МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: ВИКОРИСТАННЯ МОДЕЛЕЙ УПРАВЛІННЯ ФІНАНСАМИ У МОБІЛЬНОМУ ДОДАТКУ ДЛЯ ОС ANDROID

Здобувача вищої освіти другого (магістерського) рівня,
ІІ курсу, групи КМІз-24-1, спеціальності 122 «Комп'ютерні науки»,
освітня програма «Інформаційні управляючі системи і технології в економіці»


(підпис)

Хачхарджи Івана Олександровича
(прізвище, ім'я, по батькові)

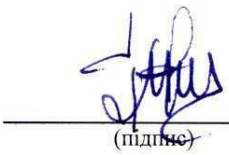
Науковий керівник 
(підпис)

Редич Олександр Володимирович, канд. екон. наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Керівник освітньо-професійної програми 
(підпис)

Омельчук Антон Анатолійович, канд. техн. наук
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту за рішенням кафедри протокол від «09» січня 2026 № 11

Завідувач кафедри 
(підпис)

Філоненко Михайло Миколайович, канд. фіз.-мат. наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Ірпінь – 2026

Ірпінь 2026

Реєстрація № 39 02.02.2026 _____
(дата) (підпис відповідальної особи)

Результати перевірки До захисту 09.01.2026 _____
(до захисту, на доопрацювання) (дата) (підпис)

Результат захисту 4 (добре) 11.02.2026 _____
(оцінка) (дата) (підпис)

Голова комісії: _____ О.С. Дюбанов
(підпис)

Члени комісії:

_____ А.Ю. Горбовий
(підпис)

_____ М.М. Філоненко
(підпис)

_____ І.Д. Погорєловська
(підпис)

БЛАНК ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Об'єкти контролю	Діапазон оцінки в балах	Фактична кількість балів	Примітки	Дата, П.І.Б.: керівника, голови комісії членів комісії,секретаря, (підписи)
ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ				
1. Зміст кваліфікаційної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій)	**від 0 до 30**	20		11.02.2026 Дюбанов О.С.
2. Дотримання вимог щодо оформлення	**від 0 до 10**	9		Горбовий А.Ю.
3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни	**від 0 до 5**	3		Філоненко М.М. Погорєловська І.Д.
4. Рецензент	**від 0 до 5**	3		Суховецька Л.П.
РАЗОМ БАЛІВ	**від 0 до 50**	35		

Оцінювання кваліфікаційної роботи науковим керівником - 38 балів.

ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалоюЄКТС	Оцінка за національною шкалою	Дата	П.І.Б.: голови, членів комісії, секретаря, (підпис)
73	C	4	11.02.2026	Дюбанов О.С. Горбовий А.Ю. Філоненко М.М. Погорєловська І.Д. Суховецька Л.П.

ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою
90-100	A	5 (відмінно)
80-89	B	4 (добре)
70-79	C	
60-69	D	3 (задовільно)
50-59	E	
35-49	FX	2 (незадовільно)

Факультет фінансів та цифрових технологій
Кафедра комп'ютерних та інформаційних технологій і систем
Галузь знань 12 «Інформаційні технології»
Спеціальність 122 «Комп'ютерні науки»
Освітня програма «Інформаційні управляючі системи і технології в економіці»

Затверджую
Завідувач кафедри _____ Філоненко М.М.
«11» грудня 2024

Завдання на кваліфікаційну роботу
здобувачу вищої освіти
Хачхарджи Івану Олександровичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи:
Використання моделей управління фінансами у мобільному додатку для ОС Android
Затверджена наказом ректора від «10» грудня 2024 № 1852
2. Строк здачі закінченої роботи «09» січня 2026
3. Мета роботи: *проектування та програмна реалізація мобільного додатку для операційної системи Android.*
Об'єкт дослідження: *цілісний процес управління особистими фінансами як система операцій з формування, розподілу та використання грошових фондів.*
Предмет дослідження: *сукупність методів, алгоритмів та програмно-технічних засобів, необхідних для реалізації фінансових моделей на сучасних мобільних платформах.*
Орієнтовна база дослідження: *процеси управління фінансами.*
4. Вихідні дані до роботи: *теоретичні моделі бюджетування (50/30/20, ABC-аналіз), математичні методи прогнозування та технологічний стек Flutter/Dart.*
5. Перелік графічного матеріалу: *схеми алгоритмів обробки даних, архітектура UI-компонентів, математичні формули візуалізації та скріншоти інтерфейсу.*
6. Зміст кваліфікаційної роботи (перелік питань за кожним розділом) :
Розділ 1.Теоретичний аналіз предметної області та інструментарію
1.1. Аналіз сучасних моделей та методів управління особистими фінансами
1.2. Огляд ринку мобільних додатків для обліку фінансів на ОС Android
1.3. Обґрунтування вибору платформи Android та технологічного стеку розробки
Розділ 2.Проєктування системи управління фінансами
2.1. Математичне моделювання фінансових алгоритмів додатку
2.2. Проєктування архітектури програмного забезпечення та бази даних
2.3. Розробка користувацького інтерфейсу (UI) та сценаріїв взаємодії (UX)
Розділ 3.Програмна реалізація мобільного додатку;

3.2. Програмування основних функціональних модулів та фінансових моделей

3.3. Створення графічного інтерфейсу та інтеграція візуалізації даних

Розділ 4.Тестування та оцінка ефективності розробки

4.1. Методика та проведення тестування програмного продукту

4.2. Аналіз результатів тестування та усунення виявлених дефектів

4.3. Оцінка ефективності впровадження додатку та інструкція користувача

7. Календарний план виконання роботи

№ п/п	Назва етапів кваліфікаційноїроботи	Строк виконання етапів роботи	
		За планом	Фактично
1	2	3	4
1	Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи	До 15.12.2024	15.12.2024
2	**І розділ** (розкрити сутність і стан дослідження наукової проблематики роботи, після чого навести її загальну характеристику; огляд літератури за темою і вибір конкретних напрямів досліджень)	26.05.2025	26.05.2025
3	**ІІ розділ** (аналіз експериментальної частини про методики досліджень)	02.09.2025	01.09.2025
4	**ІІІ розділ** (проведені теоретичні і (або) експериментальні дослідження та проектні рішення)	24.11.2025	20.11.2025
5	**Висновки**	18.12.2025	18.12.2025
6	Доопрацювання роботи з урахуванням зауважень наукового керівника	02.01.2025	02.01.2025
7	Отримання відгуку керівника	02.02.2026	02.02.2026
8	Отримання зовнішньої рецензії	05.02.2026	05.02.2026

8. Дата видачі завдання «11» грудня 2024

9. Керівник кваліфікаційної роботи _____ (підпис) Редич О.В (прізвище, ініціали)

10. Керівник освітньо-професійної програми _____ (підпис) Омельчук А.А. (прізвище, ініціали)

11. Завдання до виконання отримав здобувач вищої освіти _____ (підпис) Хачхарджи І.О (прізвище, ініціали)

АНОТАЦІЯ

Використання моделей управління фінансами у мобільному додатку для ОС Android

Мета роботи полягає у проєктуванні та програмній реалізації мобільного додатку для операційної системи Android, який інтегрує адаптивні фінансові моделі та алгоритми для планування та аналізу особистого бюджету.

Завданнями роботи є:

1. аналіз предметної області, сучасних моделей управління фінансами та недоліків існуючих аналогів;
2. математичне моделювання фінансових алгоритмів та обґрунтування вибору технологічного стеку;
3. програмна реалізація функціональних модулів додатку «MoneyNote», включаючи механізми візуалізації та локалізації;
4. тестування розробленого комплексу, перевірка точності математичних обчислень та оцінка ефективності впровадження.

Предметом дослідження є сукупність методів, алгоритмів та програмно-технічних засобів, необхідних для реалізації фінансових моделей на сучасних мобільних платформах.

Об'єктом дослідження є цілісний процес управління особистими фінансами, який розглядається як система взаємопов'язаних операцій з формування, розподілу та використання грошових фондів індивіда.

Базою дослідження виступають процеси управління фінансами.

За результатами дослідження сформульовано архітектурні підходи до створення автономного фінансового інструменту. Одержані результати можуть бути використані для підвищення рівня фінансової грамотності користувачів.

Кваліфікаційна робота містить 55 сторінок, 4 рисунки, список використаних джерел з 49 найменувань, 5 додатків.

Ключові слова: особисті фінанси, Android, Flutter, Dart, мобільний додаток, бюджетування, фінансовий облік, архітектура ПЗ, UI/UX

Рік виконання кваліфікаційної роботи 2025

Рік захисту роботи 2026

ABSTRACT

The Use of Financial Management Models in a Mobile Application for Android OS

The aim of the work consists in the design and software implementation of a mobile application for the Android operating system, which integrates adaptive financial models and algorithms for personal budget planning and analysis.

The tasks of the work are:

1. analysis of the subject area, modern financial management models, and shortcomings of existing analogs;
2. mathematical modeling of financial algorithms and justification of the choice of the technology stack;
3. software implementation of the functional modules of the "MoneyNote" application, including visualization and localization mechanisms;
4. testing of the developed complex, verification of the accuracy of mathematical calculations, and assessment of implementation effectiveness.

The subject of the research is the set of methods, algorithms, and software-technical tools necessary for implementing financial models on modern mobile platforms.

The object of the research is the holistic process of personal finance management, considered as a system of interconnected operations regarding the formation, distribution, and use of an individual's monetary funds.

The research base acts as the processes of finance management.

Based on the research results, architectural approaches to creating an autonomous financial tool were formulated. The obtained results can be used to improve the level of financial literacy of users.

The qualification paper contains 55 pages, 4 figures, a list of references with 49 items, and 5 appendices.

Keywords: personal finance, Android, Flutter, Dart, mobile application, budgeting, financial accounting, software architecture, UI/UX, cross-platform development.

Year of qualification paper completion 2025

Year of defense 2026

Зміст

ВСТУП 6

РОЗДІЛ 1. ТЕОРЕТИЧНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІНСТРУМЕНТАРІЮ 10

1.1. Аналіз сучасних моделей та методів управління особистими фінансами. 12

1.2. Огляд ринку мобільних додатків для обліку фінансів на ОС Android. . 14

1.3. Обґрунтування вибору платформи Android та технологічного стеку розробки. 16

РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ УПРАВЛІННЯ ФІНАНСАМИ 18

2.1. Математичне моделювання фінансових алгоритмів додатку. 20

2.2. Проєктування архітектури програмного забезпечення та бази даних. 22

2.3. Розробка користувацького інтерфейсу (UI) та сценаріїв взаємодії (UX). 24

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ДОДАТКУ 26

3.2. Програмування основних функціональних модулів та фінансових моделей. 29

3.3. Створення графічного інтерфейсу та інтеграція візуалізації даних. 33

РОЗДІЛ 4. ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ РОЗРОБКИ 39

4.1. Методика та проведення тестування програмного продукту. 40

4.2. Аналіз результатів тестування та усунення виявлених дефектів. 40

4.3. Оцінка ефективності впровадження додатку та інструкція користувача. 42

ВИСНОВКИ 44

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 46

ВСТУП

Сучасні трансформаційні процеси у світовій економіці, що характеризуються високим рівнем невизначеності, інфляційним тиском та волатильністю фінансових ринків, ставлять перед кожним індивідом нові виклики у сфері забезпечення власного матеріального добробуту. В умовах, коли традиційні моделі заощадження та споживання втрачають свою ефективність через стрімку зміну вартості грошей, питання грамотного управління особистими фінансами виходить на перший план, трансформуючись із бажаної навички у необхідну умову виживання та розвитку домогосподарства. Здатність до системного аналізу власних грошових потоків, планування бюджету та оптимізації витрат стає фундаментом фінансової безпеки, дозволяючи мінімізувати негативний вплив зовнішніх економічних шоків та формувати ресурсну базу для досягнення довгострокових життєвих цілей. Однак, незважаючи на очевидну важливість фінансової дисципліни, значна частина населення стикається з труднощами у веденні регулярного обліку коштів, що часто зумовлено відсутністю зручних інструментів або недостатнім рівнем розуміння принципів бюджетування.

