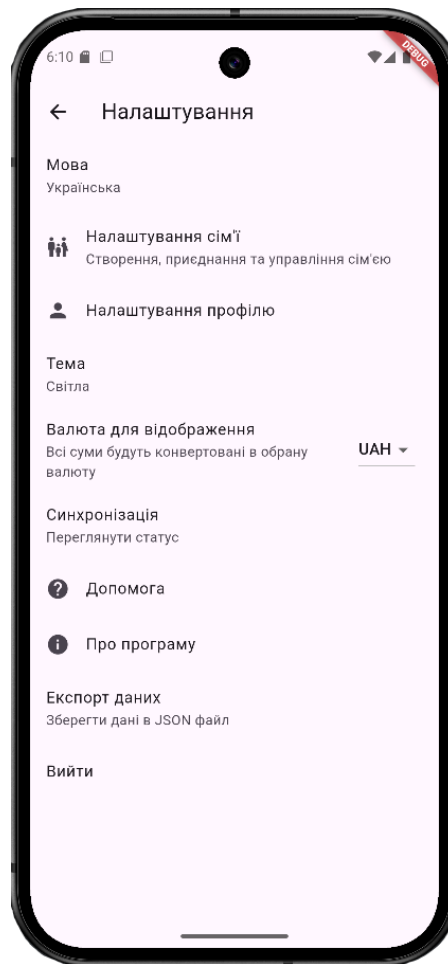


Рисунок 3.20 — Екран налаштувань застосунку

3.4 Проєктування бази даних

3.4.1 Структура бази даних

База даних мобільного застосунку «Планувальник сімейного бюджету» складається з двох частин: локальної бази даних, реалізованої за допомогою SQLite із використанням ORM Drift, і хмарної бази даних на основі Firebase Firestore. Локальна база даних використовує реляційну модель із таблицями, що забезпечує швидкий доступ до даних у режимі офлайн. Хмарна база даних застосовує документо-орієнтовану модель із колекціями, що підтримує синхронізацію даних між пристроями. Обидві бази мають схожу структуру, але адаптовані до особливостей своїх платформ.

Основними сутностями бази даних є користувачі, сімейні групи, фінансові рахунки, транзакції та категорії. Ці сутності відображають ключові

аспекти функціоналу застосунку, такі як управління фінансами, синхронізація даних і організація сімейного бюджету. Структура бази даних детально описана в таблиці 3.1, яка наводить перелік сутностей, їхні атрибути, типи даних і описи.

Таблиця 3.1 – Опис концептуальної моделі бази даних

№	Сутність	Опис	Атрибут	Тип даних	Опис
1	2	3	4	5	6
1	Users	Користувачі	id	INTEGER	Унікальний ідентифікатор
			serverId	TEXT	Ідентифікатор для хмарної синхронізації
			email	TEXT	Email користувача
			role	TEXT	Роль користувача (admin, child тощо)
			familyId	TEXT	ID сім'ї (FK до Families.id)
			permissions	TEXT	Права доступу (JSON)
			deviceModel	TEXT	Модель пристрою
			deviceLocation	TEXT	Локація пристрою
			synced	BOOLEAN	Статус синхронізації
			lastUpdated	DATETIME	Час останнього оновлення

Продовження таблиці 3.1

1	2	3	4	5	6
1	Users	Користувачі	lastSync	DATETIME	Час останньої синхронізації
			lastOnline	DATETIME	Час останнього онлайн
			displayName	TEXT	Ім'я користувача
			profileImageUrl	TEXT	URL фото профілю
			useGeoLocation	BOOLEAN	Використання геолокації
2	Families	Сім'ї	id	INTEGER	Унікальний ідентифікатор
			serverId	TEXT	Ідентифікатор для хмарної синхронізації
			name	TEXT	Назва сім'ї
			adminId	TEXT	ID адміністратора (FK до Users.id)
			synced	BOOLEAN	Статус синхронізації
			lastUpdated	DATETIME	Час останнього оновлення
			inviteCode	TEXT	Код запрошення до сім'ї
			pendingMembers	TEXT	Список очікуваних членів (JSON)

Продовження таблиці 3.1

1	2	3	4	5	6
2	Families	Сім'ї	members	TEXT	Список членів сім'ї (JSON)
			createdAt	DATETIME	Час створення
			encryptionKey	TEXT	Ключ шифрування для чату
3	Accounts	Рахунки	id	INTEGER	Унікальний ідентифікатор
			serverId	TEXT	Ідентифікатор для хмарної синхронізації
			accountId	TEXT	Унікальний ID рахунку
			name	TEXT	Назва рахунку
			balance	REAL	Баланс рахунку
			currency	TEXT	Валюта рахунку
			userId	TEXT	ID користувача (FK до Users.id)
			familyId	TEXT	ID сім'ї (FK до Families.id)
			accessibleToUsers	TEXT	Список користувачів із доступом (JSON)
			synced	BOOLEAN	Статус синхронізації

Продовження таблиці 3.1

1	2	3	4	5	6
3	Accounts	Рахунки	lastUpdated	DATETIME	Час останнього оновлення
			isShared	BOOLEAN	Спільний рахунок чи ні
4	Transactions	Транзакції	id	INTEGER	Унікальний ідентифікатор
			serverId	TEXT	Ідентифікатор для хмарної синхронізації
			amount	REAL	Сума транзакції
			category	TEXT	Категорія транзакції (FK до Categories.id)*
			isExpense	BOOLEAN	Тип транзакції (витрата/дохід)
			date	DATETIME	Дата транзакції
			accountId	TEXT	ID рахунку (FK до Accounts.id)
			userId	TEXT	ID користувача (FK до Users.id)
			familyId	TEXT	ID сім'ї (FK до Families.id)

Продовження таблиці 3.1

1	2	3	4	5	6
4	Transa ctions	Транзакції	notes	TEXT	Нотатки до транзакції (зашифровані)
			geoTag	TEXT	Геолокація транзакції
			synced	BOOLEA N	Статус синхронізації
			lastUpdated	DATETI ME	Час останнього оновлення
			currency	TEXT	Валюта транзакції
5	Categ ories	Категорії	id	INTEGER	Унікальний ідентифікатор
			name	TEXT	Назва категорії
			isExpense	BOOLEA N	Тип категорії (витрата/дохід)
			userId	TEXT	ID користувача (FK до Users.id)
			familyId	TEXT	ID сім'ї (FK до Families.id)
			sortOrder	INTEGER	Порядок сортування
			expenseCategories	TEXT	Список категорій витрат (JSON, Firestore)

Продовження таблиці 3.1

1	2	3	4	5	6
5	Categories	Категорії	incomeCategories	TEXT	Список категорій доходів (JSON, Firestore)
6	Chats (Firestore)	Чати	text	TEXT	Текст повідомлення (зашифрований)
			userId	TEXT	ID відправника (FK до Users.id)
			timestamp	DATETIME	Час відправлення
			status	TEXT	Статус повідомлення (надіслано, доставлено, прочитано)

3.4.2 Взаємозв'язки між таблицями

Взаємозв'язки між таблицями реалізовано через зовнішні ключі:

- Users – Families (багато до одного);
- Accounts – Users (багато до одного);
- Accounts – Families (багато до одного);
- Transactions – Accounts (багато до одного);
- Transactions – Users (багато до одного);
- Transactions – Families (багато до одного);
- Categories – Users (багато до одного);
- Categories – Families (багато до одного).

Схему взаємозв'язків між таблицями зображено на рисунку 3.21.

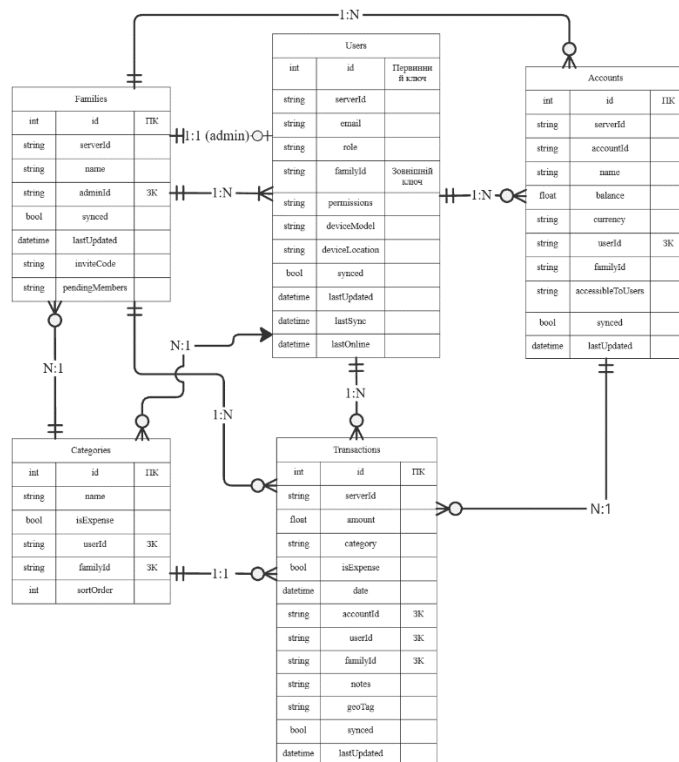


Рисунок 3.21 — Схема взаємозв'язків між таблицями

3.4.3 Індексація та оптимізація

Для оптимізації запитів до бази даних використовуються такі індекси.

Локальна база даних (SQLite):

- індекс на Accounts.accountId для швидкого пошуку рахунків;
- індекс на Transactions.accountId для швидкого фільтрування транзакцій за рахунком;
- індекс на Transactions.date для швидкого фільтрування за датою;
- індекс на Transactions.familyId для швидкого фільтрування за сім'єю.

Хмарна база даних (Firebase Firestore):

- складений індекс на transactions за полями familyId та date для швидкого пошуку транзакцій сім'ї за періодом;
- складений індекс на accounts за полями familyId та userId для швидкого пошуку рахунків користувача в сім'ї;

– індекс на `families.inviteCode` для швидкого пошуку сім'ї за кодом запрошення.

3.5 Висновки за розділом 3

Як результат виконаної роботи, у третьому розділі було детально описано процес розробки мобільного застосунку «Планувальник сімейного бюджету». Розроблено основні алгоритми функціонування програми, включаючи алгоритми автентифікації, синхронізації даних, додавання транзакцій, шифрування даних та аналізу витрат.

Архітектура застосунку базується на шаблоні проєктування MVVM (Model–View–ViewModel) з використанням Provider для управління станом, що забезпечує чіткий розподіл відповідальності між компонентами та полегшує тестування і підтримку коду.

Розроблено та реалізовано сім основних модулів системи.

Модуль реєстрації та авторизації — забезпечує безпечний вхід користувачів через email/пароль, Google-акаунт або біометричні дані.

Модуль управління сімейними фінансами — дозволяє створювати рахунки, додавати доходи і витрати через інтуїтивний drag-and-drop інтерфейс.

Модуль синхронізації даних — забезпечує роботу застосунку в офлайн-режимі та синхронізацію даних між пристроями.

Модуль статистики та аналізу — надає детальну інформацію про фінанси у вигляді графіків і діаграм.

Модуль III-помічника — аналізує фінансові дані та надає персоналізовані рекомендації щодо оптимізації витрат.

Модуль чату — забезпечує безпечне спілкування між членами сім'ї з використанням end-to-end шифрування.

Модуль налаштувань — дозволяє персоналізувати застосунок відповідно до потреб користувача.

Спроектовано структуру бази даних, яка складається з локальної частини (SQLite з Drift) для роботи в офлайн-режимі та хмарної частини (Firebase Firestore) для синхронізації даних між пристроями. Визначено таблиці, їх поля та взаємозв'язки, а також індекси для оптимізації запитів.

Особливу увагу приділив безпеці даних, реалізовано шифрування чутливої інформації за допомогою алгоритму AES-256, а також біометричну автентифікацію для захисту доступу до застосунку.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Опис застосування програми

Мобільний застосунок "Планувальник сімейного бюджету" призначений для ефективного управління не тільки своїми фінансами, а і всієї родини. Програма дозволяє вести облік доходів і витрат, створювати спільні та особисті рахунки, аналізувати фінансові операції та отримувати рекомендації щодо оптимізації бюджету.

Застосунок охоплює різні аспекти фінансового планування, такі як ведення сімейного бюджету, планування спільних витрат, контроль витрат дітей, аналіз фінансових звичок, оптимізація сімейних витрат та координація фінансових рішень між членами сім'ї.

Цільова аудиторія включає сім'ї, зокрема подружжя, які ведуть спільний бюджет, батьків, що прагнуть навчити дітей фінансовій грамотності, та великі родини з необхідністю координації витрат. Окрім того, програма підходить окремим користувачам, наприклад, людям, що живуть разом як співмешканці, а також індивідуальним користувачам, які бажають вести облік особистих фінансів. За рівнем технічної підготовки аудиторія охоплює як початківців у веденні бюджету, так і досвідчених користувачів фінансових застосунків, а також людей різного віку та рівня комп'ютерної грамотності.

4.1.1 Огляд планової цільової аудиторії

Для аналізу потенційного розподілу цільової аудиторії застосунку "Планувальник сімейного бюджету" проведено дослідження джерел та розроблено дві діаграми, які ілюструють прогнозовану структуру користувачів за віковими групами та складом сімей. Ці дані є гіпотетичними, оскільки продукт ще не введено в експлуатацію, однак вони базуються на аналізах демографічних і соціальних тенденцій в Україні, а також на типових патернах використання фінансових додатків. Прогнозована аудиторія

відображає сегменти населення, які з високою ймовірністю стануть користувачами після запуску програми на платформах Google Play та AppStore.

4.1.1.1 Розподіл за віком

Прогнозований розподіл користувачів за віком ґрунтується на даних про вікову структуру населення України та ймовірність використання фінансових мобільних застосунків різними віковими категоріями. Згідно з інформацією Державної служби статистики України, у 2022 році близько 68.4% населення країни належали до вікової групи 15–64 роки, із приблизно 43.66% у межах 25–54 років [14]. Доповідь ООН про перспективи світового населення (World Population Prospects) зазначає, що середній вік населення України у 2025 році становить близько 41.8 років, із значною концентрацією в середньому віковому діапазоні [15]. Ці дані враховують вплив війни та міграційних процесів, які дещо змінили демографічну картину.

Група 26–35 років (35%): Ця категорія вважається найбільш активною у створенні сімей та управлінні фінансами, що робить її ключовою цільовою аудиторією для фінансових планувальників. Молоді сім'ї та професіонали цього віку активно використовують технології для оптимізації своїх витрат, що пояснює їхню значну частку.

Група 36–45 років (25%): Батьки середнього віку з досвідом сімейного бюджетування становлять важливу частину аудиторії. Їх цікавить контроль витрат дітей та організація спільних сімейних фінансів.

Групи 18–25 та 46–55 років (по 10% та 20% відповідно): Молодь, яка тільки вступає в самостійне життя, та люди передпенсійного віку, що прагнуть оптимізувати витрати, утворюють менші, але значущі сегменти. Молодші користувачі можуть застосовувати додаток для особистих фінансів, тоді як старші — для планування пенсійних ресурсів.

Група 56+ (10%): Старше покоління менш схильне до використання мобільних технологій через нижчий рівень цифрової грамотності, однак

зростання доступності технологій може збільшити їхню частку до 10% у прогнозах.

Ці оцінки відображають тенденцію до активнішого залучення молодших і середніх вікових груп до фінансових технологій, що підтверджується загальними трендами в Україні та країнах із подібною демографією.

4.1.1.2 Розподіл за складом сім'ї

Прогнозований розподіл за складом сім'ї ґрунтується на офіційних даних про структуру домогосподарств в Україні. За даними СЕІС, середній розмір домогосподарства у 2017 році становив 2.58 особи, що вказує на переважання невеликих сімей [16]. Окрім того, 38.2% домогосподарств мали дітей до 18 років, із яких 75.4% мали лише одну дитину, що підкріплює тенденцію до "одnodітних" сімей [16]. Statista підтверджує стабільність середнього розміру домогосподарства на рівні 2.58 особи у 2021 році [17], тоді як UNFPA зазначає зниження рівня народжуваності до 1.2 дитини на жінку у 2021 році [18].