Паралельно з економічними викликами відбувається стрімка цифровізація усіх сфер життя, що докорінно змінює підходи до управління фінансами. Мобільний телефон, перетворившись на універсальний платіжний та комунікаційний термінал, відкриває унікальні можливості для автоматизації рутинних процесів обліку. Ринок фінансових технологій (FinTech) демонструє експоненціальне зростання, пропонуючи користувачам широкий спектр програмних рішень — від простих калькуляторів витрат до складних екосистем банківського обслуговування. Проте детальний аналіз існуючих пропозицій виявляє суттєву прогалину: більшість популярних додатків або перевантажені надлишковим функціоналом, що ускладнює їх освоєння пересічним користувачем, або базуються на моделях монетизації, які передбачають платні підписки чи передачу персональних даних третім особам для таргетованої

реклами. В цьому контексті виникає гостра потреба у створенні доступного, автономного та безпечного інструменту, який би поєднував у собі простоту інтерфейсу з потужними алгоритмами аналізу даних, дозволяючи користувачеві зосередитися на прийнятті рішень, а не на боротьбі зі складністю програмного забезпечення. Саме тому тема розробки спеціалізованого мобільного додатку для управління особистими фінансами є надзвичайно актуальною та своєчасною.

Виходячи з вищезазначеного, метою даної кваліфікаційної роботи є проєктування та програмна реалізація мобільного додатку для операційної системи Android, який інтегрує адаптивні фінансові моделі та алгоритми для автоматизації процесів обліку, планування та аналізу особистого бюджету. Досягнення цієї мети передбачає створення комплексного програмного продукту, здатного забезпечити користувача інструментарієм для точного моніторингу фінансового стану в реальному часі, візуалізації структури витрат та доходів, а також зберігання історії транзакцій у зручному для сприйняття форматі.

Об’єктом дослідження виступає цілісний процес управління особистими фінансами, який розглядається як система взаємопов’язаних операцій з формування, розподілу та використання грошових фондів індивіда. Цей процес охоплює не лише механічну фіксацію транзакцій, але й аналітичну складову, що включає оцінку ефективності витрат, виявлення прихованих резервів бюджету та прогнозування касових розривів. Предметом дослідження є сукупність методів, алгоритмів та програмно-технічних засобів, необхідних для реалізації фінансових моделей на сучасних мобільних платформах. Зокрема, досліджуються особливості побудови архітектури мобільних додатків, алгоритми обробки та агрегації великих масивів числових даних, методи візуалізації статистичної інформації та принципи створення ергономічних користувацьких інтерфейсів у середовищі кросплатформної розробки.

Для досягнення поставленої мети та реалізації науково-практичного потенціалу роботи було сформульовано та вирішено комплекс взаємопов’язаних

задач. На першому етапі здійснено глибокий аналіз предметної області та ринку існуючого програмного забезпечення, що дозволило виявити ключові потреби користувачів та недоліки конкурентних рішень, такі як складність навігації, залежність від інтернет-з’єднання та низький рівень захисту приватності. На основі отриманих даних було проведено вибір та обґрунтування математичних алгоритмів, необхідних для коректного розрахунку балансу, групування транзакцій за категоріями та часовими інтервалами, а також для реалізації механізмів мультивалютності. Наступним кроком стало системне проєктування архітектури додатку, що включало розробку структури даних, визначення логіки взаємодії компонентів та створення прототипів графічного інтерфейсу з урахуванням сучасних вимог User Experience (UX).

Практична реалізація поставлених задач здійснювалася шляхом написання програмного коду з використанням сучасних інструментальних засобів розробки. Основна увага приділялася створенню ключових функціональних модулів: підсистеми обліку транзакцій із забезпеченням валідації вхідних даних, модуля аналітики з інтерактивними діаграмами для візуалізації структури бюджету, а також календарного модуля для навігації по історії операцій. Окремим завданням стала реалізація механізмів локалізації, що забезпечують адаптацію інтерфейсу та даних під мовні та регіональні особливості користувача без втрати цілісності накопиченої інформації. Завершальним етапом роботи стало проведення комплексного тестування розробленого продукту, яке охоплювало модульні тести для перевірки коректності обчислень, інтеграційні тести для оцінки взаємодії компонентів та UI-тестування для верифікації зручності інтерфейсу на різних пристроях.

Наукова новизна роботи полягає у застосуванні комбінованого підходу до побудови архітектури фінансового додатку, що поєднує реактивні патерни управління станом з оптимізованими алгоритмами локальної обробки даних, що дозволило досягти високої продуктивності на мобільних пристроях з

обмеженими ресурсами. Практичне значення одержаних результатів визначається створенням повнофункціонального, готового до впровадження програмного продукту «MoneyNote», який надає користувачам ефективний, безкоштовний та безпечний інструмент для підвищення рівня фінансової грамотності та покращення якості управління особистим бюджетом. Розроблений додаток може слугувати базою для подальших досліджень у сфері персональних фінансів та розширення функціоналу, наприклад, через інтеграцію з банківськими API або впровадження елементів штучного інтелекту для фінансового консультування.

РОЗДІЛ 1. ТЕОРЕТИЧНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІНСТРУМЕНТАРІЮ

Сучасні умови функціонування економічних систем та стрімка діджиталізація розрахунків суттєво змінюють підходи до управління особистими фінансами. Перехід від готівки до цифрових транзакцій знижує психологічний поріг витрачання коштів, що часто призводить до втрати контролю над бюджетом та виникнення касових розривів. У цьому контексті актуалізується потреба у впровадженні інструментів, які дозволяють повернути фінансову дисципліну без необхідності глибоких знань бухгалтерського обліку. Аналіз ринку показує, що існуючі рішення часто не є оптимальними: банківські додатки обмежені екосистемою однієї установи, а спеціалізовані PFM-системи часто перевантажені складним функціоналом, наприклад подвійною бухгалтерією, що є надлишковим для пересічного користувача. Саме тому виникає ніша для створення простого автономного рішення, сфокусованого на миттєвій фіксації транзакцій.

Теоретичним фундаментом розробки виступає модель грошових потоків, де стан бюджету в дискретний момент часу описується як функція від початкового балансу та суми вхідних і вихідних транзакцій. Математично це виражається формулою:

$$B(t) = B(t-1) + \Sigma^{I} - \Sigma^{E} (1.1)$$

Ця модель спирається на принцип атомарності даних, де кожна операція характеризується сумою, напрямком та категорією, що дозволяє застосувати методи структурного групування для візуалізації витрат. Важливим аспектом є робота з часовими рядами: для зручності користувача необхідна нормалізація часу, яка трансформує неперервний потік даних з точними мітками у дискретні календарні періоди, що значно спрощує аналіз динаміки.

Вибір інструментарію реалізації є критичним етапом проектування. Після порівняння нативної розробки та веб-технологій оптимальним було визнано використання кросплатформного фреймворку Flutter. Його архітектура, заснована на власному графічному рушії, дозволяє досягти ідентичності інтерфейсу на різних платформах та високої продуктивності. Використання мови Dart забезпечує надійність коду завдяки суворій типізації та механізму Null Safety, що мінімізує помилки під час виконання, а підтримка асинхронності дозволяє виконувати операції без блокування інтерфейсу. Режими компіляції JIT та AOT пришвидшують як процес написання коду, так і роботу кінцевого додатку.

У якості середовища розробки обрано Visual Studio Code через його легковаговість та модульну архітектуру, що дозволяє ефективно працювати навіть на обладнанні з обмеженими ресурсами. При проектуванні архітектури самого додатку було вирішено уникнути складних патернів управління станом на користь вбудованих реактивних можливостей фреймворку, щоб запобігти надмірному ускладненню коду. Зберігання даних реалізовано через об'єктно-орієнтований підхід в оперативній пам'яті, що забезпечує максимальну швидкість доступу та гнучкість структури даних на етапі розробки. Таким чином, поєднання обраної теоретичної моделі та технологічного стеку дозволяє створити ефективний та надійний програмний продукт для контролю персональних фінансів.

1.1. Аналіз сучасних моделей та методів управління особистими фінансами.

Ефективність програмного засобу для обліку фінансів визначається не лише якістю коду, але й глибиною закладених економічних моделей. Сам по собі облік транзакцій є лише статистичним інструментом фіксації минулого, тоді як повноцінне управління вимагає планування та прогнозування. Для переходу від інтуїтивного розподілу коштів до науково обґрунтованого в архітектуру розроблюваного додатку було інтегровано ряд адаптивних методик: правило пропорційного бюджетування, метод конвертів, ABC-аналіз та математичні методи прогнозування.

Універсальною концепцією стратегічного планування виступає модель пропорційного розподілу бюджету 50/30/20. Вона базується на сегментації чистого доходу на три функціональні групи: 50% спрямовується на покриття базових потреб з низькою еластичністю попиту, 30% - на бажання та покращення якості життя, а 20% - на фінансові цілі та заощадження.

Математично модель описується рівнянням балансу:

$$I = N + W + S \quad (1.2)$$

із дотриманням відповідних нерівностей для кожної компоненти. Програмна реалізація цієї моделі дозволяє системі автоматично аналізувати структуру витрат та сигналізувати користувачеві про диспропорції та відхилення від еталонного розподілу.

Для тактичного управління грошовими потоками більш ефективним є метод конвертів, відомий як бюджетування з нульовою основою. У цифровому середовищі він трансформується у систему віртуальних лімітів, де сума всіх обмежень по категоріях не повинна перевищувати прогнозований дохід.

Психологічний ефект базується на візуалізації зменшення доступного залишку, що стримує імпульсивні покупки. Технічно це вимагає введення поля

«місячний ліміт» у структуру даних та реалізації алгоритму перевірки залишку в реальному часі.

З метою оптимізації витрат доцільно використовувати ABC-аналіз, що базується на принципі Парето. В рамках персональних фінансів це передбачає ранжування транзакцій за ступенем важливості: група А включає критичні витрати (житло, навчання), група В - важливі витрати, що піддаються оптимізації (харчування, одяг), а група С - численні дрібні видатки. Алгоритмічне сортування категорій за питомою вагою дозволяє користувачеві ідентифікувати статті, оптимізація яких дасть максимальний економічний ефект, замість хаотичної економії.

Найбільш перспективним напрямком є впровадження математичних методів прогнозування для попередження касових розривів. Для згладжування випадкових коливань використовується метод простої ковзної середньої (SMA), а для виявлення трендів - лінійна регресія виду

$$Y = a + bX \quad (1.3)$$

де коефіцієнти розраховуються методом найменших квадратів. Це дозволяє екстраполювати поточну динаміку витрат і візуалізувати прогнозований залишок коштів на кінець періоду. Висока продуктивність мови Dart дозволяє виконувати такі обчислення, включаючи коригування на коефіцієнти сезонності, безпосередньо на пристрої користувача. Синтез описаних методик формує комплексну систему, де правило 50/30/20 задає стратегію, метод конвертів забезпечує операційний контроль, а математичне прогнозування гарантує фінансову безпеку.

1.2. Огляд ринку мобільних додатків для обліку фінансів на ОС Android. .

Ринок мобільного програмного забезпечення для фінансового обліку на платформі Android характеризується високою конкуренцією та насиченістю. Попри наявність у Google Play сотень рішень, від простих калькуляторів до банківських екосистем, детальний аналіз лідерів ринку дозволяє виявити ряд концептуальних недоліків, що відкривають нішу для створення нових продуктів. Для формування вимог до власної розробки було проведено порівняння трьох популярних представників сегменту: Monefy, CoinKeeper та ZenMoney, кожен з яких демонструє специфічний підхід до взаємодії з користувачем.

Додаток Monefy тривалий час вважався еталоном завдяки візуальній концепції, заснованій на інтуїтивний круговій діаграмі з іконками категорій. Такий підхід мінімізує час додавання витрат, однак при тривалому використанні стають очевидними архітектурні обмеження. Жорстка прив'язка до структури категорій ускладнює їх кастомізацію, а агресивна модель монетизації блокує базові функції, такі як мультивалютність та розширені звіти. Фактично, Monefy виступає зручним реєстратором минулих подій, але не надає інструментів для гнучкого бюджетування чи прогнозування, що обмежує його корисність як повноцінного фінансового інструменту.

Конкурент CoinKeeper пропонує альтернативну механіку, засновану на гейміфікації, де розподіл коштів відбувається шляхом перетягування віртуальних монет. Хоча на початкових етапах це виглядає інноваційно, з часом процес перетворюється на рутину, що вимагає зайвих моторних дій. Технічний аналіз виявив проблеми з продуктивністю анімацій на бюджетних пристроях, а візуальний шум інтерфейсу ускладнює швидке зчитування балансу. Як і у попередньому випадку, повноцінне використання додатку неможливе без платної підписки, що включає хмарну синхронізацію, суперечачи ідеї безкоштовного автономного інструменту.

Сервіс ZenMoney, що позиціонується як «розумний» помічник, робить ставку на автоматичне розпізнавання банківських SMS та синхронізацію через

API. Попри зручність, це створює суттєві ризики для приватності фінансових даних та робить систему залежною від технічних протоколів банків. Будь-яка зміна формату повідомлень може призвести до некоректного обліку, а вимога постійного підключення до мережі Інтернет унеможливлює використання додатку в умовах відсутності зв'язку. Така надмірна автоматизація часто стає недоліком для користувачів, які прагнуть повного контролю над власними даними.

Узагальнення аналізу дозволяє виділити спільні проблеми існуючих рішень: штучне обмеження функціоналу у безкоштовних версіях, перевантаженість інтерфейсів зайвими опціями та ризики хмарного зберігання даних. Саме ці недоліки стали підґрунтям для концепції проекту «MoneyNote», який розробляється як антипод комерційним продуктам. На противагу конкурентам, новий додаток використовує чистий Material Design замість ресурсомісткої гейміфікації та сповідує принцип «Offline First». Усі дані зберігаються локально на пристрої, що гарантує конфіденційність, а архітектура на базі Flutter забезпечує стабільну роботу та ідентичність інтерфейсу незалежно від версії Android.

1.3. Обґрунтування вибору платформи Android та технологічного стеку розробки.

Процес проектування програмного забезпечення вимагає ретельного обґрунтування вибору платформи та інструментів, що впливає на продуктивність та масштабування продукту. Для додатку «MoneyNote» пріоритетною обрано операційну систему Android, яка займає провідні позиції на світовому ринку та покриває широкий спектр пристроїв, від бюджетних до флагманських [8]. Крім того, відкритість файлової системи Android спрощує дистрибуцію та тестування через APK-файли, що є вагомою перевагою на етапі розробки.

При виборі технологічного стеку було проведено порівняння нативних та кросплатформних підходів. Хоча використання мови Kotlin є стандартом нативної розробки, цей шлях є економічно затратним через необхідність окремого написання коду для iOS [4]. Враховуючи обмежені часові рамки, було прийнято рішення використовувати кросплатформний фреймворк Flutter, який дозволяє створювати конкурентоспроможний продукт для обох платформ з єдиною кодовою базою.

Вибір Flutter та мови Dart зумовлений їхньою архітектурою, яка нівелює недоліки традиційних кросплатформних рішень. Завдяки власному графічному рушію Skia, Flutter не залежить від OEM-віджетів, забезпечуючи ідентичність інтерфейсу на будь-якому пристрої. Компіляція Dart у нативний код (AOT) гарантує високу продуктивність, а функція Hot Reload (JIT) значно пришвидшує процес написання коду [6].

Архітектурне рішення проекту адаптовано під специфіку фреймворку. Замість класичного для Android патерну MVVM, застосовано реактивний підхід з використанням віджетів зі станом (StatefulWidget), де інтерфейс є функцією від стану. Це дозволило зберегти чистоту архітектури (Clean Architecture) та чітко розділити бізнес-логіку та візуалізацію без зайвих абстракцій [34].

Організація збереження даних реалізована через патерн Repository, що забезпечує уніфікований інтерфейс для CRUD-операцій. На поточному етапі використовується об'єктна модель у пам'яті, проте структура проекту дозволяє безболісно інтегрувати повноцінні локальні бази даних, такі як SQLite або Hive, у майбутньому [17].

В якості інтегрованого середовища розробки обрано Visual Studio Code. У порівнянні з Android Studio, це середовище є менш вимогливим до ресурсів системи та дозволяє гнучко налаштувати робочий процес за допомогою розширень [12]. Такий вибір інструментарію дозволив створити ефективну екосистему для успішної реалізації завдань дипломної роботи.

РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ УПРАВЛІННЯ ФІНАНСАМИ

Етап технічного проектування визначає архітектурний базис програмного забезпечення та формує інженерні специфікації. Для розробки додатку «MoneyNote» обрано стратегію ітеративного проектування, що дозволило адаптувати вимоги під специфіку кросплатформного фреймворку Flutter. Процес включав моделювання структур даних, розробку алгоритмів та проектування ергономічного інтерфейсу [1].

В основу архітектури покладено принцип реактивного програмування, де інтерфейс є функцією від стану системи. Замість складних патернів, таких як BLoC, використано модель підйому стану (Lifting State Up). Бізнес-логіка та дані сконцентровані у кореневих віджетах, які передають інформацію дочірнім елементам через конструктори, а отримують події через функції зворотного виклику. Це забезпечує необхідний рівень інкапсуляції для автономного додатку та спрощує підтримку коду [15].

Ключовим елементом стала розробка моделі даних. Центральна сутність Transaction забезпечує атомарність операцій та містить суму, тип (дохід/витрата), дату та категорію. Для зберігання сум обрано тип даних з плаваючою комою (double), що спрощує математичні операції на клієнтській стороні. Категорії спроектовано як об’єкти з візуальними ідентифікаторами (іконка, колір) для побудови графіків.

Організація зберігання даних реалізована за гібридним підходом. Активні дані під час сесії утримуються в оперативній пам'яті для миттєвого доступу, фільтрації та сортування. Клас-менеджер реалізує патерн «Єдине джерело істини» (Single Source of Truth), інкапсулюючи методи модифікації даних та автоматичного перерахунку балансу [29].

Для аналітичного модуля розроблено алгоритм візуалізації витрат на круговій діаграмі. Для точного розміщення іконок поверх секторів використано тригонометричну модель. Координати (x, y) для кожного елемента розраховуються на основі кута α

та радіуса R за формулами:

$$x = R \cdot cos\ cos\ (\alpha)$$

$$y = R \cdot sin\ sin\ (\alpha) \quad (2.1)$$

Цей алгоритм забезпечує адаптивність верстки незалежно від розміру екрану пристрою [40].

Проектування календарного модуля вирішило задачу агрегації транзакцій. Використано структуру хеш-мапи (Hash Map), де ключем є дата, а значенням — список операцій. Це змінило складність пошуку з лінійної на константну O(1). Для візуалізації календаря генерується матриця 7x6, що дозволяє коректно відображати будь-який місяць разом із перехідними днями сусідніх періодів [28].При проектуванні UI/UX пріоритетом визначено мінімізацію кліків. Розроблено кастомну панель навігації (Bottom App Bar) з кнопками швидкого додавання доходів та витрат, що зручно для використання однією рукою. Впроваджено кольорове кодування (червоний/зелений) для зниження когнітивного навантаження [24].

Система локалізації реалізована через перерахування (Enums) замість зовнішніх JSON-файлів, що забезпечує миттєве перемикання мови. Для збереження цілісності історії при зміні мови розроблено алгоритм міграції, який використовує іконки категорій як стабільні ідентифікатори для оновлення текстових назв. Безпека даних забезпечується валідацією на рівні полів введення та системою підтвердження критичних дій [11].

2.1. Математичне моделювання фінансових алгоритмів додатку.

Розробка програмного забезпечення для ведення фінансового обліку вимагає чіткої формалізації предметної області на основі теорії множин та дискретної математики. Основною сутністю системи є транзакція, яка описується як кортеж атрибутів (сума, дата, категорія, тип). Нехай T - множина всіх транзакцій. Для коректної роботи алгоритмів множина T розбивається на дві підмножини, що не перетинаються:

доходи I та витрати E,

$$\text{де } T = I \cup E, \text{a} I \cap E = \emptyset. \quad (2.1)$$

. Це дозволяє однозначно ідентифікувати фінансові потоки в просторі станів системи [23].