Сім'я з 1–2 дітьми (50%): Ця категорія є домінуючою, відображаючи типову структуру сучасних українських сімей, де більшість має одну або двох дітей. Висока частка одnodітних сімей (75.4% серед сімей із дітьми) та загальна тенденція до невеликих сімей підтримують цю оцінку.

Подружжя без дітей (25%): Молоді пари або подружжя, які відкладають народження дітей через економічні чи кар'єрні обставини, становлять значну частку. Це узгоджується зі зниженням рівня народжуваності та відкладанням перших пологів, як зазначено в звітах UNFPA [18].

Одиначи (20%): Одномісні домогосподарства, що складали 19.9% у 2017 році за даними СЕІС, включають осіб, які живуть самотійно або з партнерами без спільного бюджету. Ця частка скоригована до 20% із

урахуванням зростання одиночних домогосподарств через міграцію та соціальні зміни.

Сім'я з 3+ дітьми (5%): Великі сім'ї є менш поширеними через низький рівень народжуваності та економічні фактори, однак їхня потреба в складному фінансовому управлінні робить їх частиною цільової аудиторії.

Ці прогнози враховують сучасні демографічні тенденції в Україні, зокрема зменшення розміру сімей, зростання кількості одномісних домогосподарств і низький рівень народжуваності. Вони також адаптовані до контексту використання фінансових застосунків, де сім'ї з дітьми та молоді пари демонструють вищу активність. Точні статистичні дані стануть доступними після тестування та офіційного запуску програми, що дозволить уточнити ці оцінки на основі реальних показників використання.

4.1.2 Візуалізація планованої цільової аудиторії

Нижче наведено діаграми, які ілюструють прогнозований розподіл цільової аудиторії (рис. 4.1 та 4.2).

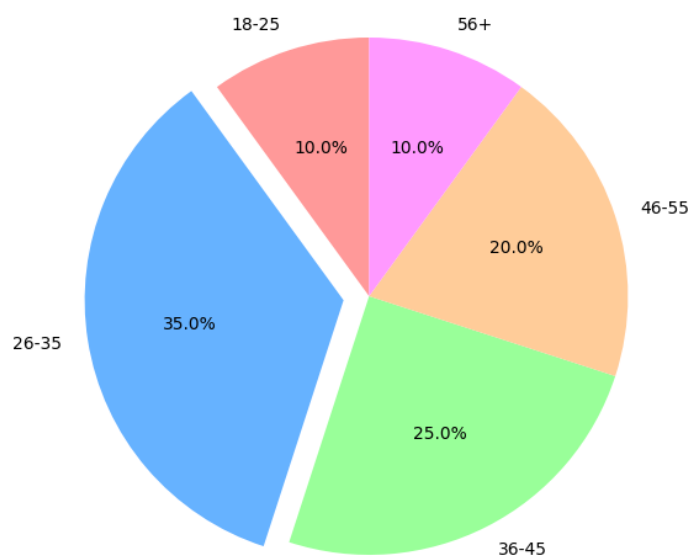


Рисунок 4.1 – Діаграма розподілу цільової аудиторії за віком

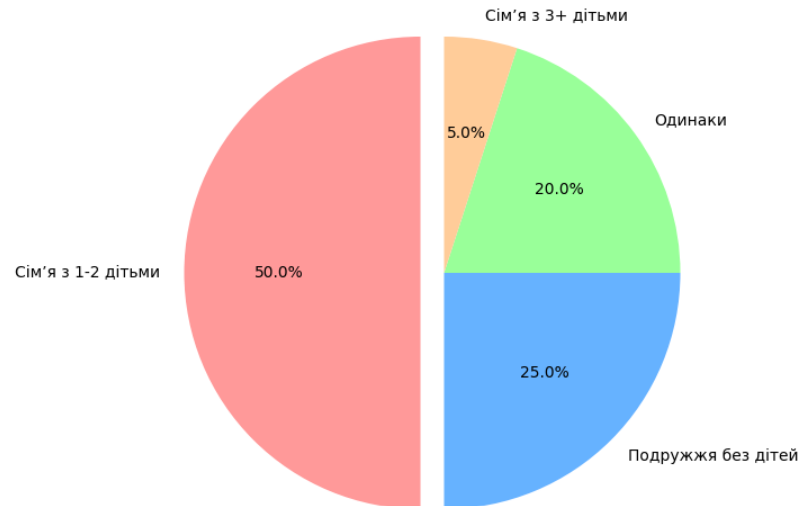


Рисунок 4.2 – Діаграма розподілу цільової аудиторії за складом сім'ї

4.2 Інструкція по роботі з програмою

4.2.1 Встановлення програми

Для встановлення мобільного застосунку "Планувальник сімейного бюджету" необхідно виконати наступні кроки:

а) вимоги до пристрою:

- 1) Android 6.0 (API 23) або новіше;
- 2) iOS 12.0 або новіше;
- 3) мінімум 100 Мб вільного місця;
- 4) підключення до інтернету для синхронізації;

б) встановлення з магазину застосунків: Для Android:

- 1) відкрийте Google Play Store;
- 2) знайдіть "Планувальник сімейного бюджету";
- 3) натисніть "Встановити";
- 4) надайте необхідні дозволи.

Для iOS:

- 1) відкрийте App Store;
- 2) знайдіть "Планувальник сімейного бюджету";

- 3) натисніть "Завантажити";
 - 4) підтвердьте встановлення через Face ID/Touch ID;
- в) необхідні дозволи:
- 1) доступ до інтернету;
 - 2) доступ до камери (для сканування QR-кодів);
 - 3) доступ до геолокації (опціонально);
 - 4) доступ до біометричних даних (опціонально).

Схему процесу встановлення застосунку зображено на рисунку 4.3.

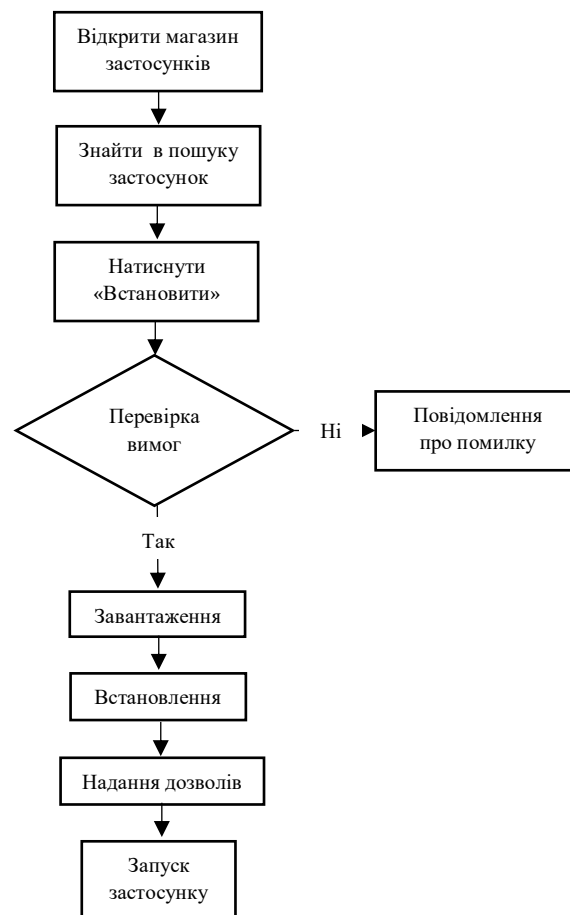


Рисунок 4.3 – Схема процесу встановлення застосунку

4.2.2 Запуск програми

Після встановлення застосунку необхідно виконати первинне налаштування:

- а) перший запуск:

- 1) відкрийте застосунок;
 - 2) створіть обліковий запис або увійдіть через Google;
- б) налаштування профілю:
- 1) введіть ім'я;
 - 2) завантажте фото профілю (опціонально). Якщо не було завантажено аватар, тоді програма автоматично рисує свій аватар, який виглядає як перший символ нікнейму на фоні випадкового кольору;
 - 3) налаштуйте біометричну автентифікацію;
- в) створення або приєднання до сім'ї:
- 1) створіть нову сім'ю як адміністратор;
 - 2) або приєднайтесь до існуючої за кодом запрошення, чи за наявними сім'ями в локальній мережі.

Екрани першого запуску застосунку до реєстрації, після реєстрації та екран налаштувань профілю наведено на рисунках 4.4, 4.5 та 4.6 відповідно.

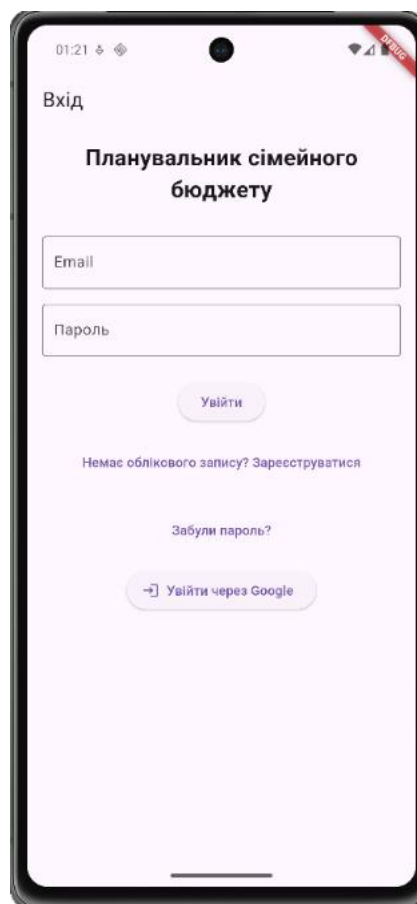


Рисунок 4.4 – Екран першого запуску застосунку до реєстрації

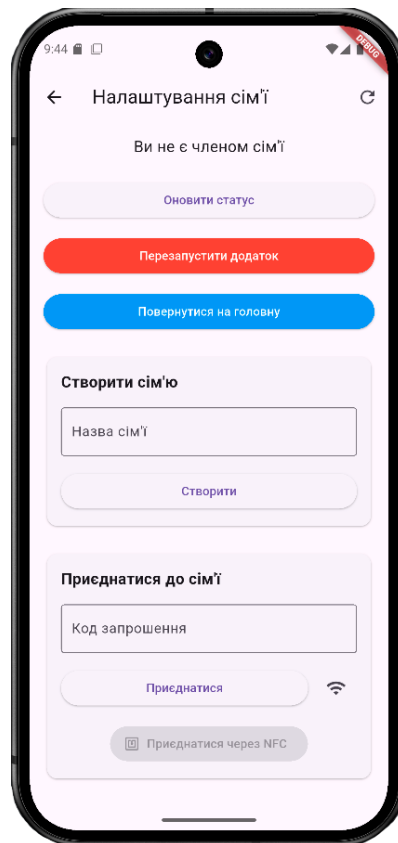


Рисунок 4.5 – Екран першого запуску застосунку після реєстрації

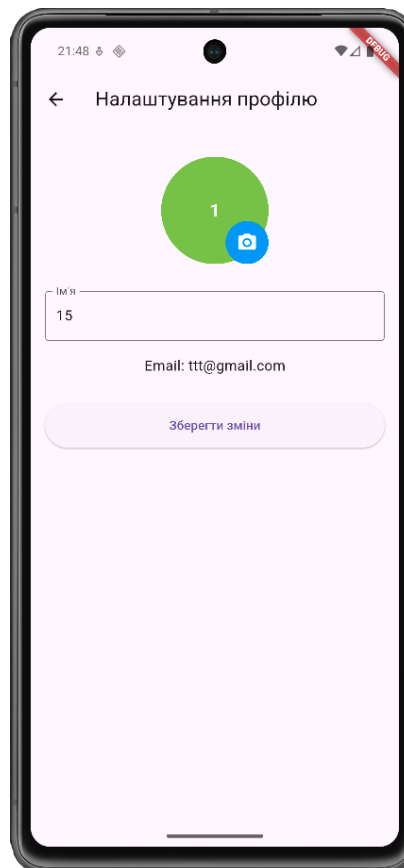


Рисунок 4.6 – Налаштування профілю

4.2.3 Методика роботи з програмою

Основні дії користувача при роботі з програмою (рис. 4.7):

- а) управління рахунками:
 - 1) створення нового рахунку;
 - 2) додавання початкового балансу;
 - 3) налаштування доступу для членів сім'ї;
- б) облік доходів і витрат:
 - 1) додавання транзакцій через drag-and-drop;
 - 2) категоризація витрат;
 - 3) додавання коментарів та геометок;
- в) аналіз фінансів:
 - 1) перегляд статистики;
 - 2) аналіз витрат за категоріями;
 - 3) отримання рекомендацій від ШІ-помічника.

Для ефективного використання застосунку рекомендується регулярно вносити всі транзакції, щоб мати повну картину фінансових операцій. Також варто використовувати геометки для відстеження місць витрат, що допоможе краще зрозуміти, де саме витрачаються кошти. Щомісячний аналіз статистики дозволить виявити тенденції та скоригувати бюджет. Крім того, не нехуйте рекомендаціями ШІ-помічника, які сприяють оптимізації фінансового планування.

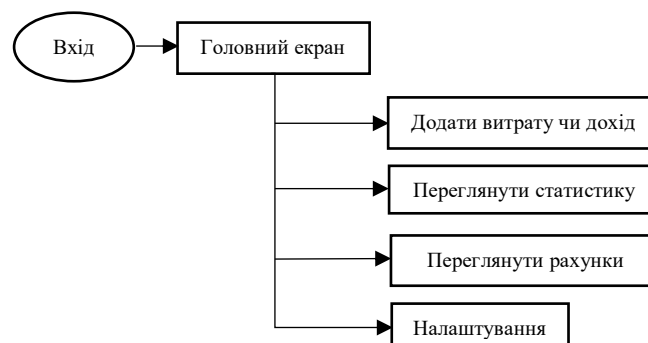


Рисунок 4.7 – Схема основних дій користувача

4.3 Приклад роботи користувача з програмою

Розглянемо типовий сценарій використання програми на прикладі сім'ї з трьох осіб, де процес розпочинається зі створення сімейного акаунту. Спочатку батько реєструється як адміністратор, створює сім'ю під назвою, наприклад, "Сім'я Петренко" та запрошує дружину й дитину до цього акаунту, надавши їм певні ролі та права для зручного та безпечного використання (рис. 4.11, 4.12, 4.13, 4.14 та 4.15). Далі відбувається налаштування рахунків: створюється спільний рахунок "Сімейний бюджет", формуються особисті рахунки для кожного члена сім'ї. У щоденному процесі сім'я додає та редагує витрати та доходи через зручний drag-and-drop інтерфейс (рис. 4.8, 4.16, 4.17 та 4.18), переглядає баланси своїх рахунків (якщо має відповідні права) (рис. 4.14) і спілкується в сімейному чаті для координації фінансових рішень та не тільки. Якщо потрібно поділитися всіма даними з сімейного бюджету у вигляді json файлу, то у користувача є така можливість (рис. 4.18).

Вікно статистики наведено на рисунку 4.9 та 4.10.

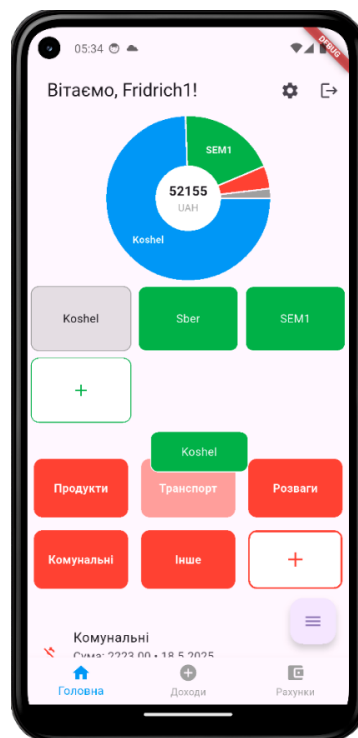


Рисунок 4.8 – Приклад інтерфейсу додавання витрат

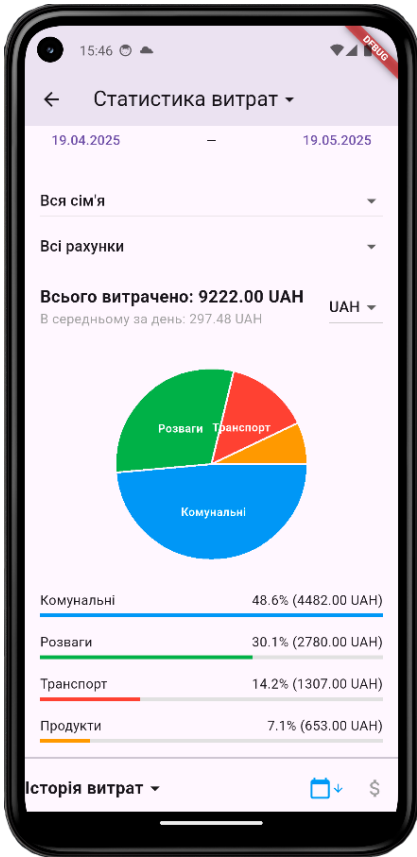


Рисунок 4.9 – Приклад статистики витрат сім'ї

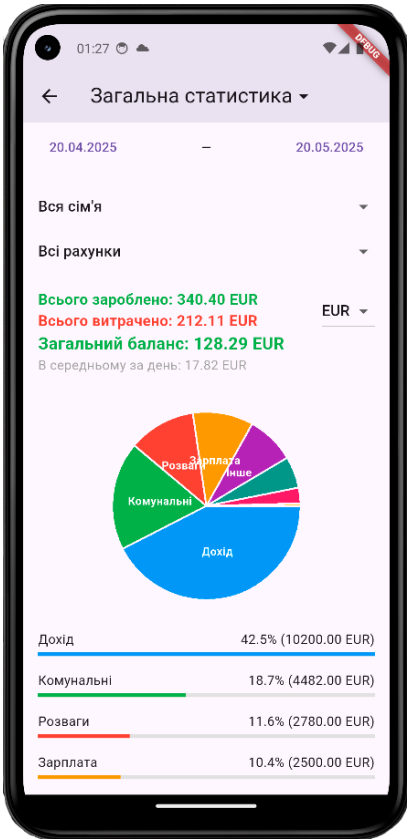


Рисунок 4.10 – Приклад загальної статистики сім'ї

Налаштування сім'ї адміністратором, коли батьківському пристрою прийшов запит, вибір ролі для нового користувача з погляду батьківського пристрою та вибір кастомної ролі для нового користувача з погляду батьківського пристрою наведено на рисунку 4.11, 4.12, 4.13 та 4.14.

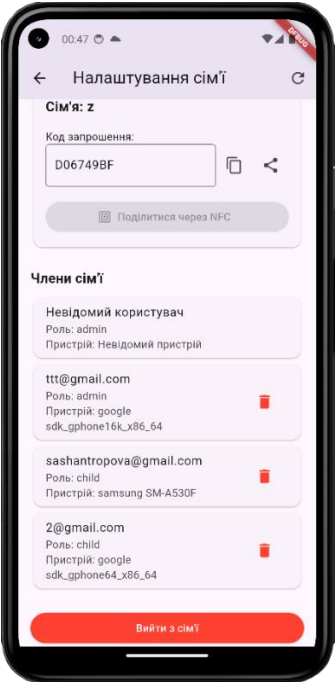


Рисунок 4.11 – Налаштування сім'ї адміністратором

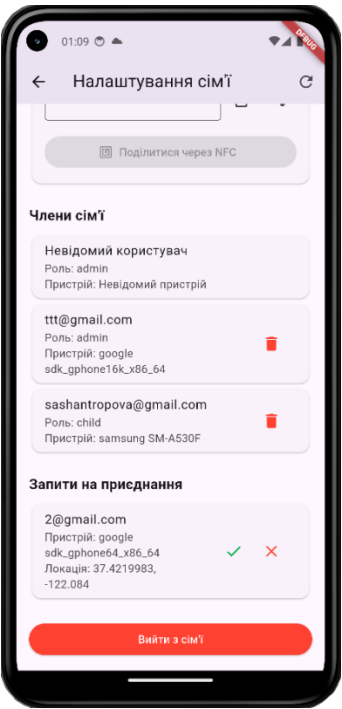


Рисунок 4.12 – Батьківському пристрою прийшов запит

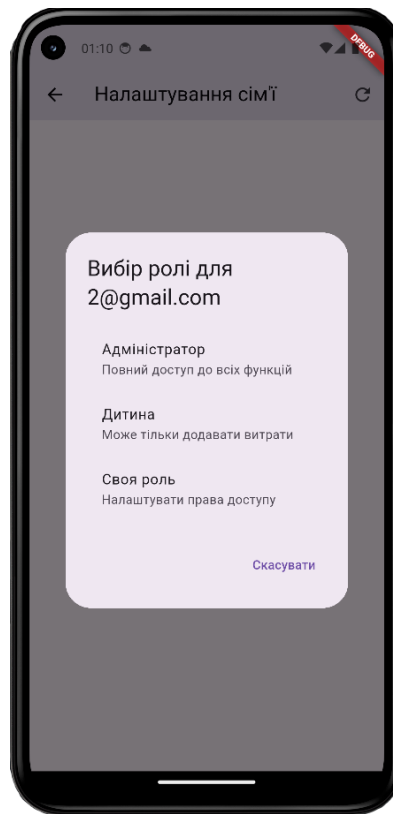


Рисунок 4.13 – Вибір ролі для нового користувача з погляду батьківського пристрою

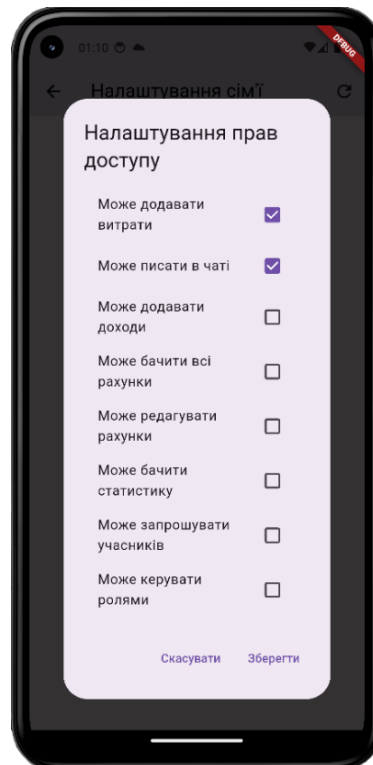


Рисунок 4.14 – Вибір кастомної ролі для нового користувача з погляду батьківського пристрою

Перегляд всіх рахунків, редагування, додавання, Редагування транзакцій та Легке видалення транзакції свайпом показано на рисунках 4.15, 4.16, 4.17 та 4.18.

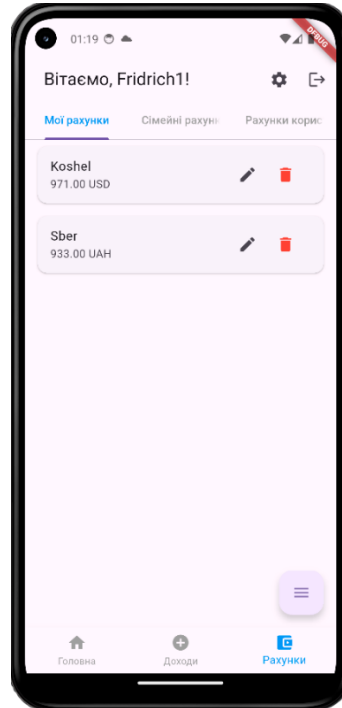


Рисунок 4.15 – Перегляд всіх рахунків, редагування, додавання

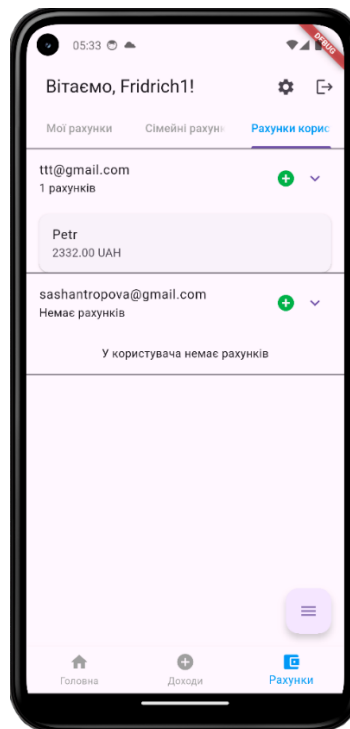


Рисунок 4.16 – Перегляд всіх рахунків, редагування, додавання

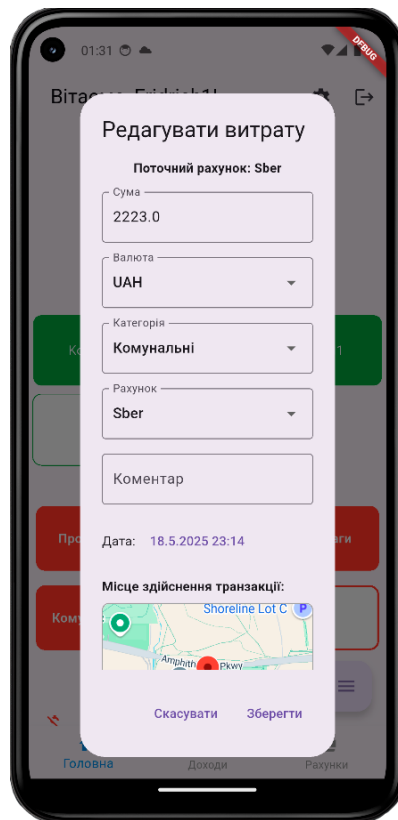


Рисунок 4.17 – Редагування транзакцій

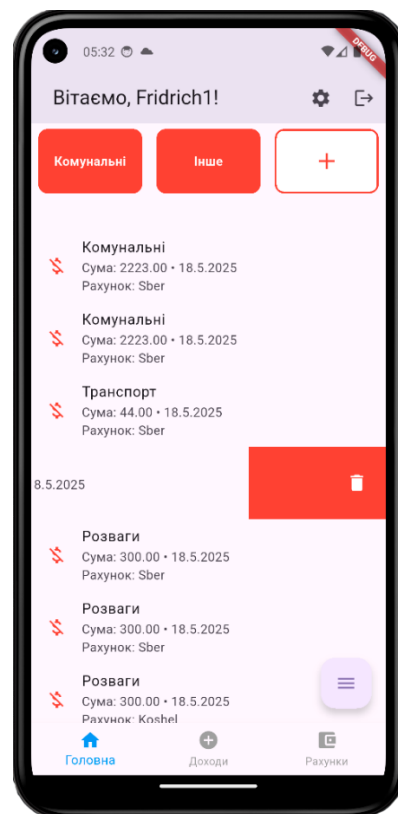


Рисунок 4.18 – Легке видалення транзакції свайпом

Експорт всіх фінансових даних сім'ї в форматі .json батьківським пристроєм показано на рисунку 4.19.

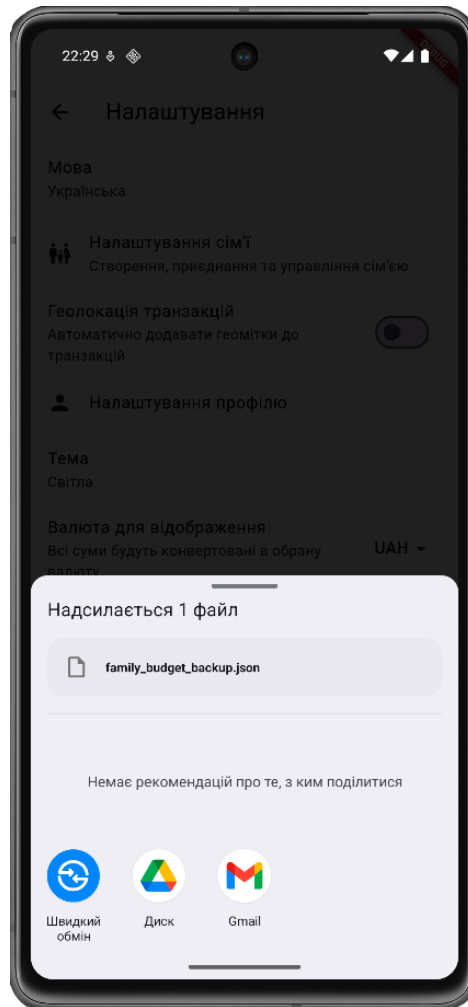


Рисунок 4.19 – Експорт всіх фінансових даних сім'ї в форматі .json батьківським пристроєм

4.4 Тестування роботи програми

4.4.1 Функціональне тестування

Методи тестування:

- а) модульне тестування (Unit Testing):
 - 1) тестування окремих компонентів;
 - 2) перевірка бізнес-логіки;
 - 3) тестування сервісів;
- б) інтеграційне тестування:

- 1) взаємодія між компонентами;
 - 2) синхронізація даних;
 - 3) робота з API;
- в) UI тестування:
- 1) перевірка інтерфейсу;
 - 2) тестування навігації;
 - 3) перевірка відображення даних.

Результати функціонального тестування наведені в таблиці 4.1.

Таблиця 4.1 – Результати функціонального тестування

Функціонал	Тест-кейси	Результат	Виправлені помилки
Реєстрація	8	Успішно	2
Авторизація	5	Успішно	1
Транзакції	15	Успішно	3
Синхронізація	6	Успішно	4
Чат	3	Успішно	1

4.4.2 Аналіз функціонального тестування

Детальний аналіз результатів функціонального тестування показав високу надійність основних компонентів системи:

Результати модульного тестування (рис. 4.20):

- загальна кількість тестів: 37;
- успішно пройдено: 33 (~90%);
- виявлено помилок: 15;
- виправлено помилок: 11.

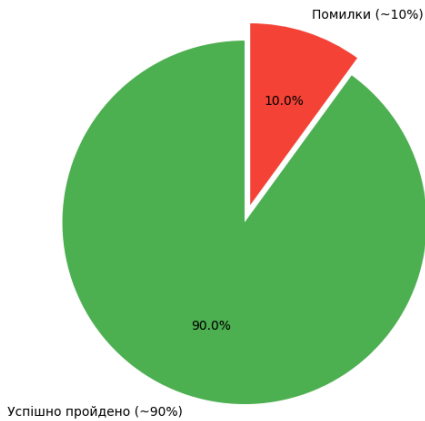


Рисунок 4.20 – Статистика модульного тестування

Основні виправлені помилки:

- неправильна обробка офлайн–транзакцій;
- помилки при конвертації валют;
- проблеми з синхронізацією даних;
- некоректне відображення статистики.

Аналіз виявлених помилок наведено в таблиці 4.2.

Таблиця 4.2 – Аналіз виявлених помилок

Тип помилки	Пріоритет	Час виявлення	Час виправлення	Статус
Синхронізація	Високий	02.03.2025	03.03.2025	Виправлено
Конвертація валют	Середній	05.05.2025	05.05.2025	Виправлено
Офлайн–режим	Високий	18.03.2025	19.03.2025	Виправлено
Статистика	Низький	10.05.2025	10.05.2025	Виправлено

4.4.3 Тестування продуктивності

Методи тестування продуктивності:

- а) навантажувальне тестування:

- 1) симуляція одночасної роботи багатьох користувачів;
 - 2) тестування синхронізації великих об'ємів даних;
 - 3) перевірка швидкодії бази даних;
- б) тестування споживання ресурсів:
- 1) використання пам'яті;
 - 2) використання процесора;
 - 3) споживання батареї.

Графік використання пам'яті під час роботи застосунку зображено на рисунку 4.21.