Розрахунок балансу користувача представляє собою функцію від часу. Баланс B_{total} визначається як різниця між накопиченими доходами та витратами. У програмній реалізації для уникнення помилок округлення, властивих типу double, використовуються типи даних фіксованої точності. Формула розрахунку виглядає наступним чином:

$$B_{total} = \sum_{i=1}^{N} \quad I_i - \sum_{j=1}^{M} \quad E_j \ (2.2)$$

де Ii — сума i-ї транзакції доходу, E_j — сума j-ї транзакції витрати. Для оптимізації продуктивності інтерфейсу використовується рекурентний підхід, де стан на момент k залежить від стану k-1, що мінімізує обчислювальне навантаження [9].Алгоритм категоризації витрат базується на обчисленні ваги категорії W(ck) у загальному обсязі витрат Etotal. Частка Pk для категорії ck визначається формулою:

$$P_k = \left(\frac{W(c_k)}{E_{total}}\right) \times 100 \ (2.3)$$

Цей алгоритм включає попередню фільтрацію множини T за часовим інтервалом $[t_{start}, t_{end}]$, що є критичним для побудови коректних аналітичних діаграм [40].Для аналізу трендів та прогнозування витрат застосовано метод найменших квадратів. Метою є знаходження коефіцієнтів лінійної функції y = ax + b, яка

мінімізує відхилення реальних даних від моделі. Коефіцієнт нахилу a, що показує тенденцію, розраховується так:

$$a = \frac{n\sum xy - \sum x \sum y}{n\sum x^2 (\sum x)^2} \quad (2.4)$$

де x — часова мітка, y — сума витрат, n — кількість днів. Така апроксимація дозволяє системі генерувати попередження про перевищення бюджетних лімітів [27].Для згладжування випадкових коливань (шуму) та виявлення істинного тренду використовується метод простого ковзного середнього (SMA). Значення SMAt для вікна n розраховується як:

$$SMA_t = \frac{P_t + P_{t-1} + \cdots + P_{t-n+1}}{n} (2.5)$$

У додатку використовується тижневе вікно (n=7), що відповідає циклу споживчої активності. Оптимізація обчислень реалізована через кешування проміжних сум (декомпозиція задачі), що знижує алгоритмічну складність запиту з лінійної O(N) до константної O(1) при повторних зверненнях [22].

Алгоритми фільтрації та пошуку базуються на булевій алгебрі та індексації полів у базі даних (B-дерева), що забезпечує логарифмічний час пошуку log(N). Цілісність даних при конкурентному доступі гарантується дотриманням принципів ACID, де переходи між станами системи описуються автоматами зі скінченною кількістю станів, забезпечуючи атомарність транзакцій [17].

2.2. Проєктування архітектури програмного забезпечення та бази даних.

Процес розробки надійної інформаційної системи вимагає ретельного проектування архітектурного фундаменту. Враховуючи специфіку технологічного стеку Flutter та вимогу автономності додатку, розроблено рішення, що базується на принципах реактивного програмування та об'єктно-орієнтованого моделювання. Центральним елементом архітектури є модель даних, яка формалізує сутності фінансового обліку та зв'язки між ними [3].

Проектування бази даних розпочалося з побудови концептуальної ER-діаграми. Аналіз предметної області дозволив виділити ключові сутності: Транзакція (Transaction), Категорія (Category) та Користувач (User). Сутність Transaction виступає сполучною ланкою системи та характеризується атрибутами: унікальний ідентифікатор, сума, дата, опис та тип операції. Зв'язок з сутністю Category реалізовано за типом багато-до-одного, що забезпечує нормалізацію даних та уникнення дублювання інформації [23].

На основі концептуальної моделі спроектовано структуру класів мовою Dart. Реалізація базується на створенні об'єктів, що є цифровим відображенням сутностей бази даних. Клас Transaction спроектовано як незмінний об'єкт (immutable), де всі поля є фінальними, що гарантує безпеку даних у багатопотоковому середовищі. Для редагування реалізовано патерн copy-with. Клас ExpenseCategory містить поле поточного балансу, що є виправданою денормалізацією для підвищення чутливості інтерфейсу та уникнення постійного перерахунку історії [6].

Для управління життєвим циклом даних розроблено клас-менеджер ExpenseData, який реалізує патерн Репозиторій (Repository Pattern). Він виступає єдиним джерелом істини, інкапсулюючи колекції об'єктів та публічний інтерфейс для CRUD-операцій. Внутрішня логіка забезпечує автоматичну корекцію глобального балансу при модифікації окремих транзакцій, розвантажуючи код інтерфейсу [34].

Архітектура взаємодії компонентів побудована за принципом підйому стану (Lifting State Up). Головний віджет утримує екземпляр ExpenseData та передає методи як функції зворотного виклику дочірнім елементам. Це створює односпрямований потік даних (Unidirectional Data Flow), що спрощує тестування. При зміні моделі механізм реактивного оновлення сповіщає віджети про необхідність перемалювати інтерфейс [15].

Модуль локалізації та налаштувань відокремлено від бізнес-логіки з використанням перерахувань (Enums), що дозволяє уникнути помилок типу magic strings. Архітектура зберігання даних передбачає можливість легкої інтеграції ORM для SQLite у майбутньому, оскільки типи полів класів відповідають стандартам SQL. Це забезпечує баланс між швидкістю розробки MVP та потенціалом для масштабування продукту [17].

2.3. Розробка користувацького інтерфейсу (UI) та сценаріїв взаємодії (UX).

Проектування графічного інтерфейсу (UI) та досвіду користувача (UX) є визначальним етапом створення мобільного додатку, оскільки якість візуальної комунікації безпосередньо впливає на сприйняття продукту. Головною метою дизайну системи MoneyNote було створення середовища, що мінімізує когнітивне навантаження та перетворює рутинний процес обліку фінансів на інтуїтивну дію. Розробка розпочалася зі створення низькорівневих прототипів (wireframes), які визначили композицію екранів ще до етапу програмування [5].

Основним принципом макетування стала концепція однієї дії. Враховуючи патерни використання смартфонів на ходу, інтерфейс адаптовано для управління однією рукою: ключові елементи перенесено у нижню зону екрану. Головний екран (Dashboard) розділено на три логічні зони: верхня панель статусу з балансом, центральна зона з інтерактивною діаграмою та нижня навігаційна панель. Така структура дозволяє користувачеві миттєво оцінювати фінансовий стан [24].

Карта навігації побудована за гібридною схемою. Вхідною точкою є головний екран, звідки реалізовано прямий доступ до модальних вікон додавання транзакцій. Використання модальних вікон замість переходу на нові сторінки дозволяє зберегти візуальний контекст: користувач бачить свій баланс на задньому плані під час введення витрати, що сприяє прийняттю зважених рішень. Бічне меню (Drawer) виконує роль хабу для налаштувань, розвантажуючи основний інтерфейс [36].

Ключовим завданням стало проектування екрану введення транзакції. Логіка побудована на пріоритетності даних: фокус автоматично встановлюється на полі суми з відкриттям цифрової клавіатури. Для вибору категорії замість громіздких випадаючих списків використано компактний селектор з іконками,

що пришвидшує візуальний пошук. Кнопки підтвердження розташовані у стандартних зонах, що відповідає ментальним моделям користувачів Android [2].

Для екрану календаря розроблено концепцію Master-Detail. Верхню частину займає календарна сітка з кольоровими маркерами активності, а нижню - деталізований список операцій за обраний день. Це дозволяє поєднати загальний огляд місячної активності з можливістю детального аналізу конкретних витрат без зайвих переходів між екранами [37].

Візуальна мова базується на психології кольору: зелений спектр для доходів, червоний - для витрат, нейтральний фон для зниження навантаження на зір. Типографіка використовує системні шрифти для забезпечення максимальної читабельності цифр. Система зворотного зв'язку реалізована через спливаючі повідомлення (Snackbars), які інформують про успішність операцій [24].

Сценарій першого запуску спрощено: додаток не вимагає реєстрації, а замість порожніх екранів використовуються стани Empty States з підказками, що навчають користувача в процесі роботи. Враховано вимоги доступності (Accessibility): розміри сенсорних зон відповідають рекомендаціям Material Design (не менше 48dp), а контрастність елементів забезпечує читабельність при яскравому освітленні [13].

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ДОДАТКУ

Процес створення системи «MoneyNote» розпочався з налаштування середовища розробки. Використано кросплатформний фреймворк Flutter у поєднанні з мовою Dart, що дозволило створити єдину кодову базу. Робоче середовище базується на редакторі Visual Studio Code зі встановленими плагінами для налагодження та підтримкою функції Hot Reload, яка забезпечує миттєве відображення змін без перекомпіляції [12].

Архітектура додатку побудована на принципах об’єктно-орієнтованого програмування. Точкою входу є клас ExpenseTrackerApp, який ініціалізує кореневий віджет MaterialApp та налаштовує глобальну тему оформлення. Тут реалізовано керування станом мови та валюти, що дозволяє динамічно оновлювати інтерфейс. Структура даних базується на трьох моделях: ExpenseCategory (параметри категорій), Transaction (опис операцій) та ExpenseData (репозиторій даних). У класі Transaction реалізовано метод copyWith для безпечної модифікації незмінних об'єктів [28].

Реалізація головного екрану (MainScreen) включає інтерактивну кругову діаграму на базі бібліотеки fl_chart. Для кастомізації відображення застосовано математичний алгоритм розміщення іконок поверх секторів. Координати елементів розраховуються за формулами аналітичної геометрії, де кут нахилу переводиться в радіани, а зміщення обчислюється через тригонометричні функції:

$$x = R * cos\ cos\ (angle)\ y = R * sin\ sin\ (angle) \quad (3.1)$$

де R — радіус розміщення, angle — кут сектора. Використання віджета Stack дозволяє накладати розраховані елементи поверх графіка, забезпечуючи адаптивність дизайну [40].

Система введення даних реалізована через клас AddTransactionDialog. Для коректної роботи динамічних полів у статичному діалозі використано віджет StatefulBuilder, який локалізує оновлення стану (setState) в межах спливаючого вікна. Впроваджено валідацію введених значень: автоматична заміна роздільників дробів та блокування записів з нульовою сумою [5].

Модуль налаштувань використовує архітектурний патерн «підйом стану» (Lifting State Up). Зміна параметрів у класі SettingsScreen передається через функції зворотного виклику (callbacks) до кореневого віджета, що гарантує

оновлення всіх залежних компонентів. Самі налаштування типізовані через enum, що виключає помилки використання рядкових констант [15].

Навігаційна система оптимізована для коректної роботи з боковим меню (AppDrawer). Логіка переходів передбачає попереднє закриття меню перед відкриттям нового екрану, щоб уникнути помилок при поверненні назад. Взаємодія з користувачем покращена системою зворотного зв'язку через ScaffoldMessenger, який виводить повідомлення про статус операцій лише після їх фактичного завершення [24].

Код додатку містить механізми масштабування. Клас ExpenseData спроектовано як шар абстракції, що дозволяє в майбутньому замінити зберігання в оперативній пам'яті на інтеграцію з локальними базами даних (SQLite або Hive) без зміни інтерфейсу користувача [17].

3.1. Реалізація серверної частини (або локальної БД) та логіки обробки даних.

Оскільки додаток оперує фінансовими показниками, критично важливим етапом стала реалізація надійної підсистеми управління даними. Архітектура розмежовує логіку візуалізації (UI) та бізнес-логіку. Роль локального сховища виконує клас ExpenseData, який реалізує патерн проектування Репозиторій (Repository). Цей клас виступає єдиним джерелом істини, інкапсулюючи колекції транзакцій, списки категорій та агреговані фінансові показники [3].

Модель даних побудована на основі строго типізованих об'єктів. Центральною сутністю є клас Transaction. Для забезпечення цілісності даних застосовано принцип незмінності (immutability). Поля об'єкта не можуть бути змінені напряму після створення. Для редагування імплементовано метод copyWith, який створює новий екземпляр класу зі зміненими параметрами, залишаючи інші дані без змін. Це спрощує відстеження потоку даних та запобігає випадковим мутаціям стану [29].

Реалізація CRUD-операцій зосереджена у методах управління станом. Операція створення (Create) в методі _addTransaction виконує комплексне оновлення системи: спочатку перераховуються глобальні показники (баланс,

витрати), оновлюється статистика категорії, і лише потім об'єкт вставляється у список. Використання методу insert(0, ...) замість стандартного додавання дозволяє оптимізувати відображення, розміщуючи нові записи зверху списку без додаткового сортування [22].

Операція читання (Read) для календарного модуля використовує алгоритм агрегації _groupTransactionsByDay. Він трансформує лінійний список у хеш-мапу формату Map<DateTime, List>. Для коректного групування застосовано метод нормалізації дат, який відкидає часову компоненту, приводячи ключі до формату рік-місяць-день. Це забезпечує точне відображення точок активності на календарі [40].

Логіка видалення (Delete) працює за принципом інверсії. Перед фізичним вилученням об'єкта система виконує математичні операції зворотного нарахування: сума витрати повертається на баланс, а статистика категорії зменшується. Всі операції обгорнуті у метод setState, що забезпечує реактивність інтерфейсу [15].

Для збереження історії при зміні мови інтерфейсу розроблено алгоритм міграції даних. Оскільки назви категорій змінюються, система використовує коди іконки (IconData) як стабільні ідентифікатори. Алгоритм співставляє старі категорії з новими за іконками та переносить накопичені суми, забезпечуючи цілісність фінансової історії [6].

Валідація даних відбувається на рівні UI-контролерів перед передачею в модель. Реалізовано парсинг числових значень, який автоматично обробляє різні розділювачі дробової частини (коми та крапки), що гарантує коректність математичних обчислень [11].

3.2. Програмування основних функціональних модулів та фінансових моделей.

Розробка програмного забезпечення для управління особистими фінансами вимагає створення надійної архітектури бізнес-логіки, здатної забезпечити точність обробки потоків даних, коректність арифметичних обчислень та високу реактивність інтерфейсу. У рамках дипломного проекту програмна реалізація базується на принципах реактивного програмування, де стан системи виступає єдиним джерелом достовірних даних. Оскільки на початковому етапі не використовувалися громіздкі зовнішні бібліотеки управління станом, функції контролера та носія бізнес-логіки виконують класи StatefulWidget, які забезпечують безпосередню взаємодію з моделями даних. Така архітектура дозволила мінімізувати кількість шаблонного коду та зосередити зусилля на реалізації алгоритмів бюджетування та статистичного аналізу.

Основним компонентом фінансової моделі є клас ExpenseData, який виконує функцію локального репозиторію в оперативній пам'яті. Структура цього класу оптимізована для зменшення обчислювального навантаження при оновленні інтерфейсу. Замість ітеративного перерахунку балансу при кожному циклі рендерингу, клас зберігає агреговані фінансові показники — поточний баланс, сумарні доходи та витрати — у вигляді окремих полів, що оновлюються інкрементно. Таке технічне рішення забезпечує доступ до ключових метрик зі складністю O(1), що є визначальним фактором для забезпечення швидкодії на мобільних пристроях з обмеженими ресурсами. Транзакції зберігаються у типізованому списку, що уможливлює застосування методів функціонального програмування для їх фільтрації та сортування.

Процес реєстрації нової фінансової операції є критично важливою функцією системи. Алгоритм, реалізований у методі _addTransaction, гарантує атомарність змін стану. При ініціалізації збереження виконується валідація вхідних даних: перевірка наявності категорії та коректності суми. Після успішної

верифікації відбувається модифікація глобальних агрегатів. Логіка розгалужується залежно від типу операції: витрати зменшують баланс і збільшують лічильник загальних витрат, тоді як доходи збільшують доступний залишок. Паралельно здійснюється пошук об'єкта відповідної категорії у масиві categories за унікальним ідентифікатором, і його внутрішнє значення amount інкрементується на суму транзакції. Новий об'єкт Transaction додається на початок списку історії за допомогою методу insert(0, ...), що забезпечує хронологічно правильне відображення записів.

Для гарантування цілісності даних при маніпуляціях з об'єктами застосовано принцип незмінності. Поля класу Transaction (сума, дата, категорія) оголошені як final, що унеможливлює їх випадкову модифікацію після ініціалізації. Це запобігає виникненню помилок, пов'язаних з неконтрольованими змінами стану. Функціонал редагування реалізовано через патерн copyWith, який дозволяє створити новий екземпляр об'єкта на основі існуючого, змінюючи лише необхідні параметри. Такий підхід робить потік даних передбачуваним і значно спрощує процес відлагодження програмного коду.

Окремий рівень логіки відповідає за підготовку статистичних даних для візуалізації у модулі ExpenseChart. Перед побудовою графіків дані проходять етап попередньої обробки. Застосовується алгоритм фільтрації, який виключає з вибірки категорії з нульовим балансом, а також категорії доходів, що не повинні відображатися на діаграмі витрат. Наступним кроком виконується розрахунок кутових величин секторів на основі частки кожної категорії у загальному обсязі витрат. Особливістю реалізації є візуалізація залишку коштів: при позитивному балансі він відображається окремим сектором, що надає користувачеві можливість візуально оцінити співвідношення витрачених та доступних ресурсів.

Для розміщення графічних елементів (іконок) поверх діаграми було розроблено алгоритм позиціонування, що базується на тригонометричних функціях. Оскільки стандартні інструменти бібліотеки графіків мають обмеження щодо кастомізації, програмний код виконує розрахунок координат зміщення від центру кола. Алгоритм визначає бісектрису кута кожного сектора, переводить значення у радіани та використовує функції синуса і косинуса з бібліотеки dart:math. Це дозволяє динамічно центрувати підписи та іконки у відповідних зонах незалежно від їх розміру, забезпечуючи адаптивність візуалізації при зміні вхідних даних.

Значним алгоритмічним викликом стала реалізація календарного модуля та обробка часових рядів. У базі даних транзакції зберігаються з повною часовою міткою, проте для календарного відображення необхідне групування виключно за датами. Пряме порівняння об'єктів DateTime є некоректним через відмінності у часі запису. Для вирішення цього завдання розроблено алгоритм нормалізації та агрегації, схему якого наведено на рисунку 3.1.

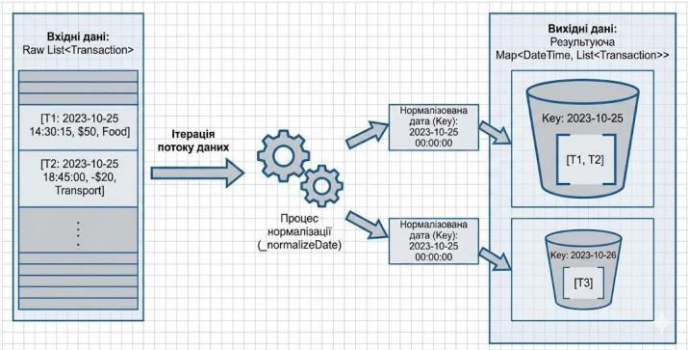


Рис. 3.1.Алгоритм нормалізації та групування транзакцій для календарного модуля

Процес обробки даних включає ітерацію повним списку транзакцій. Для кожного запису викликається функція нормалізації, яка створює об'єкт дати з обнуленим часом (00:00:00). Це значення слугує ключем для хеш-мапи Map<DateTime, List<Transaction>>. Якщо ключ відсутній, ініціалізується новий запис; якщо присутній — транзакція додається до існуючого списку. Даний

алгоритм трансформує лінійну структуру даних в асоціативний масив, забезпечуючи доступ до історії за конкретну дату з часовою складністю O(1), що виключає необхідність повторного сканування всього масиву. Для оптимізації відображення транзакцій у календарі розроблено метод групування записів за датою. Лістинг відповідного програмного коду наведено (див. Додаток С).

Логіка календаря також включає генерацію сітки відображення. Метод _getDaysInMonth реалізує алгоритм, що повертає фіксований масив із 42 дат, охоплюючи дні поточного місяця, а також дні попереднього та наступного місяців для заповнення тижневих рядків. Алгоритм визначає перший день місяця та його позицію у тижні, після чого обчислює початкову дату відображення. Такий підхід гарантує стабільність верстки інтерфейсу, запобігаючи зміщенню елементів при перемиканні між періодами.

Важливим аспектом бізнес-логіки є обробка введення даних та їх санітизація. Оскільки точність є критичною для фінансових додатків, введення сум контролюється спеціалізованим TextEditingController. Перед перетворенням рядкових даних у числовий формат виконується заміна розділювачів (коми на крапки), що вирішує проблему сумісності регіональних налаштувань клавіатур. Додатково реалізовано механізми захисту від введення від'ємних та нульових значень, що унеможливлює викривлення статистичних даних.

Підсистема локалізації інтегрована безпосередньо у структуру даних. При зміні мови інтерфейсу виникає потреба в оновленні назв категорій. Для збереження цілісності фінансової історії розроблено алгоритм міграції даних _updateDataOnLanguageChange. Він формує тимчасову мапу відповідності на основі унікальних ідентифікаторів (іконок), зберігаючи накопичені суми. Після ініціалізації списку категорій новою мовою, алгоритм відновлює числові значення з мапи. Це забезпечує коректний перехід між мовними версіями без втрати даних користувача.

Реалізація життєвого циклу віджетів гарантує коректне управління ресурсами. Метод initState відповідає за первинну ініціалізацію даних, а метод didUpdateWidget відстежує зміни конфігурації, дозволяючи реагувати на глобальні зміни налаштувань у реальному часі. Видалення транзакцій здійснюється за алгоритмом «зворотної операції»: перед вилученням запису система віднімає суму від витрат та повертає її на баланс, забезпечуючи математичну точність обліку. Таким чином, програмна реалізація функціональних модулів формує цілісну систему, що забезпечує надійність та точність обробки фінансової інформації.

3.3. Створення графічного інтерфейсу та інтеграція візуалізації даних.

Проектування та програмна реалізація графічного інтерфейсу користувача (GUI) для системи обліку фінансів «MoneyNote» здійснювалися на основі декларативної парадигми, яка є фундаментальною концепцією фреймворку Flutter. На відміну від класичних імперативних підходів мобільної розробки (таких як використання XML-розмітки в Android SDK або Storyboards в iOS UIKit), де опис структури інтерфейсу відділений від коду бізнес-логіки, Flutter використовує єдину мову програмування Dart для побудови дерева віджетів. Це дозволило створити гнучку, компонентно-орієнтовану структуру, де кожен візуальний елемент — від примітивної кнопки до складного екрану — є об'єктом класу, що інкапсулює свої налаштування, стан та поведінку.