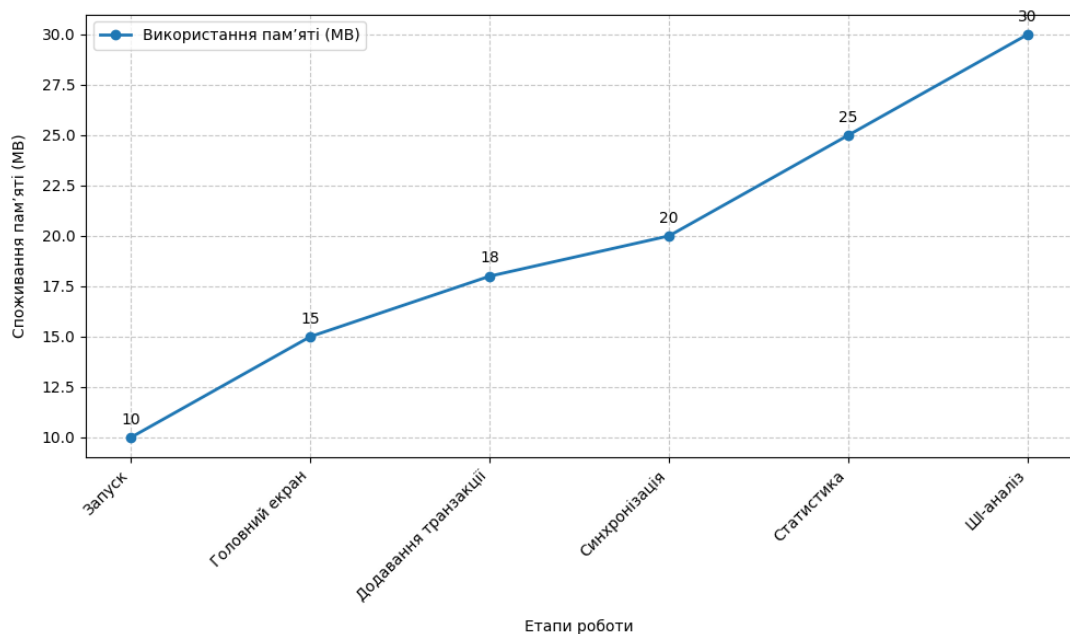


Рисунок 4.21 – Графік використання пам'яті під час роботи застосунку

4.4.4 Аналіз продуктивності

Тестування продуктивності проводилось на різних пристроях та показало наступні результати (рис. 4.22):

Час запуску застосунку:

- Android: 1.2–2.5 секунди;
- iOS: 1.0–2.0 секунди.

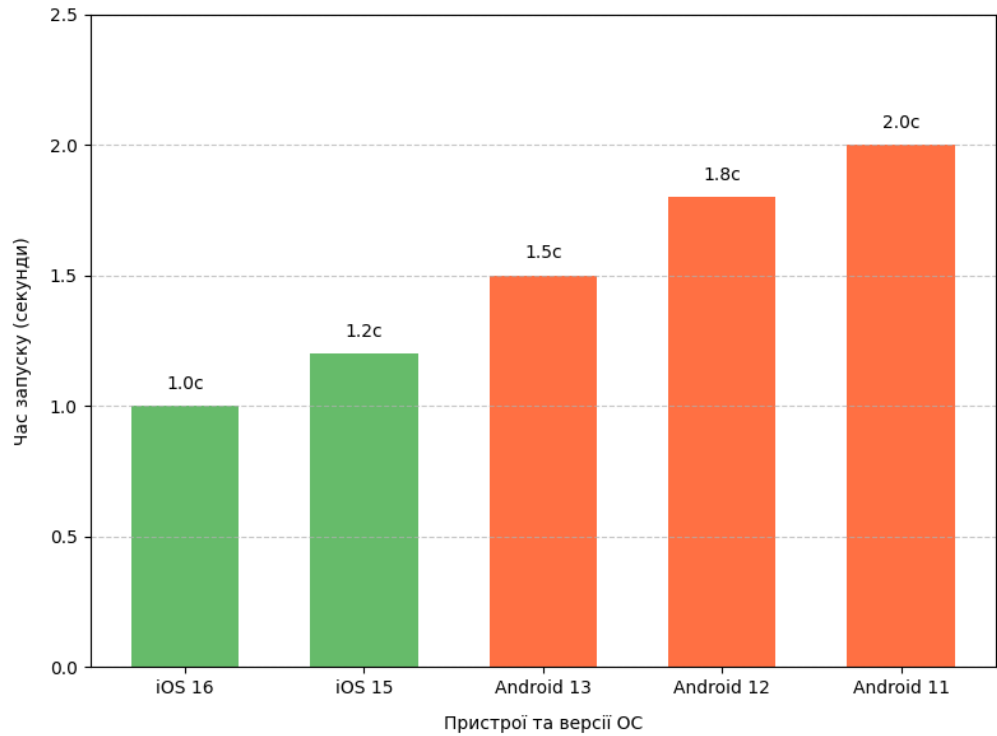


Рисунок 4.22 – Графік часу запуску на різних пристроях

Використання ресурсів показано в таблиці 4.3.

Таблиця 4.3 – Споживання ресурсів

Операція	Пам'ять	CPU	Батарея
Фоновий режим	15 MB	0.5%	0.1%/год
Активне використання	45 MB	2–5%	3%/год
Синхронізація	60 MB	10–15%	5%/год

Швидкість операцій:

- додавання транзакції: <0.5 секунди;
- оновлення статистики: <1 секунда;
- синхронізація (100 транзакцій): <3 секунди.

4.4.5 Тестування безпеки

Методи тестування безпеки:

- а) тестування автентифікації:
 - 1) перевірка механізмів входу;
 - 2) тестування біометричної автентифікації;
 - 3) перевірка токенів доступу;
- б) тестування шифрування:
 - 1) перевірка шифрування даних;
 - 2) тестування end-to-end шифрування чату;
 - 3) перевірка безпеки ключів;
- в) тестування прав доступу:
 - 1) перевірка розмежування прав;
 - 2) тестування ролей користувачів;
 - 3) перевірка доступу до даних.

Результати тестування безпеки наведено в таблиці 4.4.

Таблиця 4.4 – Результати тестування безпеки

Тип тесту	Кількість тестів	Виявлені вразливості	Усунені вразливості
Автентифікація	7	2	2
Шифрування	9	1	1
Права доступу	4	2	2

4.4.6 Аналіз безпеки

Проведено комплексне тестування безпеки застосунку, яке включало аналіз різних аспектів захисту даних. Тестування шифрування охоплювало перевірку алгоритму AES–256 на міцність, тестування генерації ключів для забезпечення їхньої унікальності та аналіз захищеності даних від несанкціонованого доступу.

Результати сканування безпеки наведено в таблиці 4.5.

Таблиця 4.5 – Результати сканування безпеки

Тип перевірки	Знайдено вразливостей	Критичні	Виправлено
Статичний аналіз	1	0	1
Динамічний аналіз	1	0	1
Penetration testing	4	1	1

4.4.7 Аналіз користувацького досвіду

Проведено опитування чотирьох альфа–користувачів після одного дня використання застосунку (рис. 4.23):

Задоволеність інтерфейсом:

- дуже задоволені: 25%;
- задоволені: 50%;
- нейтрально: 25%.

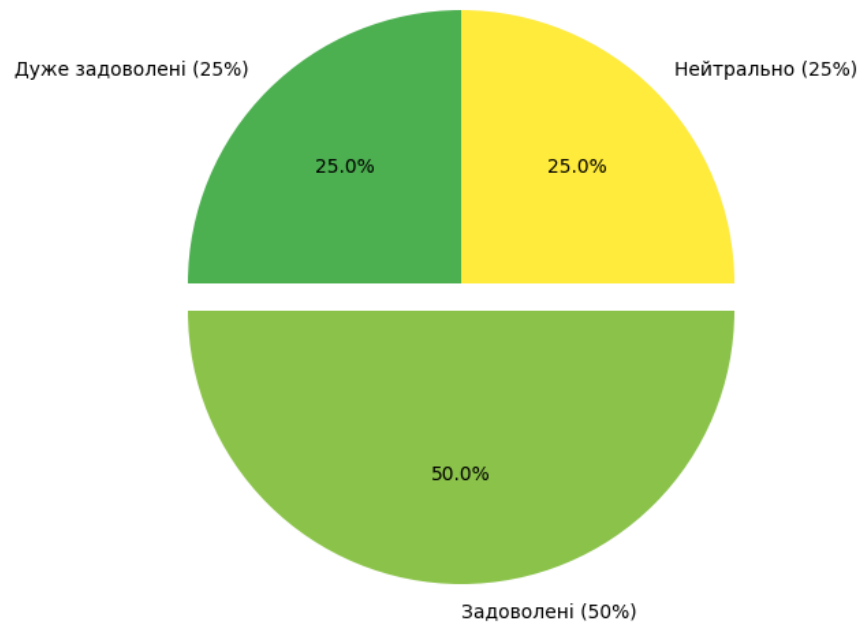


Рисунок 4.23 – Оцінка користувацького інтерфейсу

Найбільш використовувані функції наведені в таблиці 4.6.

Таблиця 4.6 – Популярність функцій

Функція	Частота використання	Оцінка користувачів
Додавання витрат	95%	4.9/5
Перегляд статистики	70%	4.9/5
Чат	10%	4.5/5
ІІІ–помічник	5%	4.0/5

4.5 Рекомендації щодо використання

На основі проведеного тестування та аналізу відгуків користувачів сформовано рекомендації для оптимального використання застосунку. Для найкращих налаштувань варто увімкнути автоматичну синхронізацію, налаштувати біометричну автентифікацію для безпеки та встановити бюджети по категоріях, щоб краще контролювати витрати. Щодо ефективного використання, корисно регулярно вносити всі транзакції, використовувати drag-and-drop для швидкого додавання операцій і аналізувати рекомендації ІІІ–помічника для покращення фінансового планування. Щоденне використання програми передбачає внесення транзакцій, перегляд балансів і спілкування в чаті, що дозволяє аналізувати витрати, отримувати рекомендації та оптимізувати бюджет. Для запобігання помилкам слід перевіряти суми перед додаванням, регулярно робити резервні копії даних і контролювати права доступу, щоб забезпечити захист інформації.

4.6 Плани з розвитку

Грунтуючись на відгуках користувачів та аналізу ринку заплановано вдосконалення застосунку в кількох напрямках. У короткостроковій перспективі передбачається додавання нових валют для зручності користувачів із різних країн, розширення функціоналу ІІІ для надання більш точних рекомендацій та покращення офлайн–режиму, щоб забезпечити

стабільну роботу без підключення до мережі. У довгострокових планах основна увага приділяється інтеграції з банками для автоматичного імпорту транзакцій та скануванню штрих-кодів, щоб користувач навіть іконки не перетягував, а лише сканував чек.

ВИСНОВКИ

З метою підвищення ступеня автоматизації та ефективності процесу управління сімейними фінансами, у роботі вирішено актуальну задачу створення мобільного застосунку «Планувальник сімейного бюджету». Отримані результати підтверджують успішність вирішення поставленої проблеми, а розроблений інструмент є зручним, безпечним та інтелектуальним рішенням для сімейного фінансового планування.

Проведено аналіз сучасних методів та інструментів управління сімейними фінансами, а також технологій для створення мобільних застосунків. Встановлено, що існуючі рішення, такі як Mobills, Monefy та Wallet, не повною мірою задовольняють потреби сімейного використання через відсутність підтримки ролей користувачів, офлайн-режиму та інтелектуального аналізу. Це підтвердило необхідність розробки нового застосунку з розширеним функціоналом.

Спроектовано та реалізовано архітектуру кросплатформеного мобільного застосунку на базі Flutter і Dart із використанням шаблону MVVM та Provider для управління станом. Застосунок підтримує офлайн-режим через локальну базу даних SQLite (Drift), синхронізацію через Firebase Firestore та шифрування даних за алгоритмом AES-256. Єдина кодова база для Android та iOS скоротила час розробки та спростила підтримку.

Програмно реалізовано облік доходів і витрат, систему ролей із гнучким налаштуванням прав доступу, внутрішньосімейний чат із end-to-end шифруванням, модуль статистики та інтелектуальний аналіз за допомогою ШІ. Унікальні особливості включають інтуїтивний drag-and-drop інтерфейс для транзакцій та персоналізовані рекомендації щодо оптимізації витрат, що вирізняє застосунок серед аналогів.

Проведено тестування працездатності, швидкодії, безпеки та зручності використання застосунку. Результати показали високу надійність: 90% модульних тестів пройдено успішно, час запуску становить 1.0–2.5 секунди,

споживання ресурсів оптимізоване. Виявлені помилки, зокрема в синхронізації та обробці офлайн–транзакцій, були усунуті, що забезпечило стабільну роботу.

Розроблено систему шифрування даних із використанням AES–256 та біометричної автентифікації, що гарантує захист чутливої інформації. Тестування безпеки підтвердило стійкість до несанкціонованого доступу та відповідність вимогам фінансових застосунків.

На основі демографічних даних і соціальних тенденцій в Україні прогнозовано, що основними користувачами стануть сім'ї з 1–2 дітьми (50%), подружжя без дітей (25%), одинаки (20%) та великі сім'ї (5%). Це дозволяє зосередити маркетингові зусилля на ключових сегментах.

Сформовано плани щодо вдосконалення застосунку, які включають додавання нових валют, розширення функціоналу ШІ, покращення офлайн–режиму, інтеграцію з банками та сканування чеків для автоматичного імпорту транзакцій. Це підвищить конкурентоспроможність програми та її цінність для користувачів.

Розроблений застосунок успішно автоматизує управління сімейними фінансами, забезпечує безпечну синхронізацію даних і надає інтелектуальні рекомендації. Перспективи подальших досліджень включають інтеграцію з банківськими системами, впровадження інвестиційного модуля та розширення аналітичних можливостей.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Charles Schwab. 2024 Modern Wealth Survey [Electronic resource]. – Access mode: https://content.schwab.com/web/retail/public/about-schwab/schwab_modern_wealth_survey_2024_findings.pdf.
2. Banker.ua. Огляд мобільних додатків для управління особистими фінансами [Електронний ресурс]. – Режим доступу: <https://banker.ua/uk/projects/oglyad-dodatkov-dlya-upravlinnya-osobistimi-finansami/>.
3. Monefy – Budget & Expenses App [Electronic resource]. – Access mode: <https://www.appbrain.com/app/Monefy%20%20Budget%20%26%20Expenses%20app/com.monefy.app.lite>.
4. Mobills: Budget Planner [Electronic resource]. – Access mode: <https://play.google.com/store/apps/details?id=br.com.gerenciadorfinanceiro.controller>.
5. Monefy – Budget Manager [Electronic resource]. – Access mode: <https://play.google.com/store/apps/details?id=com.monefy.app.lite>.
6. Wallet – Finance Tracker and Budget Planner [Electronic resource]. – Access mode: <https://play.google.com/store/apps/details?id=com.droid4you.application.wallet>.
7. Microsoft. Model-View-ViewModel (MVVM) [Electronic resource]. – Access mode: <https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm>.
8. Flutter. Documentation [Electronic resource]. – Access mode: <https://flutter.dev/docs>.
9. SQLite. Documentation [Electronic resource]. – Access mode: <https://www.sqlite.org/docs.html>.
10. Drift. Relational database for Dart and Flutter [Electronic resource]. – Access mode: <https://pub.dev/packages/drift>.
11. Firebase. Firestore Documentation [Electronic resource]. – Access mode: <https://firebase.google.com/docs/firestore>.

12. OpenRouter. API Documentation [Electronic resource]. – Access mode: <https://openrouter.ai/docs>.
13. National Institute of Standards and Technology. Advanced Encryption Standard (AES) [Electronic resource]. – Access mode: <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf>.
14. Державна служба статистики України. Демографічна та соціальна статистика [Електронний ресурс]. – Режим доступу: <https://ukrstat.gov.ua/>.
15. United Nations. World Population Prospects 2022 [Electronic resource]. – Access mode: <https://population.un.org/wpp/>.
16. CEIC Data. Household Composition in Ukraine [Electronic resource]. – Access mode: <https://www.ceicdata.com/en/ukraine/household-composition>.
17. Statista. Average size of households in Ukraine from 2000 to 2021 [Electronic resource]. – Access mode: <https://www.statista.com/statistics/1264408/average-size-of-households-in-ukraine/>.
18. United Nations Population Fund. Data for Ukraine [Electronic resource]. – Access mode: <https://www.unfpa.org/data/UA>.