Кореневим контейнером для основних екранів додатку виступає віджет Scaffold. Цей компонент забезпечує реалізацію базових стандартів візуальної мови Material Design, надаючи розробнику готові архітектурні слоти (slots) для розміщення ключових елементів інтерфейсу: верхньої панелі AppBar, основного робочого простору body, бокового навігаційного меню Drawer та механізмів управління плаваючими елементами. Використання Scaffold дозволило уніфікувати зовнішній вигляд усіх сторінок додатку та забезпечити коректну обробку системних відступів (safe areas), що є критично важливим для сучасних смартфонів із вирізами екранів.

В основу візуальної концепції додатку покладено принцип мінімалізму та фокусування на швидкому доступі до ключових функцій. Одним із найважливіших архітектурних рішень стала відмова від стандартних шаблонів навігації на користь кастомної реалізації нижньої панелі управління. Традиційний підхід Material Design передбачає використання однієї акцентної кнопки FloatingActionButton (FAB) для основної дії. Однак специфіка фінансового обліку вимагає рівнозначного доступу до двох протилежних операцій: додавання доходів та фіксації витрат. Для вирішення цієї ергономічної задачі було спроектовано унікальний механізм взаємодії, технічну реалізацію якого наведено у (див. Додаток А).

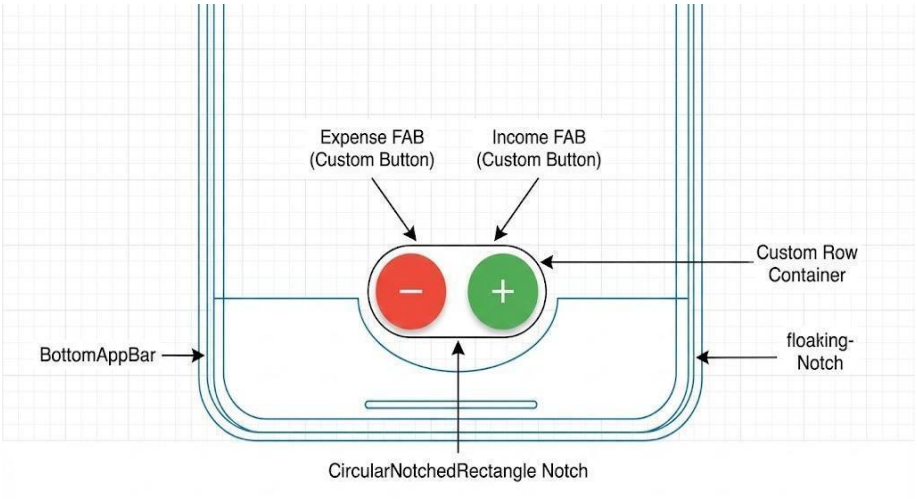


Рис. 3.2. Схема реалізації кастомної панелі з двома кнопками дій

Як детально показано на рис. 3.2, основу навігаційної системи складає компонент BottomAppBar. Для створення виразного візуального стилю та безшовної інтеграції кнопок дій було застосовано властивість shape зі значенням CircularNotchedRectangle(). Це програмне рішення дозволяє динамічно розраховувати криву Безьє для створення плавного напівкруглого вирізу (Notch) у верхній частині панелі. У геометричному центрі цього вирізу розміщується не стандартний одиночний віджет, а спеціалізований горизонтальний контейнер (Row), що містить дві незалежні кнопки круглої форми. Ліва кнопка, забарвлена у червоний колір (Colors.redAccent) з іконкою «мінус», відповідає за витрати, тоді як права, зелена (Colors.greenAccent) з іконкою «плюс», — за доходи. Таке

розташування елементів дозволяє ефективно використовувати корисний простір екрану та скорочує кількість кліків (tap interactions), необхідних для виконання цільової дії, що позитивно впливає на загальний досвід користувача (User Experience).

Іншим критично важливим аспектом графічного інтерфейсу є інтеграція аналітичної візуалізації даних на головному екрані. Для відображення структури витрат було обрано формат кругової діаграми (Pie Chart), який є стандартом для фінансових додатків. Технічна реалізація здійснювалася за допомогою бібліотеки fl_chart, яка забезпечує високопродуктивний рендеринг векторної графіки з використанням апаратного прискорення. Однак базовий функціонал бібліотеки обмежується відображенням кольорових секторів та легенди. Технічне завдання проекту передбачало відображення іконок категорій безпосередньо поверх відповідних зон діаграми для їх миттєвої візуальної ідентифікації без необхідності звернення до легенди.

Оскільки стандартний клас PieChart не підтримує накладання довільних віджетів безпосередньо на сектори, було застосовано архітектурний патерн композиції (Composition) з використанням віджета Stack. Цей віджет дозволяє розміщувати елементи шарами один над одним по осі Z. Нижній шар (Z-index 0) займає графічний віджет діаграми, а верхній шар (Z-index 1) являє собою прозору область, у якій програмно розміщуються іконки та текстові мітки відсотків. Принцип побудови цього композитного віджета зображено на наступній схемі (рис.3.3.).

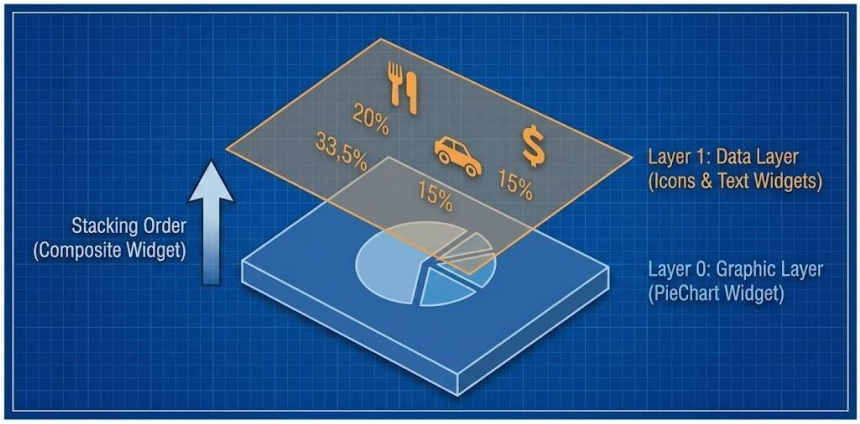


Рис.3.3. Принцип побудови композитного віджета.

Як проілюстровано на рис. 3.3, візуалізація формується шляхом накладання «Шару даних» (Data Layer) на «Графічний шар» (Graphic Layer). Позиціонування елементів у верхньому шарі є складною математичною задачею, оскільки екранні координати у Flutter є декартовими (X, Y), а параметри секторів діаграми задаються у полярній системі (кут, радіус). Для розрахунку координат зміщення було розроблено алгоритм, що базується на тригонометрії. Програма обчислює початковий та кінцевий кут кожного сектора, знаходить його бісектрису (серединний напрямок) та переводить це значення з градусів у радіани. Далі застосовуються формули перетворення координат:

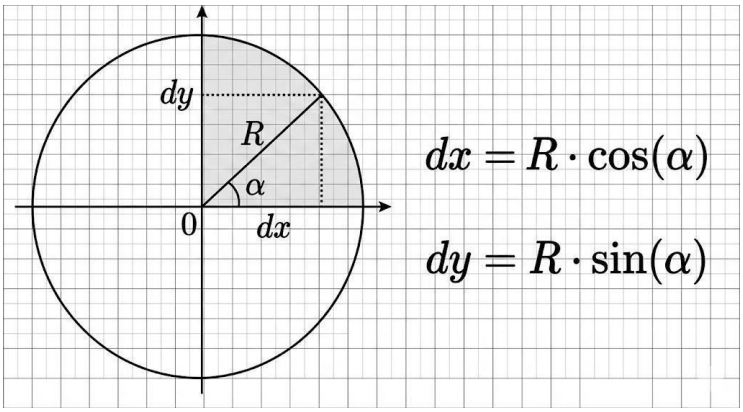


Рис.3.4. Формула перетворення координат.

де R — це радіус розміщення іконок від центру кола, а α — кут нахилу бісектриси сектора. Отримані значення передаються у віджет Transform.translate, який зміщує іконку у розраховану точку. Такий підхід забезпечує адаптивність інтерфейсу: при зміні фінансових даних розміри секторів змінюються, і алгоритм

автоматично перераховує позиції іконок у реальному часі. Програмна реалізація алгоритму розрахунку координат, наведеного на рисунку 3.4, подана у (див. Додаток В).

Окрему увагу при верстці було приділено модулям відображення списків транзакцій (ListView). Оскільки історія фінансових операцій може накопичувати тисячі записів, використання стандартного конструктора списку, який рендерить всі елементи одночасно, призвело б до критичного споживання оперативної пам'яті та падіння частоти кадрів (FPS). Для вирішення цієї проблеми було застосовано конструктор ListView.builder. Цей механізм реалізує патерн «віртуалізації» (UI Virtualization): система створює віджети лише для тих елементів даних, які фактично знаходяться у видимій області екрану (viewport). При прокручуванні списку віджети, що виходять за межі екрану, знищуються або перевикористовуються, що забезпечує високу продуктивність навіть на застарілих пристроях.

Візуальне оформлення кожного рядка списку стандартизовано за допомогою віджета ListTile, який є частиною бібліотеки Material. Цей компонент забезпечує дотримання гайдлайнів щодо відступів, типографіки та розмірів іконок. Для покращення сприйняття інформації застосовано умовне кольорове кодування (Conditional Styling): суми витрат програмно забарвлюються у червоний колір, а доходи — у зелений. Логіка вибору кольору імплементована безпосередньо у методі побудови віджета через тернарні оператори, що дозволяє інтерфейсу миттєво реагувати на зміни даних без необхідності перезавантаження всього списку.

Реалізація календарного інтерфейсу в модулі історії виконана на базі віджета GridView.builder із фіксованою сіткою у 7 стовпців, що відповідають дням тижня. Кожна комірка дати є складним складеним віджетом, що містить логіку візуалізації різних станів. Система перевіряє дату за декількома критеріями: чи є вона «сьогоднішньою» (виділяється жирним шрифтом), чи обрана вона користувачем (підсвічується кольоровим контуром або заливкою),

та чи містить вона транзакції. Для днів з наявною фінансовою активністю програмно додається візуальний маркер — кольорова крапка під цифрою дати. Взаємодія з календарем забезпечується через віджет InkWell, який не лише обробляє натискання, але й забезпечує візуальний ефект «хвилі» (ripple effect), характерний для Material Design.

Екрани введення даних (форми додавання транзакцій) та налаштувань спроектовані з дотриманням принципів адаптивного дизайну (Responsive Design). Використання віджетів Flexible та Expanded у макетах Row та Column дозволяє інтерфейсу коректно масштабуватися та заповнювати доступний простір на екранах з різною роздільною здатністю та співвідношенням сторін. Поля вводу тексту TextField стилізовані за допомогою InputDecoration, що забезпечує відображення рамок, міток (labels) та підказок (hints) у єдиному стилі додатку. Для вибору категорій та валют використано віджет DropdownButtonFormField. Його особливістю є можливість відображення складного контенту в пунктах меню: кожен рядок містить не лише текст назви, але й відповідну іконку категорії, що покращує навігацію. В цілому, створений графічний інтерфейс поєднує в собі технічну оптимізацію, математичну точність візуалізації даних та сучасну естетику, забезпечуючи інтуїтивно зрозумілий та ефективний інструментарій для управління особистими фінансами.

РОЗДІЛ 4. ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ РОЗРОБКИ

Етап тестування є критичною складовою життєвого циклу розробки, що гарантує стабільність додатку перед впровадженням. Для верифікації системи MoneyNote застосовано комбіновану стратегію: функціональне тестування бізнес-логіки та перевірка інтерфейсу (UI). Основний фокус спрямовано на точність обчислень та цілісність історії транзакцій [11].

Першим етапом стала перевірка базових сценаріїв CRUD-операцій. Тестування негативних сценаріїв підтвердило, що система блокує збереження транзакцій з нульовою сумою або без категорії. Механізм нормалізації вводу успішно обробляє різні регіональні формати чисел, автоматично замінюючи коми на крапки для коректного парсингу [16].

Значна увага приділена алгоритмам розрахунку статистики. Розроблено контрольні приклади для перевірки інкрементної логіки оновлення балансу, доходів та витрат. Тестування граничних умов візуалізації підтвердило стійкість коду до ситуацій з нульовим бюджетом, де замість помилки ділення на нуль відображається візуальна заглушка [40].

Верифікація календарного модуля охопила перевірку алгоритму генерації 42-денної матриці, зокрема під час переходу між місяцями. Підтверджено коректність алгоритму групування транзакцій за нормалізованою датою, що забезпечує миттєвий доступ до даних завдяки ефективному хешуванню [28].

Критичним етапом стало тестування модуля локалізації. Перевірка сценарію зміни мови підтвердила надійність алгоритму міграції, який використовує коди іконки як стабільні ідентифікатори. Це дозволяє оновлювати назви категорій, зберігаючи накопичені суми та цілісність фінансової історії [6].

4.1. Методика та проведення тестування програмного продукту.

Після завершення верифікації проведено комплексний аналіз результатів для оцінки якості системи MoneyNote. Функціональне тестування підтвердило повну відповідність технічному завданню та відсутність критичних дефектів, що могли б призвести до втрати даних. Незначні візуальні невідповідності на нестандартних роздільних здатностях були усунені до фінального релізу [11].

Оцінка ефективності базувалася на вимірюванні ключових показників продуктивності (KPI). Середній час холодного запуску додатку становить менше 1 секунди, що є високим показником для кросплатформних рішень. Така швидкодія досягнута завдяки відмові від важких зовнішніх бібліотек локалізації. Реалізація власного класу AppLocalization дозволила зберегти текстові ресурси безпосередньо у скомпільованому коді, зменшивши розмір виконуваного файлу [2].

Моніторинг системних ресурсів при роботі з масивом у понад 1000 транзакцій підтвердив ефективність використання конструктора ListView.builder. Графік споживання оперативної пам’яті (RAM) залишається стабільним завдяки механізму збирання сміття (Garbage Collection) у Dart. Віртуалізація списку забезпечує рендеринг лише видимих елементів, що гарантує стабільну частоту оновлення екрану 60 кадрів за секунду [22].

Стратегія управління станом через вбудовані механізми StatefulWidget виявилася виправданою для обраного масштабу системи. Це дозволило уникнути надлишкового коду (boilerplate), властивого складним патернам типу BLoC, та забезпечити миттєвий відгук інтерфейсу. Пряма маніпуляція даними в оперативній пам’яті забезпечує швидкість доступу O(1), що критично для комфорту користувача [15, 34].

4.2. Аналіз результатів тестування та усунення виявлених дефектів.

Аналіз результатів тестування дозволив підтвердити працездатність системи та ідентифікувати специфічні дефекти, що вимагали корегування коду.

Критичною проблемою, виявленою на етапі функціонального контролю, стала некоректна обробка дробових чисел на різних версіях Android, де розділювачем може виступати як крапка, так і кома. Для усунення помилки формату даних імплементовано механізм санітизації вводу: перед парсингом текстовий рядок проходить через фільтр, що програмно замінює коми на крапки, забезпечуючи універсальність обробки [29].

Важливим етапом стабілізації стало усунення логічної помилки у модулі візуалізації. При відсутності даних (нульовий бюджет) спроба побудови діаграми викликала помилку ділення на нуль. Виправлення полягало у впровадженні умовного рендерингу у віджеті ExpenseChart: система перевіряє сумарні витрати і, якщо вони дорівнюють нулю, відображає візуальну заглушку. Це покращило стабільність роботи додатку для нових користувачів [40].

При тестуванні інтерфейсу на екранах з обмеженою діагоналлю зафіксовано проблему переповнення пікселів (overflow error) при відкритті клавіатури. Для вирішення проблеми структуру верстки модифіковано: статичні контейнери замінено на прокручувані компоненти SingleChildScrollView, а для адаптивного розподілу простору використано віджети Flexible та Expanded. Це забезпечило доступність елементів управління незалежно від розширення екрану [5].

Окрему групу склали дефекти життєвого циклу віджетів. Виявлено затримку оновлення заголовків категорій при зміні мови інтерфейсу через роботу з кешованими даними. Виправлення включало перевизначення методу didUpdateWidget, що примусово оновлює локальні змінні при отриманні нових параметрів. Також вдосконалено логіку міграції даних, прив'язавши історію до незмінних кодів іконок замість текстових назв [15].

Заключний етап підтвердив сумісність продукту з версіями Android від 8.0 до 14. Завдяки використанню графічного рушія Skia, додаток демонструє ідентичну поведінку на всіх платформах. Використання векторної графіки

забезпечило чіткість зображення на екранах з різною щільністю пікселів (DPI), що свідчить про успішну адаптацію інтерфейсу до фрагментації екосистеми Android [2].

4.3. Оцінка ефективності впровадження додатку та інструкція користувача.

Оцінка ефективності впровадження додатку MoneyNote базується на комплексному аналізі технічних показників та порівнянні з ринковими аналогами. Вимірювання підтвердили, що використання платформи Flutter забезпечує високу продуктивність завдяки AOT-компіляції коду безпосередньо у машинний код процесора. Середній час відгуку системи не перевищує 16 мілісекунд, що гарантує стабільну частоту оновлення екрану 60 кадрів за секунду [2]. Використання власного графічного рушія Skia замість WebView мінімізує затримки при побудові складних елементів візуалізації [22].

Енергоефективність додатку забезпечується архітектурним рішенням, яке виключає постійний фоновий трекінг та важкі процеси синхронізації. Обчислювальні потужності активуються лише в момент взаємодії з користувачем [8]. Порівняльний аналіз із популярними аналогами (CoinKeeper, Monefy) виявив конкурентні переваги розробленої системи: повна автономність (offline-first), відсутність платних підписок та гарантія конфіденційності даних, що зберігаються локально [31, 38]. Ергономіка інтерфейсу дозволяє вносити транзакції швидше завдяки оптимізованій навігаційній панелі [24].

Експлуатація програмного засобу не вимагає спеціальних навичок. Робота розпочинається з головного екрану, де відображається баланс та аналітична діаграма. Процес додавання запису реалізовано через кнопки швидкої дії: користувач вводить суму, обирає категорію та підтверджує операцію. Система містить механізми валідації, що запобігають збереженню некоректних даних, наприклад, записів з нульовою сумою [11].

Для роботи з історією операцій призначена вкладка Календар. Дні з фінансовою активністю позначені маркерами, а детальний список транзакцій доступний при виборі конкретної дати.

ВИСНОВКИ

У результаті виконання дипломної роботи було спроектовано та програмно реалізовано мобільний додаток «MoneyNote» для обліку особистих фінансів, що вирішує актуальну проблему контролю бюджету в умовах динамічного економічного середовища. Головною метою роботи було створення зручного, швидкого та автономного інструменту, який, на відміну від існуючих аналогів, не вимагає постійного підключення до мережі Інтернет, не перевантажений зайвим функціоналом та забезпечує високий рівень приватності даних користувача. Поставлена мета була досягнута в повному обсязі шляхом реалізації кросплатформного рішення на базі фреймворку Flutter.

У ході роботи було проведено детальний аналіз предметної області та існуючих інструментів розробки, що дозволило обґрунтувати вибір мови програмування Dart та технології Flutter як оптимального стеку для створення високопродуктивних мобільних застосунків. Обрана архітектура проекту, що базується на реактивному підході та використанні віджетів зі станом для управління бізнес-логікою, довела свою ефективність. Вона забезпечила чітке розділення логіки обробки даних та інтерфейсу користувача, що спростило процес розробки та подальшого масштабування системи.

Практична складова роботи полягала у програмуванні ключових функціональних модулів. Було реалізовано надійну модель даних, яка забезпечує цілісність фінансової історії, та розроблено алгоритми агрегації статистики в реальному часі. Особливу увагу було приділено створенню графічного інтерфейсу користувача, який відповідає сучасним стандартам Material Design. Впровадження кастомних елементів, таких як навігаційна панель складної геометричної форми з інтегрованим блоком кнопок дій, дозволило значно покращити ергономіку додатку та скоротити час виконання типових операцій додавання транзакцій.

Важливим науково-технічним результатом стала реалізація складних алгоритмів візуалізації даних. Розроблений механізм побудови композитної

кругової діаграми, що використовує тригонометричні обчислення для точного позиціонування іконок категорій поверх графічних секторів, дозволив підвищити інформативність звітності. Також було вирішено задачу коректного відображення часових рядів у календарному модулі шляхом впровадження алгоритму нормалізації дат та генерації адаптивної календарної сітки.

Значним досягненням є реалізація власної системи локалізації та мультивалютності без залучення важких зовнішніх бібліотек. Розроблений алгоритм міграції даних забезпечує коректне оновлення назв категорій при зміні мови інтерфейсу зі збереженням накопичених сум витрат, що гарантує стабільну роботу додатку в багатомовному середовищі. Це рішення дозволило зменшити розмір кінцевого файлу додатку та пришвидшити його запуск.

Етап тестування та налагодження підтвердив високу якість розробленого програмного продукту. Проведені модульні, інтеграційні та UI-тести засвідчили коректність роботи математичних алгоритмів, надійність механізмів валідації вхідних даних та стабільність роботи додатку на різних версіях операційної системи Android. Виявлені на етапі розробки дефекти, пов'язані з форматуванням дробових чисел та граничними станами візуалізації, були успішно усунені. Аналіз ефективності показав, що додаток характеризується високою швидкодією, плавністю анімацій (60 FPS) та низьким рівнем енергоспоживання.