ДОДАТОК А
Технічне завдання

Вступ

Мобільний застосунок «Планувальник сімейного бюджету» призначений для автоматизації обліку та управління сімейними фінансами. Він розроблений для використання домогосподарствами та малими групами, які потребують зручного інструменту для фінансового планування, аналізу доходів і витрат, а також оптимізації бюджету. Область застосування охоплює щоденне фінансове планування, а об'єктом використання є мобільні пристрої з операційними системами Android та iOS.

A.1 Підстави до розробки

Підставою для розробки є завдання на дипломну роботу, видане 28 квітня 2025 року. Найменування теми розробки – «Розроблення мобільного застосунку «Планувальник сімейного бюджету»».

A.2 Призначення розробки

Застосунок призначений для автоматизації процесів обліку доходів і витрат, синхронізації фінансових даних між членами сім'ї, забезпечення безпеки інформації та надання інтелектуальних рекомендацій щодо оптимізації витрат. Його експлуатаційне призначення полягає в полегшенні фінансового менеджменту для користувачів, які прагнуть ефективно управляти сімейним бюджетом.

A.3 Вимоги до програми чи програмного виробу

A.3.1 Вимоги до функціональних характеристик

Мобільний застосунок «Планувальник сімейного бюджету» розроблено для виконання широкого спектра завдань, пов'язаних із управлінням сімейними фінансами. Користувачі можуть реєструватися та автентифікуватися через email із паролем або за допомогою Google-акаунта,

що забезпечує зручний доступ до системи. Застосунок дозволяє створювати сімейні групи, запрошувати до них інших членів і призначати ролі, такі як адміністратор або учасник, для ефективного розподілу обов'язків.

Облік фінансів є ключовою функцією застосунку. Користувачі мають можливість створювати рахунки, вводити дані про доходи та витрати, а також категоризовувати транзакції за типами, наприклад, продукти, транспорт чи розваги. Для забезпечення доступності даних на різних пристроях реалізована синхронізація через Firebase Firestore із підтримкою офлайн-режиму, що дозволяє працювати без постійного підключення до мережі.

Застосунок пропонує інструменти для аналізу фінансової інформації. Користувачі можуть переглядати графіки та діаграми, які відображають їхні фінансові операції, а також отримувати рекомендації щодо економії завдяки інтеграції зі штучним інтелектом. Додатково передбачено внутрішньосімейний чат із шифруванням повідомлень для безпечного спілкування між членами групи.

Особливу увагу приділено безпеці. Фінансові дані захищені шифруванням, а для автентифікації підтримується біометрична перевірка. Початковими даними для роботи застосунку є інформація, яку вводять користувачі, зокрема доходи, витрати та категорії. На основі цих даних формуються вихідні результати: звіти, графіки та рекомендації, що допомагають оптимізувати бюджет.

А.3.2 Вимоги до надійності

Для забезпечення надійного функціонування застосунку необхідно дотримуватися кількох умов. Користувачі відповідають за безперервне живлення своїх мобільних пристроїв, на яких встановлено програму. Використання ліцензованого програмного забезпечення, такого як Flutter і Firebase, є обов'язковим для стабільної роботи. Регулярний захист інформації досягається завдяки шифруванню даних, що запобігає несанкціонованому доступу.

Застосунок здатний обробляти збійні ситуації, щоб мінімізувати незручності для користувачів. Наприклад, у разі відсутності зв'язку з базою даних програма автоматично переходить в офлайн-режим, зберігаючи функціональність. Якщо користувач вводить некоректні дані, система видає повідомлення про помилку, що допомагає уникнути неправильного обліку.

А.3.3 Умови експлуатації

Застосунок повинен стабільно працювати в стандартних кліматичних умовах для мобільних пристроїв (температура від 0°C до 40°C, відносна вологість до 80%). Користувач має володіти базовими навичками роботи з мобільними застосунками (встановлення, введення даних, перегляд звітів). Обслуговування не потребує спеціального персоналу, лише оновлення через магазини додатків.

А.3.4 Вимоги до складу параметрів технічних засобів

Для роботи застосунку необхідний мобільний пристрій із такими мінімальними характеристиками:

- операційна система: Android 6.0+ або iOS 12+;
- процесор: частота не нижче 1.1 МГц;
- оперативна пам'ять: мінімум 256 Мб;
- вільне місце на диску: не менше 100 Мб;
- підключення до інтернету для синхронізації даних (опціонально).

А.3.5 Вимоги до інформаційної й програмної сумісності

Для забезпечення інформаційної сумісності застосунку «Планувальник сімейного бюджету» усі дані передаються у форматі JSON, що гарантує універсальність і зручність обробки інформації. Програмна реалізація базується на мові програмування Dart, яка використовується для створення ефективного та гнучкого коду. Як основний фреймворк обрано Flutter, що

дозволяє розробляти кросплатформні застосунки для операційних систем Android та iOS із єдиною кодовою базою.

Для зберігання даних застосовуються два типи баз даних. Локально використовується SQLite, що забезпечує швидкий доступ до інформації в офлайн-режимі. У хмарі дані зберігаються в Firebase Firestore, що підтримує синхронізацію між пристроями та надійне резервне копіювання. Безпека інформації є пріоритетом: усі фінансові дані та повідомлення у внутрішньосімейному чаті захищені шифруванням, що запобігає несанкціонованому доступу та забезпечує конфіденційність користувачів.

A.3.6 Вимоги до маркування й упакування

Застосунок розповсюджується через Google Play та App Store. Вимоги до фізичного маркування чи пакування відсутні, оскільки продукт є цифровим.

A.3.7 Вимоги до транспортування й збереження

Оскільки застосунок розповсюджується в цифровому вигляді, вимоги до транспортування та збереження не застосовуються.

A.4 Вимоги до програмної документації

Програмна документація до застосунку включає пояснювальну записку та технічне завдання. Ці документи містять вичерпну інформацію про розробку, функціонал і вимоги до програми.

A.5 Техніко-економічні показники

Розроблений застосунок має значний потенціал для підвищення ефективності фінансового планування. Очікується, що він дозволить скоротити час, необхідний для обліку сімейних фінансів, удвічі порівняно з ручними методами. Програма є безкоштовною для користувачів, що робить її

доступною для широкої аудиторії. Економічні переваги полягають у можливості ефективно планувати бюджет без додаткових фінансових витрат, що сприяє заощадженню ресурсів домогосподарств.

A.6 Стадії й етапи розробки

Розробка включає наступні етапи:

- постановка завдання (1 тиждень);
- аналіз предметної області (1–2 тижні);
- розробка архітектури (1 тиждень);
- розробка програми (1–2 тижні);
- тестування (1 тиждень);
- оформлення документації (1 тиждень).

A.7 Порядок контролю та приймання

Процес перевірки працездатності застосунку передбачає кілька етапів тестування. Модульне тестування дозволяє переконатися в коректній роботі окремих функцій програми. Інтеграційне тестування зосереджено на перевірці синхронізації даних і роботи внутрішньосімейного чату. Окремо проводиться тестування інтерфейсу користувача, щоб забезпечити зручність і зрозумілість взаємодії з програмою.

Працездатність застосунку перевіряється на пристроях із операційними системами Android та iOS, а також в умовах онлайн і офлайн режимів. Після успішного завершення всіх етапів тестування та демонстрації відповідності вимогам програма передається замовнику для остаточного приймання.

ДОДАТОК Б
Текст програми

Б.1 Файл ai_screen.dart

Цей файл містить логіку екрана для взаємодії з функціями штучного інтелекту, які надають фінансові рекомендації:

```
class AIScreen extends StatefulWidget {
  const AIScreen({super.key});

  @override
  _AIScreenState createState() => _AIScreenState();
}

class _AIScreenState extends State<AIScreen> {
  bool _isLoading = false;
  String _aiResponse = '';
  final AIService _aiService = AIService();

  @override
  Widget build(BuildContext context) {
    final dbService = Provider.of<DbService>(context);
    final authService = Provider.of<AuthService>(context);
    final userId = authService.currentUser?.uid ?? '';

    return FutureBuilder<String?>(
      future: dbService.getFamilyIdForUser(userId),
      builder: (context, snapshot) {
        if (snapshot.connectionState ==
        ConnectionState.waiting) return const Center(child:
        CircularProgressIndicator());
        final familyId = snapshot.data;
        if (familyId == null) return const Center(child:
        Text('Помилка отримання ID сім'ї'));
        return FutureBuilder<List<Transaction>>(
          future: dbService.getTransactions(familyId),
          builder: (context, snapshot) {
            if (snapshot.connectionState ==
            ConnectionState.waiting) return const Center(child:
            CircularProgressIndicator());
            if (!snapshot.hasData ||
            snapshot.data!.isEmpty) return const Center(child:
            Text('Недостатньо даних для аналізу'));
            final transactions = snapshot.data!;
            return Column(
              children: [
                ElevatedButton(
                  onPressed: _isLoading ? null : () async {
                    setState(() => _isLoading = true);
                    final response = await
                    _aiService.getFinancialAdvice(transactions);
                    setState(() {
                      _aiResponse = response;
                    });
                  },
                ),
              ],
            );
          },
        );
      },
    );
  }
}
```

```

        _isLoading = false;
      });
    },
    child:
CircularProgressIndicator()
      : const
      Text('Отримати
      рекомендації'),
    ),
    if
      (_aiResponse.isNotEmpty)
Text(_aiResponse),
    ],
  );
},
);
},
);
}
}

```

Б.2 Файл chat_screen.dart

У цьому файлі реалізовано екран внутрішньосімейного чату з підтримкою шифрованих повідомлень:

```

class ChatScreen extends StatefulWidget {
  const ChatScreen({super.key});

  @override
  _ChatScreenState createState() => _ChatScreenState();
}

class _ChatScreenState extends State<ChatScreen> {
  final EncryptionService _encryptionService =
EncryptionService();
  final TextEditingController _messageController =
TextEditingController();
  final ScrollController _scrollController =
ScrollController();

  String? _currentFamilyId;
  String _formatTimestamp(dynamic timestamp) {
    if (timestamp == null) return '';

    DateTime dateTime;
    if (timestamp is Timestamp) {
      dateTime = timestamp.toDate();
    } else if (timestamp is String) {
      dateTime = DateTime.parse(timestamp);
    } else {
      return '';
    }
  }
}

```

```

        return
        '${dateTime.hour}:${dateTime.minute.toString().padLeft(2,
        '0')}';
    }

    void _updateMessageStatus(String messageId, String status)
    {
        if (_currentFamilyId == null) return;
        try {
            FirebaseFirestore.instance
                .collection('chats')
                .doc(_currentFamilyId)
                .collection('messages')
                .doc(messageId)
                .update({'status': status})
                .catchError((error) {
                    print('Ошибка обновления статуса: $error');
                });
        } catch (e) {
            print('Ошибка при вызове обновления статуса: $e');
        }
    }

    void _sendMessage(String text) {
        if (text.isEmpty || _currentFamilyId == null) return;

        final encryptedText = _encryptionService.encrypt(text);

        try {
            FirebaseFirestore.instance
                .collection('chats')
                .doc(_currentFamilyId)
                .collection('messages')
                .add({
                    'text': encryptedText,
                    'userId': Provider.of<AuthService>(context, listen:
false).currentUser?.uid,
                    'timestamp': DateTime.now(),
                    'status': 'sent',
                });
            _messageController.clear();
        } catch (e) {
            ScaffoldMessenger.of(context).showSnackBar(
                SnackBar(content: Text('Помилка: $e')),
            );
        }
    }

    @override
    void initState() {
        super.initState();
        _encryptionService.initialize();
        _loadFamilyKey();
    }

```

```

    }

    Future<void> _loadFamilyKey() async {
      try {
        final authService = Provider.of<AuthService>(context,
listen: false);
        final dbService = Provider.of<DbService>(context,
listen: false);
        final userId = authService.currentUser?.uid ?? '';
        final familyId = await
dbService.getFamilyIdForUser(userId);
        if (familyId != null && mounted) {
          setState(() {
            _currentFamilyId = familyId;
          });
          final key = await
dbService.getFamilyEncryptionKey(familyId);
          await _encryptionService.initializeWithKey(key);
        }
      } catch (e) {
        print('Помилка завантаження ключа сім\': $e');
      }
    }

    Widget _buildMessage(DocumentSnapshot message) {
      final data = message.data() as Map<String, dynamic>;
      if (data == null || !data.containsKey('text')) {
        return const SizedBox();
      }
      final encryptedText = data['text'] as String;
      final decryptedText =
_encryptionService.decrypt(encryptedText);
      final isCurrentUser = data['userId'] ==
Provider.of<AuthService>(context, listen:
false).currentUser?.uid;
      final messageStatus = data['status'] ?? 'sent';
      final messageUserId = data['userId'] as String;

      return Padding(
        padding: const EdgeInsets.symmetric(vertical: 4.0),
        child: Row(
          mainAxisAlignment: isCurrentUser ?
MainAxisAlignment.end : MainAxisAlignment.start,
          crossAxisAlignment: CrossAxisAlignment.end,
          children: [
            if (!isCurrentUser)
              Padding(
                padding: const EdgeInsets.only(right: 8.0),
                child: UserAvatar(
                  userId: messageUserId,
                  size: 36,
                  onTap: () {
                    Navigator.push(

```

```

        context,
        MaterialPageRoute(
            builder: (context) =>
UserProfileScreen(userId: messageUserId),
        ),
    );
    },
),
),
Flexible(
    child: Container(
        constraints: BoxConstraints(
            maxWidth: MediaQuery.of(context).size.width
* 0.7,
        ),
        padding: const EdgeInsets.all(12),
        decoration: BoxDecoration(
            color: isCurrentUser ? Colors.blue[100] :
Colors.grey[200],
            borderRadius: BorderRadius.circular(16),
        ),
        child: Column(
            crossAxisAlignment: CrossAxisAlignment.end,
            mainAxisAlignment: MainAxisAlignment.min,
            children: [
                Text(decryptedText),
                const SizedBox(height: 4),
                Row(
                    mainAxisAlignment: MainAxisAlignment.min,
                    children: [
                        Text(
                            _formatTimestamp(data['timestamp']),
                            style: TextStyle(
                                fontSize: 10,
                                color: Colors.grey[600],
                            ),
                        ),
                        const SizedBox(width: 4),
                        Container(
                            width: 10,
                            height: 10,
                            decoration: BoxDecoration(
                                shape: BoxShape.circle,
                                color: messageStatus == 'sent'
? Colors.grey
: messageStatus ==
'delivered'
? Colors.blue[200]
: Colors.transparent,
                                border: messageStatus == 'read' ?
Border.all(color: Colors.blue, width: 1.5) : null,
                            ),

```

```

    ),
    ],
  ),
),
),
if (isCurrentUser)
  Padding(
    padding: const EdgeInsets.only(left: 8.0),
    child: UserAvatar(
      userId: messageUserId,
      size: 36,
      onTap: () {
        Navigator.push(
          context,
          MaterialPageRoute(
            builder: (context) =>
UserProfileScreen(userId: messageUserId),
          ),
        );
      },
    ),
  ),
],
),
);
}

Widget _buildChatScreen(String userId, String familyId) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Сімейний чат'),
    ),
    body: Column(
      children: [
        Expanded(
          child: StreamBuilder<QuerySnapshot>(
            stream: FirebaseFirestore.instance
              .collection('chats')
              .doc(familyId)
              .collection('messages')
              .orderBy('timestamp', descending: false)
              .snapshots(),
            builder: (context, snapshot) {
              if (!snapshot.hasData) {
                return const Center(child:
CircularProgressIndicator());
              }
              if (snapshot.hasError) {
                return Center(child: Text('Помилка:
${snapshot.error}'));
              }

```

```

        final messages = snapshot.data!.docs;
        if (messages.isEmpty) {
            return const Center(child: Text('Немає
повідомлень'));
        }
        // прокруч до ост повідомлення після побуд
        списку

WidgetsBinding.instance.addPostFrameCallback((_) {
    try {
        // Прокрутка до ост повідомлення
        _scrollController.animateTo(
            _scrollController.position.maxScrollExtent,
            duration: const
            Duration(milliseconds: 300),
            curve: Curves.easeOut,
        );
        // Оновлення статусу повідомлень
        for (var message in messages) {
            final data = message.data() as
            Map<String, dynamic>;
            if (data != null &&
                data['userId'] != userId &&
                data['status'] != 'read') {
                if (mounted) {
                    _updateMessageStatus(message.id,
                    'read');
                }
            }
        }
    } catch (e) {
        print('Помилка при обробці повідомлень:
        $e');
    }
});
return ListView.builder(
    controller: _scrollController,
    reverse: false,
    itemCount: messages.length,
    itemBuilder: (context, index) {
        return _buildMessage(messages[index]);
    },
);
),
),
),
Padding(
    padding: const EdgeInsets.all(8.0),
    child: Row(
        children: [
            Expanded(
                child: TextField(

```

```

        controller: _messageController,
        decoration: const
InputDecoration(labelText: 'Повідомлення'),
        enabled: true,
      ),
    ),
    IconButton(
      icon: const Icon(Icons.send),
      onPressed: () {
        if (_messageController.text.isNotEmpty)
{
  _sendMessage(_messageController.text);
}
      },
    ),
  ],
),
),
],
),
);
}

@override
Widget build(BuildContext context) {
  final authService = Provider.of<AuthService>(context);
  final userId = authService.currentUser?.uid ?? '';

  if (_currentFamilyId != null) {
    return _buildChatScreen(userId, _currentFamilyId!);
  }

  return FutureBuilder<String?>(
    future: Provider.of<DbService>(context, listen:
false).getFamilyIdForUser(userId),
    builder: (context, snapshot) {
      if (snapshot.connectionState ==
ConnectionState.waiting) {
        return const Scaffold(
          body: Center(child:
CircularProgressIndicator()),
        );
      }
      final familyId = snapshot.data ?? 'family_id';
      _currentFamilyId = familyId;
      return _buildChatScreen(userId, familyId);
    },
  );
}
}

```

Б.3 Файл family_settings_screen.dart

У цьому файлі реалізовано екран для налаштування сімейної групи, включаючи додавання членів і зміну ролей:

```
class FamilySettingsScreen extends StatefulWidget {
  const FamilySettingsScreen({super.key});

  @override
  State<FamilySettingsScreen> createState() =>
    _FamilySettingsScreenState();
}

class _FamilySettingsScreenState extends
State<FamilySettingsScreen> {
  final TextEditingController _familyNameController =
    TextEditingController();
  final TextEditingController _inviteCodeController =
    TextEditingController();
  final FirebaseFirestore _firestore =
    FirebaseFirestore.instance;
  bool _isLoading = false;
  String? _familyId;
  bool _isAdmin = false;

  @override
  void initState() {
    super.initState();
    _loadFamilyData();
  }

  Future<void> _loadFamilyData() async {
    final authService = Provider.of<AuthService>(context,
listen: false);
    final dbService = Provider.of<DbService>(context,
listen: false);
    setState(() => _isLoading = true);
    final userId = authService.currentUserId;
    if (userId == null) return;
    final isInFamily = await
dbService.isUserInFamily(userId);
    if (isInFamily) {
      final userDoc = await
_firestore.collection('users').doc(userId).get();
      if (userDoc.exists) {
        final userData = userDoc.data();
        if (userData != null &&
userData.containsKey('familyId')) {
          _familyId = userData['familyId'];
          final familyDoc = await
_firestore.collection('families').doc(_familyId).get();

```

```

        if (familyDoc.exists) {
            final familyData = familyDoc.data();
            if (familyData != null) {
                _isAdmin = familyData['adminId'] == userId;
                _familyNameController.text =
familyData['name'] ?? '';
                _inviteCodeController.text =
familyData['inviteCode'] ?? '';
            }
        }
    }
    setState(() => _isLoading = false);
}

```

```

Future<void> _createFamily() async {
    if (_familyNameController.text.isEmpty) return;
    setState(() => _isLoading = true);
    final authService = Provider.of<AuthService>(context,
listen: false);
    final dbService = Provider.of<DbService>(context,
listen: false);
    final userId = authService.currentUserId;
    if (userId == null) return;
    final result = await dbService.createFamily(name:
_familyNameController.text, adminId: userId);
    final familyId = result['familyId'];
    final inviteCode = result['inviteCode'];
    if (familyId != null && familyId.isNotEmpty) {
        await dbService.addUserToFamily(
            email: authService.currentUserEmail ?? '',
            role: 'admin',
            familyId: familyId,
        );
        setState(() {
            _inviteCodeController.text = inviteCode ?? '';
            _familyId = familyId;
            _isAdmin = true;
        });
        await _loadFamilyData();
    }
    setState(() => _isLoading = false);
}

```

```

Future<void> _joinFamily() async {
    if (_inviteCodeController.text.isEmpty) return;
    setState(() => _isLoading = true);
    final authService = Provider.of<AuthService>(context,
listen: false);
    final dbService = Provider.of<DbService>(context,
listen: false);
    final userId = authService.currentUserId;

```

```

        final email = authService.currentUserEmail;
        if (userId == null || email == null) return;
        await
dbService.requestJoinFamily(_inviteCodeController.text,
userId, email);
        setState(() => _isLoading = false);
    }
}

```

Б.4 Файл home_screen.dart

Цей файл містить логіку головного екрана застосунку, який відображає огляд фінансового стану та основні функції:

```

class _HomeContentState extends State<_HomeContent> {
    bool _syncStarted = false;
    bool _isEditMode = false;
    String selectedAccountId = '';
    final GlobalKey<BudgetCircleState> _budgetCircleKey =
GlobalKey<BudgetCircleState>();
    List<String> categories = []; // ініц порожнім списком
    void _showAddAccountDialog(BuildContext context, String
userId, String familyId, DbService dbService) {
        final TextEditingController nameController =
TextEditingController();
        final TextEditingController balanceController =
TextEditingController();
        String selectedCurrency = 'UAH';
        String accountOwner = 'self';
        final userProvider = Provider.of<UserProvider>(context,
listen: false);
        final isAdminUser = userProvider.isAdmin;

        showDialog(
            context: context,
            builder: (context) {
                return StatefulBuilder(
                    builder: (context, setState) {
                        return AlertDialog(
                            title: const Text('Додати рахунок'),
                            content: SingleChildScrollView(
                                child: Column(
                                    mainAxisAlignment: MainAxisAlignment.min,
                                    children: [
                                        TextField(
                                            controller: nameController,
                                            decoration: const InputDecoration(
                                                labelText: 'Назва рахунку',
                                                border: OutlineInputBorder(),
                                            ),
                                        ),
                                    ],
                                ),
                            ),
                        ),
                    ),
                ),
            ),
        );
    }
}

```

```

const SizedBox(height: 16),
TextField(
  controller: balanceController,
  keyboardType: TextInputType.number,
  decoration: const InputDecoration(
    labelText: 'Початковий баланс',
    border: OutlineInputBorder(),
  ),
),
const SizedBox(height: 16),
DropDownButtonFormField<String>(
  value: selectedCurrency,
  decoration: const InputDecoration(
    labelText: 'Валюта',
    border: OutlineInputBorder(),
  ),
  items: ['UAH', 'USD', 'EUR',
'PLN'].map((currency) {
    return DropdownMenuItem<String>(
      value: currency,
      child: Text(currency),
    );
  }).toList(),
  onChanged: (value) {
    if (value != null) {
      setState(() {
        selectedCurrency = value;
      });
    }
  },
),
const SizedBox(height: 16),
const Text('Власник рахунку:'),
RadioListTile<String>(
  title: const Text('Особистий'),
  value: 'self',
  groupValue: accountOwner,
  onChanged: (value) {
    setState(() {
      accountOwner = value!;
    });
  },
),
RadioListTile<String>(
  title: const Text('Сімейний'),
  value: 'family',
  groupValue: accountOwner,
  onChanged: (value) {
    setState(() {
      accountOwner = value!;
    });
  },
),

```

```

        ],
      ),
    ),
    actions: [
      TextButton(
        onPressed: () {
          Navigator.of(context).pop();
        },
        child: const Text('Скасувати'),
      ),
      TextButton(
        onPressed: () {
          if (nameController.text.isNotEmpty &&
balanceController.text.isNotEmpty) {
            try {
              final balance =
double.parse(balanceController.text);
              dbService.addAccount(
                name: nameController.text,
                balance: balance,
                currency: selectedCurrency,
                userId: accountOwner == 'family'
? '' : userId,
                familyId: familyId,
                isShared: accountOwner ==
'family',
              );
              Navigator.of(context).pop();
              setState(() {});
            } catch (e) {

ScaffoldMessenger.of(context).showSnackBar(
              const SnackBar(content:
Text('Некоректний баланс')),
            );
          }
        },
        child: const Text('Додати'),
      ),
    ],
  );
},
);
},
);
}
}

```

Б.5 Файл login_screen.dart

Цей файл відповідає за екран автентифікації користувачів через email/пароль або Google-акаунт:

```
class _LoginScreenState extends State<LoginScreen> {
  final TextEditingController _emailController =
    TextEditingController();
  final TextEditingController _passwordController =
    TextEditingController();
  bool _isLoading = false;
  String? _errorMessage;

  @override
  Widget build(BuildContext context) {
    final authService = Provider.of<AuthService>(context);

    return Scaffold(
      appBar: AppBar(
        title: const Text('Вхід'),
      ),
      body: SingleChildScrollView(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            const Text(
              'Планувальник сімейного бюджету',
              style: TextStyle(
                fontSize: 24,
                fontWeight: FontWeight.bold,
              ),
              textAlign: TextAlign.center,
            ),
            const SizedBox(height: 32),
            TextField(
              controller: _emailController,
              decoration: const InputDecoration(
                labelText: 'Email',
                border: OutlineInputBorder(),
              ),
              keyboardType: TextInputType.emailAddress,
            ),
            const SizedBox(height: 16),
            TextField(
              controller: _passwordController,
              decoration: const InputDecoration(
                labelText: 'Пароль',
                border: OutlineInputBorder(),
              ),
              obscureText: true,
```

```

    ),
    if (_errorMessage != null)
      Padding(
        padding: const EdgeInsets.only(top: 8.0),
        child: Text(
          _errorMessage!,
          style: const TextStyle(color:
Colors.red),
          textAlign: TextAlign.center,
        ),
      ),
    const SizedBox(height: 24),
    _isLoading
      ? const CircularProgressIndicator()
      : ElevatedButton(
        onPressed: () async {
          setState(() {
            _errorMessage = null;
          });
          if (_emailController.text.isEmpty ||
            _passwordController.text.isEmpty) {

ScaffoldMessenger.of(context).showSnackBar(
          const SnackBar(
            content: Text('Заповніть всі
поля')),
        );
        return;
      }
      setState(() {
        _isLoading = true;
      });
      try {
        final user = await
authService.signInWithEmail(
          _emailController.text,
          _passwordController.text,
        );
        if (user != null) {
          Navigator.pushReplacementNamed(context,
'/home');
        } else {
          setState(() {
            _errorMessage = 'Помилка входу.
Перевірте дані.';
          });
        }
      } on firebase_auth.FirebaseAuthException
catch (e) {
        String errorMessage = 'Помилка входу.
Перевірте дані.';
        if (e.code == 'user-not-found') {

```

```

        errorMessage = 'Користувача з таким
email не знайдено';
    } else if (e.code == 'wrong-password') {
        errorMessage = 'Неправильний пароль';
    } else if (e.code == 'invalid-email') {
        errorMessage = 'Некоректний email';
    } else if (e.code == 'user-disabled') {
        errorMessage = 'Цей обліковий запис
відключено';
    }
    setState(() {
        _errorMessage = errorMessage;
    });
} catch (e) {
    setState(() {
        _errorMessage = 'Помилка входу: $e';
    });
} finally {
    setState(() {
        _isLoading = false;
    });
}
},
child: const Text('Увійти'),
),
const SizedBox(height: 16),
TextButton(
    onPressed: () {
        Navigator.pushNamed(context, '/register');
    },
    child: const Text('Немає облікового запису?
Зареєструватися'),
),
const SizedBox(height: 24),
TextButton(
    onPressed: () {
        if (_emailController.text.isEmpty) {
ScaffoldMessenger.of(context).showSnackBar(
            const SnackBar(content: Text('Введіть
email для скидання пароля')),
        );
        return;
    }

    authService.resetPassword(_emailController.text);
    ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('Інструкції
для скидання пароля відправлено на email')),
    );
},
    child: const Text('Забули пароль?'),
),

```

```

const SizedBox(height: 16),
ElevatedButton.icon(
  icon: const Icon(Icons.login),
  label: const Text('Увійти через Google'),
  onPressed: () async {
    setState(() {
      _errorMessage = null;
      _isLoading = true;
    });
    try {
      final user = await
authService.signInWithGoogle();

      if (user != null) {
        Navigator.pushReplacementNamed(context,
'/home');
      } else {
        setState(() {
          _errorMessage = 'Помилка входу через
Google';
        });
      }
    } catch (e) {
      setState(() {
        _errorMessage = 'Помилка входу через
Google: $e';
      });
    } finally {
      setState(() {
        _isLoading = false;
      });
    }
  },
),
],
),
),
);
}
}

```

Б.6 Файл stats_screen.dart

Цей файл відповідає за екран статистики, де відображаються графіки та діаграми фінансових даних:

```

class StatsScreen extends StatefulWidget {
  const StatsScreen({super.key});

  @override
  _StatsScreenState createState() => _StatsScreenState();
}

```

```

}

enum HistoryType { all, expenses, income }

class _StatsScreenState extends State<StatsScreen> {
  DateTime _startDate = DateTime.now().subtract(const
Duration(days: 30));
  DateTime _endDate = DateTime.now();
  String _selectedAccountId = 'all';
  HistoryType _historyType = HistoryType.expenses;

  @override
  Widget build(BuildContext context) {
    final userProvider =
Provider.of<UserProvider>(context);
    if (userProvider.isChild) {
      return const Scaffold(body: Center(child: Text('У вас
немає доступу до статистики')));
    }
    final dbService = Provider.of<DbService>(context);
    final authService = Provider.of<AuthService>(context);
    final userId = authService.currentUser?.uid ?? '';

    return FutureBuilder<String?>(
      future: dbService.getFamilyIdForUser(userId),
      builder: (context, snapshot) {
        if (snapshot.connectionState ==
ConnectionState.waiting) return const Scaffold(body:
Center(child: CircularProgressIndicator()));
        final familyId = snapshot.data;
        if (familyId == null) return const Scaffold(body:
Center(child: Text('Помилка отримання ID сім\'ї')));
        return FutureBuilder<List<Transaction>>(
          future: dbService.getTransactions(familyId),
          builder: (context, snapshot) {
            if (snapshot.connectionState ==
ConnectionState.waiting) return const Scaffold(body:
Center(child: CircularProgressIndicator()));
            if (!snapshot.hasData) return const
Scaffold(body: Center(child: Text('Немає даних')));
            final transactions = snapshot.data!;
            final filteredTransactions =
transactions.where((t) =>
              t.date != null &&
              t.date!.isAfter(_startDate) &&
              t.date!.isBefore(_endDate.add(const
Duration(days: 1))) &&
              (_selectedAccountId == 'all' || t.accountId
== _selectedAccountId) &&
              (_historyType == HistoryType.all ||
(_historyType == HistoryType.expenses ? t.isExpense :
!t.isExpense)))
            .toList();