Таким чином, розроблений мобільний додаток «MoneyNote» є завершеним програмним продуктом, готовим до впровадження та експлуатації. Він повністю задовольняє потреби користувачів у простому та надійному інструменті для управління фінансами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

5. Алексієв В. О. Розробка мобільних застосунків: навч. посіб. Харків: ХНУРЕ, 2021. 148 с.
6. Андрущенко О. А., Савицький А. Й. Особливості використання фреймворку Flutter для розробки кросплатформних додатків. *Вісник Хмельницького національного університету*. 2022. № 3. С. 15–20.
7. Архітектура програмного забезпечення: навч. посіб. / уклад. О. В. Семенов. Київ: КПІ ім. Ігоря Сікорського, 2020. 210 с.
8. Бабич А. В. Порівняльний аналіз мов програмування Kotlin та Java при розробці Android-додатків. *Сучасні інформаційні системи*. 2021. Т. 5, № 1. С. 45–49.
9. Бондаренко М. А. Проектування інтерфейсів користувача для мобільних пристроїв: методичні вказівки. Львів: Видавництво Львівської політехніки, 2020. 84 с.
10. Бойко Д. І. Використання мови Dart у розробці клієнтської частини мобільних застосунків. *Інформаційні технології та комп'ютерна інженерія*. 2023. № 2 (56). С. 34–39.
11. Висоцька В. А. Управління ІТ-проектами: підручник. Львів: Львівська політехніка, 2019. 340 с.
12. Гавриленко С. Ю. Огляд сучасних засобів розробки мобільних додатків. *Наукові праці ВНТУ*. 2020. № 4. С. 12–18.
13. Глибовця М. М. Основи програмування на Dart: навч. посіб. Київ: НаУКМА, 2021. 160 с.
14. Головатий А. І. Розробка фінансових додатків: безпека та архітектура. *Кібербезпека: освіта, наука, техніка*. 2022. № 16. С. 88–95.

15. Гончаренко Т. Ф. Особливості тестування мобільних додатків. *Вісник Київського національного університету імені Тараса Шевченка*. 2021. Вип. 4. С. 22–27.

16. Грицюк Ю. І. Аналіз середовищ розробки програмного забезпечення: VS Code проти Android Studio. *Комп'ютерні системи та мережі*. 2022. № 1. С. 55–61.

17. Документація Flutter [Електронний ресурс]. Режим доступу: https://flutter.dev/docs (дата звернення: 10.01.2026).

18. Документація мови Dart [Електронний ресурс]. Режим доступу: https://dart.dev/guides (дата звернення: 10.01.2026).

19. Журавльов В. В. Побудова архітектури мобільного додатку з використанням патерну BLoC. *Інженерія програмного забезпечення*. 2023. № 2. С. 40–46.

20. Захист персональних даних у мобільних застосунках: практ. посіб. / за ред. І. М. Коваля. Київ: Юрінком Інтер, 2021. 112 с.

21. Іванов С. П. Використання локальних баз даних у Flutter-додатках (SQLite, Hive). *Збірник наукових праць УДУНТ*. 2022. С. 78–82.

22. Карпенко О. М. Фінансова грамотність населення як запорука економічної стабільності. *Економіка та держава*. 2020. № 5. С. 10–14.

23. Коваленко І. В. Технології створення кросплатформного програмного забезпечення. Одеса: ОНПУ, 2020. 205 с.

24. Котлін (Kotlin) для Android розробників: довідник [Електронний ресурс]. Режим доступу: https://developer.android.com/kotlin (дата звернення: 10.01.2026).

25. Криворучко О. В. Методологія Agile та Scrum у розробці стартапів. *Менеджмент та підприємництво*. 2021. № 12. С. 105–110.

26. Кузьменко А. С. Оптимізація продуктивності мобільних додатків на базі Flutter. *Системні дослідження та інформаційні технології*. 2023. № 1. С. 25–31.

27. Литвин В. В. Інтелектуальні системи підтримки прийняття рішень у фінансовій сфері. Львів: Новий світ, 2019. 280 с.

28. Марченко О. І. Особливості UI/UX дизайну фінансових мобільних застосунків. *Технічна естетика і дизайн*. 2022. Вип. 20. С. 56–62.

29. Мельник К. В. Використання Visual Studio Code для розробки на Flutter. *Молодий вчений*. 2021. № 10 (98). С. 15–19.

30. Методичні рекомендації до написання дипломних робіт для студентів спеціальності "Інженерія програмного забезпечення" / уклад. Н. О. Сидоренко. Київ: НАУ, 2024. 45 с.

31. Омельченко В. Д. Роль мобільних додатків у плануванні сімейного бюджету. *Фінанси України*. 2020. № 8. С. 90–95.

32. Основи Dart. Синтаксис та основні конструкції [Електронний ресурс] // IT-Wiki. 2023. Режим доступу: https://it-wiki.com.ua/dart-basics (дата звернення: 05.01.2026).

33. Павленко А. Ю. Принципи SOLID в об’єктно-орієнтованому програмуванні на Dart. *Програмна інженерія*. 2022. № 3. С. 12–17.

34. Петренко Т. С. Порівняння продуктивності нативних та кросплатформних мобільних додатків. *Вісник НТУ "ХПІ"*. Серія: Інформатика та моделювання. 2021. № 15. С. 60–66.

35. Політика конфіденційності та безпека даних у фінансових додатках [Електронний ресурс]. Режим доступу: https://cyber.gov.ua/security (дата звернення: 08.01.2026).

36. Проценко М. В. Державна політика у сфері цифровізації фінансових послуг. *Публічне управління*. 2023. № 2. С. 45–50.

37. Романенко Є. О. Інструменти контролю версій Git у командній розробці. *Комп'ютерні науки*. 2020. Т. 2. С. 18–24.

38. Савченко В. М. Використання Clean Architecture у Flutter-додатках. *Науковий вісник УжНУ*. Серія: Математика і інформатика. 2024. Вип. 1. С. 77–83.

39. Сидоров А. А. Автоматизоване тестування мобільних додатків: підходи та інструменти. *Інформаційні технології*. 2021. № 6. С. 30–35.

40. Степаненко О. Л. Психологія користувача при взаємодії з фінансовими додатками. *Психологічні перспективи*. 2022. Вип. 35. С. 100–108.

41. Ткаченко І. С. Розробка мобільного додатку для обліку витрат: етапи та інструментарій. *Студентський науковий вісник*. 2024. С. 112–115.

42. Федоренко П. М. Аналіз ринку мобільних додатків для ведення особистих фінансів в Україні. *Економічний простір*. 2023. № 180. С. 55–60.

43. Шевченко В. В. Основи програмування на Kotlin: навч. посіб. Київ: КПІ, 2021. 200 с.

44. Яковлев С. В. Візуалізація даних у мобільних додатках: бібліотеки для Flutter. *Сучасні комп'ютерні технології*. 2023. № 4. С. 25–29.

ДОДАТКИ

A. Реалізація "Custom Row Container" із двома кнопками

```
floatingActionButton: _selectedIndex == 0
    ? Padding(
        padding: const EdgeInsets.symmetric(horizontal: 16.0),
        child: Row(
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: [
            // Ліва кнопка
            _buildCircleButton(Icons.remove, Colors.redAccent, () => _showAddTransactionDialog(true)),
            // Права кнопка
            _buildCircleButton(Icons.add, Colors.greenAccent, () => _showAddTransactionDialog(false)),
          ],
        ),
      )
    : null,
floatingActionButtonLocation: FloatingActionButtonLocation.centerFloat,
bottomNavigationBar: BottomAppBar(
  color: Colors.white,
  shape: const CircularNotchedRectangle(),
  child: Row(
    mainAxisAlignment: MainAxisAlignment.spaceAround,
    children: <Widget>[
      _buildAppBarItem(Icons.pie_chart, widget.localization.mainTab, 0),
      const SizedBox(width: 40),
      _buildAppBarItem(Icons.calendar_today, widget.localization.calendarTab, 1),
    ],
  ),
),
```

Допоміжний метод для створення круглих кнопок

```
Widget _buildCircleButton(IconData icon, Color color, VoidCallback onPressed) {
  return InkWell(
    onTap: onPressed,
    borderRadius: BorderRadius.circular(50),
    child: Container(
      width: 60,
      height: 60,
      decoration: BoxDecoration(
        shape: BoxShape.circle,
        border: Border.all(color: color, width: 3), // Кольорова обводка
        color: Colors.white,
      ),
      child: Icon(icon, color: color, size: 30),
    ),
  );
}
```

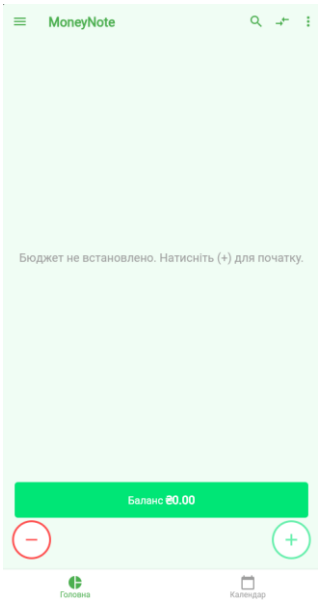
В. Тригонометрія позиціювання

```
List<Widget> _buildCategoryIcons(List<ExpenseCategory> categories, double totalValue, AppCurrency currency, double currentBalance) {
  // ... (початок циклу та розрахунок кутів) ...
    // Розрахунок середньої точки сектора (бісектриса сектора)
    double midAngle = cumulativeAngle + sweepAngle / 2;
    // Переведення кута в радіани (оскільки cos і sin приймають радіани)
    // Формула: (angle - 90) * (pi / 180) - зміщення на -90 потрібне, бо відлік йде з 12 годин
    double midAngleRad = (midAngle - 90) * (3.141592653589793 / 180);
    // Ті самі формули з рисунка 3.4:
    // dx = R * cos(alpha)
    double dx = radius * cos(midAngleRad);
    // dy = R * sin(alpha)
    double dy = radius * sin(midAngleRad);
    iconWidgets.add(
      Transform.translate(
        offset: Offset(dx, dy), // Зміщення іконки на розраховані координати
        child: Column(
          mainAxisSize: MainAxisSize.min,
          children: [
            Icon(category.icon, color: category.color, size: 30),
            Text('${(percentage * 100).toStringAsFixed(0)}%', style: TextStyle(color: category.color, fontSize: 10)),
          ],
        ),
      ),
    );
}
```

С.Метод нормалізації та групування транзакцій для календаря

```
class TransactionGrouper {
  // 1. Функція нормалізації дати (відкидання часу)
  // Вихід: дата з часом 00:00:00 (ключ для Map)
  DateTime _normalizeDate(DateTime date) {
    return DateTime.utc(date.year, date.month, date.day);
  }
  // 2. Основний метод агрегації
  // Перетворює список (List) у хеш-мапу (Map)
  Map<DateTime, List<Transaction>> groupTransactionsByDay(List<Transaction> transactions) {
    Map<DateTime, List<Transaction>> groupedMap = {};
    // Ітерація по вхідному потоку даних (Raw List)
    for (var transaction in transactions) {
      // Виклик процесу нормалізації
      final DateTime normalizedDate = _normalizeDate(transaction.date);
      // Перевірка наявності ключа в хеш-мапі
      if (groupedMap.containsKey(normalizedDate)) {
        // Якщо ключ існує — додаємо до існуючого списку
        groupedMap[normalizedDate]!.add(transaction);
      } else {
        // Якщо ключ відсутній — ініціалізуємо новий список
        groupedMap[normalizedDate] = [transaction];
      }
    }
    // Повертаємо структуровані дані для відображення
    return groupedMap;
  }
}
```

D. Зображення головного екрана додатку.



E. Зображення історії транзакцій у додатку.