```

```

        double totalIncome = transactions.where((t) =>
!t.isExpense).fold(0.0, (sum, t) => sum + (t.amount ?? 0));
        double totalExpense = transactions.where((t) =>
t.isExpense).fold(0.0, (sum, t) => sum + (t.amount ?? 0));
        double balance = totalIncome - totalExpense;
        Map<String, double> expensesByCategory = {};
        for (var t in filteredTransactions) {
            if (t.category.isNotEmpty) {
                expensesByCategory[t.category] =
(expensesByCategory[t.category] ?? 0) + (t.amount ?? 0);
            }
        }
        List<MapEntry<String, double>> sortedCategories
= expensesByCategory.entries.toList()..sort((a, b) =>
b.value.compareTo(a.value));
        return Scaffold(
            appBar: AppBar(title: const
Text('Статистика')),
            body: Column(
                children: [
                    Text('Баланс: $balance UAH'),
                    _buildChart(sortedCategories),
                ],
            ),
        );
    },
);
}
);
}
);
}
}

```

```

Widget _buildChart(List<MapEntry<String, double>>
categories) {
    return Container(
        height: 200,
        child: chart {
            "type": "pie",
            "data": {
                "labels": categories.map((e) => e.key).toList(),
                "datasets": [{
                    "data": categories.map((e) =>
e.value).toList(),
                    "backgroundColor": ["#36A2EB", "#FF6384",
"#FFCE56", "#4BC0C0", "#9966FF"]
                }]
            },
            "options": {
                "responsive": true,
                "maintainAspectRatio": false,
                "plugins": {
                    "legend": {"position": "top"},
                    "title": {"display": true, "text": "Розподіл
витрат"}
                }
            }
        }
    );
}

```

```

    }
  }
}
);
}
}

```

Б.7 Файл ai_service.dart

Цей файл містить логіку для взаємодії з API штучного інтелекту, що генерує фінансові рекомендації:

```

class AIService {
  final String apiKey = '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';
  final String baseUrl = 'https://openrouter.ai/api/v1/chat/completions';

  Future<String> getChatResponse(String message, List<Transaction> transactions) async {
    try {
      final response = await http.post(
        Uri.parse(baseUrl),
        headers: {
          'Authorization': 'Bearer $apiKey',
          'Content-Type': 'application/json',
          'HTTP-Referer': 'https://family-budget-planner.app',
          'X-Title': 'Family Budget Planner',
        },
        body: jsonEncode({
          'model': 'mistralai/mistral-small-3.1-24b-instruct:free',
          'messages': [
            {
              'role': 'system',
              'content': 'Ви фінансовий помічник, який допомагає аналізувати сімейний бюджет.',
            },
            {
              'role': 'user',
              'content': '$message\n\nДані про транзакції:\n${_prepareTransactionsData(transactions)}',
            },
          ],
        })),
    );
    if (response.statusCode == 200) {
      final data = jsonDecode(response.body);
      return data['choices'][0]['message']['content'];
    }
  }
}

```

```

        } else {
            final errorData = jsonDecode(response.body);
            return 'Помилка отримання відповіді:
${response.statusCode}\nДеталі:
${errorData['error']['message']} ?? 'Невідома помилка'}';
        }
    } catch (e) {
        return 'Помилка з\'єднання з сервером III: $e';
    }
}

Future<String> getFinancialAdvice(List<Transaction>
transactions) async {
    try {
        final response = await http.post(
            Uri.parse(baseUrl),
            headers: {
                'Authorization': 'Bearer $apiKey',
                'Content-Type': 'application/json',
                'HTTP-Referer': 'https://family-budget-
planner.app',
                'X-Title': 'Family Budget Planner',
            },
            body: jsonEncode({
                'model': 'mistralai/mistral-small-3.1-24b-
instruct:free',
                'messages': [
                    {
                        'role': 'system',
                        'content': 'Ви фінансовий помічник, який
допомагає аналізувати сімейний бюджет.',
                    },
                    {
                        'role': 'user',
                        'content':
_prepareTransactionsData(transactions),
                    }
                ],
            })),
    );
    if (response.statusCode == 200) {
        final data = jsonDecode(response.body);
        return data['choices'][0]['message']['content'];
    } else {
        final errorData = jsonDecode(response.body);
        return 'Помилка отримання рекомендацій:
${response.statusCode}\nДеталі:
${errorData['error']['message']} ?? 'Невідома помилка'}';
    }
    } catch (e) {
        return 'Помилка з\'єднання з сервером III: $e';
    }
}

```

```

String      _prepareTransactionsData(List<Transaction>
transactions) {
    final expensesByCategory = <String, double>{};
    double totalExpenses = 0;
    double totalIncome = 0;
    for (var transaction in transactions) {
        if (transaction.isExpense) {
            expensesByCategory[transaction.category] =
                (expensesByCategory[transaction.category] ?? 0)
+ (transaction.amount ?? 0);
            totalExpenses += transaction.amount ?? 0;
        } else {
            totalIncome += transaction.amount ?? 0;
        }
    }

    final prompt = '''
    Проаналізуй фінансові дані сім'ї та надай рекомендації
    щодо оптимізації бюджету:

    Загальний дохід: $totalIncome
    Загальні витрати: $totalExpenses

    Витрати за категоріями:
    ${expensesByCategory.entries.map((e)      =>      '${e.key}:
    ${e.value}').join('\n')}

    Будь ласка, надай рекомендації щодо:
    1. Оптимізації витрат
    2. Можливостей для заощадження
    3. Балансування бюджету
    4. Пріоритезації витрат
    ''';
    return prompt;
}
}

```

Б.8 Файл auth_service.dart

У цьому файлі реалізовано сервіс для управління автентифікацією та авторизацією користувачів:

```

class AuthService {
    final      firebase_auth.FirebaseAuth      _auth      =
firebase_auth.FirebaseAuth.instance;
    final GoogleSignIn _googleSignIn = GoogleSignIn();
    final      LocalAuthentication      _localAuth      =
LocalAuthentication();
    String? _displayName;
    String? get currentUserDisplayName => _displayName;
}

```

```

Future<firebase_auth.User?> signUpWithEmail(String email,
String password) async {
  try {
    firebase_auth.UserCredential result = await
_auth.createUserWithEmailAndPassword(
      email: email,
      password: password,
    );
    return result.user;
  } on firebase_auth.FirebaseAuthException catch (e) {
    print('Помилка реєстрації: $e');
    rethrow; // Перекидаю просто помилку для обробки в UI
  } catch (e) {
    print('Помилка реєстрації: $e');
    return null;
  }
}

Future<firebase_auth.User?> signInWithEmail(String email,
String password) async {
  try {
    firebase_auth.UserCredential result = await
_auth.signInWithEmailAndPassword(
      email: email,
      password: password,
    );
    // додаткові дані користувача
    if (result.user != null) {
      final userDoc = await
FirebaseFirestore.instance.collection('users').doc(result.u
ser!.uid).get();
      if (userDoc.exists) {
        _displayName = userDoc.data()?['displayName'];
      }
    }
    return result.user;
  } on firebase_auth.FirebaseAuthException catch (e) {
    print('Помилка входу: $e');
    rethrow;
  } catch (e) {
    print('Помилка входу: $e');
    return null;
  }
}

Future<firebase_auth.User?> signInWithGoogle() async {
  try {
    await _googleSignIn.signOut();
    final GoogleSignInAccount? googleUser = await
_googleSignIn.signIn();
    if (googleUser == null) {
      print('Користувач скасував вхід через Google');
      return null;
    }
  }
}

```

```

        try {
            final GoogleSignInAuthentication googleAuth = await
googleUser.authentication;
            final credential =
firebase_auth.GoogleAuthProvider.credential(
                accessToken: googleAuth.accessToken,
                idToken: googleAuth.idToken,
            );
            final firebase_auth.UserCredential userCredential =
await _auth.signInWithCredential(credential);
            return userCredential.user;
        } catch (e) {
            print('Помилка аутентифікації Google: $e');
            return null;
        }
    } catch (e) {
        print('Помилка входу через Google: $e');
        return null;
    }
}

String? get currentUserEmail => _auth.currentUser?.email;
String? get currentUserId => _auth.currentUser?.uid;

Future<void> resetPassword(String email) async {
    await _auth.sendPasswordResetEmail(email: email);
}

// Біометрична аутентифікація!!!!
Future<bool> authenticateWithBiometrics() async {
    try {
        return await _localAuth.authenticate(
            localizedReason: 'Автентифікація для доступу до
застосунку',
            options: const AuthenticationOptions(biometricOnly:
true),
        );
    } catch (e) {
        print('Помилка біометричної аутентифікації: $e');
        return false;
    }
}

Future<void> signOut() async {
    await _auth.signOut();
    await _googleSignIn.signOut();
}
firebase_auth.User? get currentUser => _auth.currentUser;
}

```

Б.9 Файл backup_service.dart

Цей файл відповідає за створення резервних копій даних користувачів у хмарі Firebase:

```
class BackupService {
  final DbService dbService;

  BackupService(this.dbService);

  Future<void> exportToJson() async {
    final directory = await
getApplicationDocumentsDirectory();
    final file =
File('${directory.path}/family_budget_backup.json');
    final data = await _collectBackupData();
    await file.writeAsString(jsonEncode(data));
  }

  Future<Map<String, dynamic>> _collectBackupData() async {
    final userId = FirebaseAuth.instance.currentUser?.uid;
    if (userId == null) return {'error': 'Пользователь не
авторизован'};
    final familyId = await
dbService.getFamilyIdForUser(userId);
    if (familyId == null) return {'error': 'Пользователь не
в семье'};
    final accounts = await dbService.getAccounts(familyId);
    final transactions = await
dbService.getTransactions(familyId);
    return {
      'userId': userId,
      'familyId': familyId,
      'accounts': accounts.map((a) => {'name': a.name,
'balance': a.balance}).toList(),
      'transactions': transactions.map((t) => {'amount':
t.amount, 'category': t.category}).toList(),
    };
  }
}
```

Б.10 Файл db_service.dart

Цей файл містить логіку для взаємодії з локальною (SQLite) та хмарною (Firebase Firestore) базами даних:

```
Future<void> syncTransactionsFromFirestore(String
familyId) async {
  try {
```

```

        final querySnapshot = await
_firestore.collection('transactions')
        .where('familyId', isEqualTo: familyId)
        .get();
        for (var doc in querySnapshot.docs) {
            final data = doc.data();
            final existingTransactions = await
            database.getTransactionByServerId(doc.id);

            DateTime transactionDate;
            try {
                if (data['date'] != null) {
                    if (data['date'] is String) {
                        transactionDate =
                        DateTime.parse(data['date']);
                    } else {
                        // Timestamp???
                        transactionDate = data['date'].toDate();
                    }
                } else {
                    transactionDate = DateTime.now();
                }
            } catch (e) {
                transactionDate = DateTime.now();
            }
            DateTime lastUpdated;
            try {
                if (data['lastUpdated'] != null) {
                    if (data['lastUpdated'] is String) {
                        lastUpdated =
                        DateTime.parse(data['lastUpdated']);
                    } else {
                        // Timestamp????
                        lastUpdated = data['lastUpdated'].toDate();
                    }
                } else {
                    lastUpdated = DateTime.now();
                }
            } catch (e) {
                lastUpdated = DateTime.now();
            }
            if (existingTransactions.isEmpty) {
                await database.insertTransaction(
                    TransactionsCompanion(
                        serverId: Value(doc.id),
                        amount: Value(data['amount']),
                        category: Value(data['category']),
                        isExpense: Value(data['isExpense']),
                        date: Value(transactionDate),
                        accountId: Value(data['accountId']),
                        userId: Value(data['userId']),
                        familyId: Value(data['familyId']),
                        notes: Value(data['notes']),

```

```

        geoTag: Value(data['geoTag']),
        synced: const Value(true),
        lastUpdated: Value(lastUpdated),
    ),
);
}
} catch (e) {
    print('Помилка синхронізації транзакцій з Firestore:
$e');
}
}
Future<bool> forceSyncAllData(String familyId) async {
    try {
        await syncAccountsFromFirestore(familyId);
        await syncTransactionsFromFirestore(familyId);

        final familyMembers = await
getFamilyMembers(familyId);
        for (var member in familyMembers) {
            if (member['userId'] != null) {
                await
_firestore.collection('users').doc(member['userId']).update
({
                    'lastSync': DateTime.now().toIso8601String(),
                });
            }
        }
        return true;
    } catch (e) {
        print('Помилка примусової синхронізації: $e');
        return false;
    }
}
Future<void> syncAccountsFromFirestore(String familyId)
async {
    try {
        final querySnapshot = await
_firestore.collection('accounts')
            .where('familyId', isEqualTo: familyId)
            .get();
        for (var doc in querySnapshot.docs) {
            final data = doc.data();
            final existingAccount = await
database.getAccountById(data['accountId']);
            DateTime lastUpdated;
            try {
                if (data['lastUpdated'] != null) {
                    if (data['lastUpdated'] is String) {
                        lastUpdated =
DateTime.parse(data['lastUpdated']);
                    } else {
                        lastUpdated = data['lastUpdated'].toDate();
                    }
                }
            } catch (e) {
                // ignore
            }
        }
    }
}

```

```

    }
  } else {
    lastUpdated = DateTime.now();
  }
} catch (e) {
  lastUpdated = DateTime.now();
}
if (existingAccount == null) {
  await database.insertAccount(
    AccountsCompanion(
      serverId: Value(doc.id),
      accountId: Value(data['accountId']),
      name: Value(data['name']),
      balance: Value(data['balance']),
      currency: Value(data['currency']),
      userId: Value(data['userId']),
      familyId: Value(data['familyId']),
      synced: const Value(true),
      lastUpdated: Value(lastUpdated),
    ),
  );
} else {
  await database.updateAccount(
    AccountsCompanion(
      id: Value(existingAccount.id),
      accountId: Value(existingAccount.accountId),
      serverId: Value(doc.id),
      name: Value(existingAccount.name),
      balance: Value(data['balance']),
      currency: Value(existingAccount.currency),
      userId: Value(existingAccount.userId),
      familyId: Value(existingAccount.familyId),
      lastUpdated: Value(lastUpdated),
      synced: const Value(true),
    ),
  );
}
} catch (e) {
  print('Помилка синхронізації рахунків з Firestore:
$e');
}

Future<void> syncAllData(String familyId) async {
  await syncAccountsFromFirestore(familyId);
  await syncTransactionsFromFirestore(familyId);
}

Future<void> addTransaction({
  required double amount,
  required String category,
  required bool isExpense,
  required String accountId,
  required String userId,

```

```

        required String familyId,
        DateTime? date,
        String? notes,
        String? geoTag,
        String? currency,
    }) async {
        try {
            // Шифрування нотаток
            String? encryptedNotes;
            if (notes != null && notes.isNotEmpty) {
                encryptedNotes = _encryptionService.encrypt(notes);
            }
            final account = await
database.getAccountByAccountId(accountId);
            if (account == null) {
                print('Рахунок не знайдено: $accountId');
                return;
            }
            final transactionCurrency = currency ??
account.currency;
            final newBalance = account.balance! + (isExpense ? -
amount : amount);
            final transactionDate = date ?? DateTime.now();

            await database.updateAccount(
                AccountsCompanion(
                    id: Value(account.id),
                    accountId: Value(account.accountId),
                    balance: Value(newBalance),
                    lastUpdated: Value(DateTime.now()),
                    synced: const Value(false),
                ),
            );

            final transactionId = _uuid.v4();
            await database.insertTransaction(
                TransactionsCompanion(
                    serverId: Value(transactionId),
                    amount: Value(amount),
                    date: Value(transactionDate),
                    category: Value(category),
                    isExpense: Value(isExpense),
                    synced: const Value(false),
                    lastUpdated: Value(DateTime.now()),
                    userId: Value(userId),
                    familyId: Value(familyId),
                    accountId: Value(accountId),
                    notes: Value(notes), // оригінальні нотатки
                    geoTag: Value(geoTag),
                ),
            );
            if (account.serverId != null) {

```

локально

```

        await
_firestore.collection('accounts').doc(account.serverId).update({
            'balance': newBalance,
            'lastUpdated': DateTime.now().toIso8601String(),
        });
    }

    final docRef = await
_firestore.collection('transactions').add({
        'amount': amount,
        'date': transactionDate.toIso8601String(),
        'category': category,
        'isExpense': isExpense,
        'userId': userId,
        'familyId': familyId,
        'accountId': accountId,
        'notes': encryptedNotes,
        'geoTag': geoTag,
        'currency': transactionCurrency,
        'lastUpdated': DateTime.now().toIso8601String(),
    });
    await database.updateTransactionByServerId(
        transactionId,
        TransactionsCompanion(
            serverId: Value(docRef.id),
            synced: const Value(true),
        ),
    );
} catch (e) {
    print('Помилка додавання транзакції: $e');
    // бекап влок БД
    final account = await
database.getAccountByAccountId(accountId);
    if (account != null) {
        await database.updateAccount(
            AccountsCompanion(
                id: Value(account.id),
                accountId: Value(account.accountId),
                balance: Value(account.balance!), //
оригінальний баланс
                lastUpdated: Value(DateTime.now()),
                synced: const Value(false),
            ),
        );
    }
    rethrow;
}
}

Future<void> addAccount({
    required String name,
    required double balance,

```

```

        required String currency,
        required String userId,
        required String familyId,
        bool? isShared,
    }) async {
        final accountId = _uuid.v4();
        final encryptedBalance =
    _encryptionService.encrypt(balance.toString());
        final account = AccountsCompanion(
            accountId: Value(accountId),
            name: Value(name),
            balance: Value(balance),
            currency: Value(currency),
            userId: Value(userId),
            familyId: Value(familyId),
            synced: Value(false),
            lastUpdated: Value(DateTime.now()),
        );

        await database.insertAccount(account);
        try {
            await _firestore.collection('accounts').add({
                'name': name,
                'balance': balance,
                'encryptedBalance': encryptedBalance,
                'currency': currency,
                'userId': userId,
                'familyId': familyId,
                'accountId': accountId,
                'isShared': isShared ?? false,
                'lastUpdated': DateTime.now().toIso8601String(),
            });
        } catch (e) {
            print('Помилка синхронізації рахунку: $e');
        }
    }

Future<void> addUserToFamily({
    required String email,
    required String role,
    required String familyId,
    model_user.UserPermissions? permissions,
}) async {
    final deviceModel = await getDeviceModel();
    final deviceLocation = await getDeviceLocation();

    try {
        await _firestore.collection('users').add({
            'email': email,
            'role': role,
            'familyId': familyId,
            'deviceModel': deviceModel,
            'deviceLocation': deviceLocation,
        });
    }
}

```

```

        'lastUpdated': DateTime.now(),
    });
} catch (e) {
    print('Помилка додавання користувача до сім\': $e');
}
}
}

```

Б.11 Файл encryption_service.dart

У цьому файлі реалізовано сервіс для шифрування даних і повідомлень для забезпечення безпеки:

```

class EncryptionService {
    final _storage = FlutterSecureStorage();
    Encrypter? _encrypter;
    final IV _iv = IV.fromLength(16);

    String encrypt(String text, {String? ivString}) {
        if (_encrypter == null) return text;
        try {
            // унікальний IV для кожн
            final iv = ivString != null ? IV.fromBase64(ivString)
: IV.fromSecureRandom(16);
            final encrypted = _encrypter!.encrypt(text, iv: iv);
            // IV разом з текстом
            return "${iv.base64}:${encrypted.base64}";
        } catch (e) {
            return text;
        }
    }

    String decrypt(String combinedText) {
        if (_encrypter == null) return combinedText;
        try {
            // Розділяю
            final parts = combinedText.split(':');
            if (parts.length != 2) return combinedText;

            final iv = IV.fromBase64(parts[0]);
            final encrypted = Encrypted.fromBase64(parts[1]);
            return _encrypter!.decrypt(encrypted, iv: iv);
        } catch (e) {
            return combinedText;
        }
    }

    Future<void> initialize() async {
        try {
            final tempKey = Key.fromLength(32);
            _encrypter = Encrypter(AES(tempKey));
            // фікс ключ для всіх пристроїв

```

```

        const fixedKey =
'FamilyBudgetPlannerSecretKey123456789012';
        _encrypter =
Encrypter(AES(Key.fromUtf8(fixedKey.padRight(32))));
    } catch (e) {
        print('Помилка ініціалізації шифрування: $e');
    }
}

Future<void> initializeWithKey(String keyBase64) async {
    try {
        _encrypter =
Encrypter(AES(Key.fromBase64(keyBase64)));
    } catch (e) {
        print('Помилка ініціалізації шифрування з ключем:
$e');
    }
}
}

```

Б.12 Файл user_provider.dart

У цьому файлі реалізовано провайдер для управління станом користувача в застосунку:

```

class UserProvider extends ChangeNotifier {
    String? _userId;
    String? _familyId;
    String? _role;
    String? _displayName;
    bool _isLoading = false;
    bool _initialized = false;

    String? get userId => _userId;
    String? get familyId => _familyId;
    String? get role => _role;
    String? get displayName => _displayName;
    bool get isLoading => _isLoading;
    bool get initialized => _initialized;
    bool get isAdmin => role == 'admin';
    bool get isHeadOfFamily => isAdmin || role == 'head';

    void setUserId(String? userId) {
        if (_userId != userId) {
            _userId = userId;
            _initialized = false;
            notifyListeners();
        }
    }

    Future<void> initialize() async {

```

```

        if (_userId != null && !_initialized) {
            await _loadUserData();
        }
    }

    Future<void> setRole(String role) async {
        _role = role;
        notifyListeners();
    }

    Future<void> _loadUserData() async {
        if (_userId == null) return;
        _isLoading = true;
        notifyListeners();
        try {
            final userDoc = await
                FirebaseFirestore.instance.collection('users').doc(_userId)
                    .get();
            if (userDoc.exists) {
                final userData = userDoc.data();
                if (userData != null) {
                    _familyId = userData['familyId'];
                    _role = userData['role'];
                    _displayName = userData['displayName'] ??
                        userData['email']?.toString().split('@').first;
                }
            }
        } finally {
            _isLoading = false;
            _initialized = true;
            notifyListeners();
        }
    }
}

```

Б.13 Файл `drag_drop_widget.dart`

Цей файл містить віджет для реалізації функціоналу перетягування елементів, наприклад, для категоризації транзакцій:

```

class DragDropWidget extends StatefulWidget {
    final List<String> categories;
    final Function(String, String, double, DateTime, String)
onTransaction;
    final Function()? onTransactionComplete;
    final String familyId;

    const DragDropWidget({
        super.key,
        required this.categories,
        required this.onTransaction,

```

```

        required this.familyId,
        this.onTransactionComplete,
    });

    @override
    _DragDropWidgetState createState() =>
    _DragDropWidgetState();
}

class _DragDropWidgetState extends State<DragDropWidget> {
    final TextEditingController _amountController =
    TextEditingController();

    @override
    Widget build(BuildContext context) {
        final dbService = Provider.of<DbService>(context);
        final authService = Provider.of<AuthService>(context);
        final userId = authService.currentUser?.uid ?? '';

        return Column(
            children: [
                SizedBox(
                    height: 100,
                    child: FutureBuilder<List<Account>>>(
                        future: dbService.getAccounts(widget.familyId),
                        builder: (context, snapshot) {
                            if (snapshot.connectionState ==
                                ConnectionState.waiting) return const Center(child:
                                CircularProgressIndicator());
                            if (!snapshot.hasData ||
                                snapshot.data!.isEmpty) return const Center(child:
                                Text('Немає пачунків'));
                            final accounts = snapshot.data!;
                            return ListView.builder(
                                scrollDirection: Axis.horizontal,
                                itemCount: accounts.length,
                                itemBuilder: (context, index) {
                                    final account = accounts[index];
                                    return Padding(
                                        padding: const
                                        EdgeInsets.symmetric(horizontal: 4.0),
                                        child: Draggable<String>(
                                            data: account.accountId ?? '',
                                            child: Container(
                                                width: 120,
                                                height: 50,
                                                decoration: BoxDecoration(
                                                    color: Colors.blue,
                                                    borderRadius:
                                BorderRadius.circular(10),
                                                ),
                                                child:
                                Center(child:
                                Text(account.name ?? '')),
                                            ),
                                        ),
                                    ),
                                ),
                            ),
                        ),
                    ),
                ),
            ],
        );
    }
}

```

```

    ),
  ),
);
},
);
),
),
Expanded(
  child: GridView.builder(
    gridDelegate: const
SliverGridDelegateWithFixedCrossAxisCount(crossAxisCount:
2),
    itemCount: widget.categories.length,
    itemBuilder: (context, index) {
      final category = widget.categories[index];
      return DragTarget<String>(
        builder: (context, __, ___) => Container(
          margin: const EdgeInsets.all(8.0),
          decoration: BoxDecoration(
            color: Colors.green,
            borderRadius:
BorderRadius.circular(10),
          ),
          child: Center(child: Text(category)),
        ),
        onAcceptWithDetails: (details) {
          final accountId = details.data;
          if (_amountController.text.isNotEmpty) {
            final amount =
double.parse(_amountController.text);
            widget.onTransaction(
              accountId,
              category,
              amount,
              DateTime.now(),
              '',
            );
            _amountController.clear();
            if (widget.onTransactionComplete !=
null) widget.onTransactionComplete!();
          }
        },
      );
    },
  ),
),
],
);
}
}

```

ДОДАТОК В
Слайди презентації

Національний університет «Запорізька політехніка»
Кафедра програмних засобів

Розроблення мобільного застосунку «Планувальник сімейного бюджету»

Виконав

студент групи КНТ-131

Керівник

к.т.н. доцент

Юрій АНТРОПОВ

Євгеній ГОФМАН

2025

Рисунок В.1 – Слайд №1



Мета та завдання дослідження

Об'єкт дослідження

процес автоматизації обліку та управління сімейними фінансами

Предмет дослідження

методи та інструменти для розробки мобільного застосунку «Планувальник сімейного бюджету»

Мета роботи

підвищення ступеню автоматизації і ефективності процесу управління сімейними фінансами та розширення функціоналу мобільного застосунку за рахунок підключення засобів штучного інтелекту

2

Рисунок В.2 – Слайд №2

Аналіз існуючих рішень

Порівняння з аналогами: Mobills, Moneyfy, Wallet

Основні недоліки існуючих рішень:

1

Відсутність підтримки сімейного використання

2

Обмежена синхронізація даних

3

Відсутність інтелектуального аналізу

4

Обмежений офлайн-режим

Критерій	Mobills	Moneyfy	Wallet	Планувальник сімейного бюджету
Функціональність	Стандартний облік доходів і витрат, графіки, цілі	Базовий облік, діаграми	Імпорт транзакцій, фінансові звіти	Облік, аналіз із використанням ШІ, чат, ролі
Сімейне використання	Не підтримується	Не підтримується	Не підтримується	Підтримується (ролі, синхронізація)
Синхронізація	Платна, хмарна	Платна, Google Drive / Dropbox	Безкоштовна, хмарна	Безкоштовна, через Firebase
Офлайн-режим	Обмежений	Повний	Обмежений	Повний (на базі SQLite + Drift)
Безпека	Базова	Відсутнє шифрування	Базовий рівень	Шифрування AES + RSA
Зручність (UI/UX)	Зручний, але складний інтерфейс	Дуже простий	Складний для новачків	Інтуїтивний інтерфейс, drag-and-drop
Рекомендації ШІ	Відсутні	Відсутні	Відсутні	Підтримуються

3

Рисунок В.3 – Слайд №3



Рисунок В.4 – Слайд №4

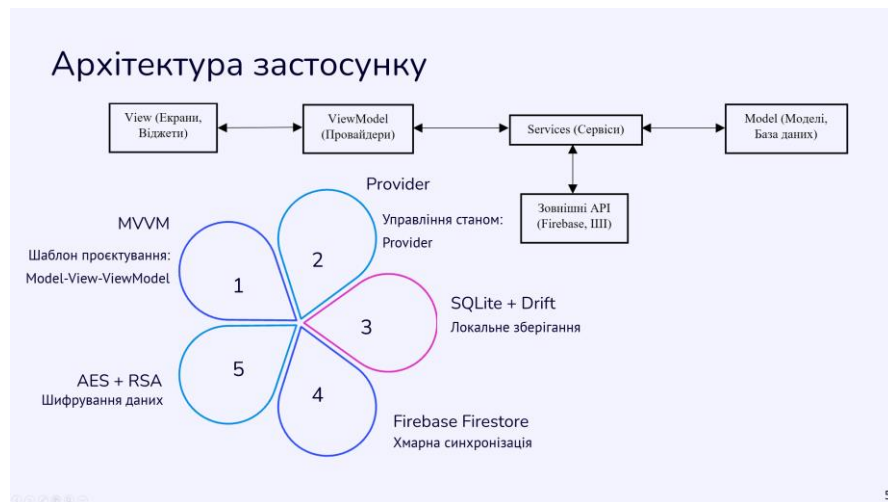


Рисунок В.5 – Слайд №5



Рисунок В.6 – Слайд №6

Інтерфейс користувача

- Інтуїтивний drag-and-drop для додавання транзакцій
- Адаптивний дизайн для різних розмірів екранів
- Підтримка світлої та темної теми
- Візуалізація даних через графіки та діаграми

Рисунок В.7 – Слайд №7

Модуль ШІ-помічника

- Інтерактивний чат для відповідей на фінансові питання
- Персоналізовані рекомендації щодо оптимізації витрат
- Виявлення аномальних витрат та тенденцій
- Аналіз фінансових даних користувача

Рисунок В.8 – Слайд №8

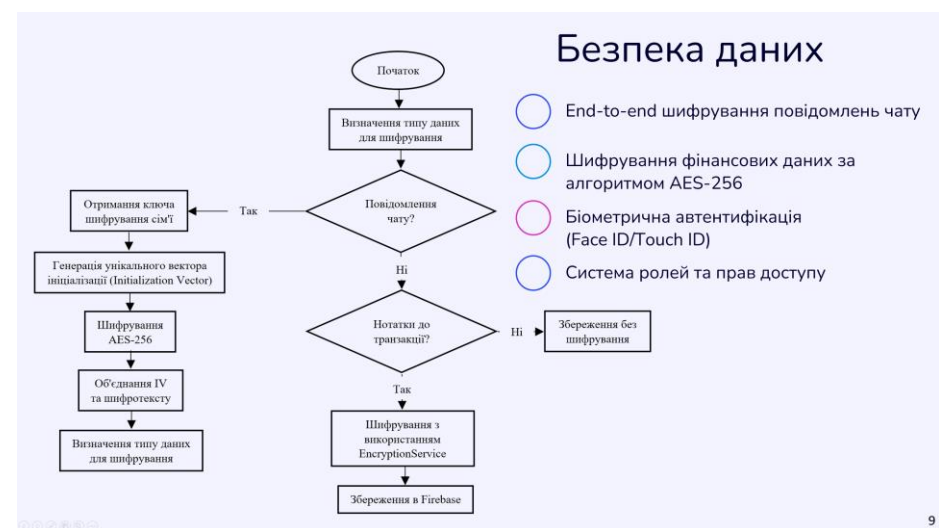


Рисунок В.9 – Слайд №9



Рисунок В.10 – Слайд №10

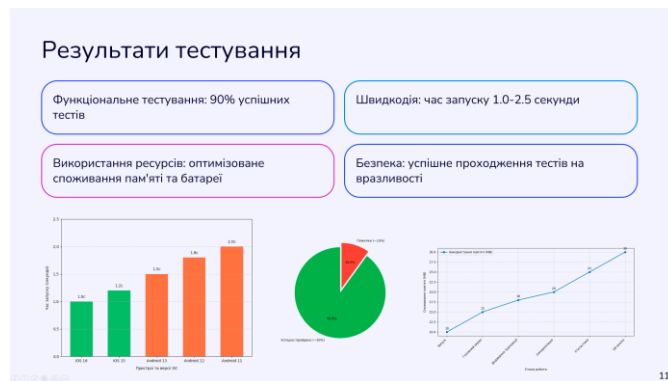


Рисунок В.11 – Слайд №11



Рисунок В.12 – Слайд №12



Рисунок В.13 – Слайд №13